HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Analyse der DevOps-Praktiken in Laravel-basierten Anwendungen – Herausforderungen, Möglichkeiten und Sicherheitsaspekte

Jon-Steven Streller

Master-Arbeit im Studiengang „Angewandte Informatik“

14. Dezember 2025

Autor Jon-Steven Streller
1717126
steven@streller.net

Erstprüfer: Prof. Dr. Stefan Wohlfeil
Abteilung Informatik, Fakultät IV
Hochschule Hannover
stefan.wohlfeil@hs-hannover.de

Zweitprüfer: Prof. Dr. Matthias Hovestadt
Abteilung Informatik, Fakultät IV
Hochschule Hannover
matthias.hovestadt@hs-hannover.de

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Master-Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 14. Dezember 2025 Unterschrift

Inhaltsverzeichnis

1 Einleitung **10**
- 1.1 Hintergrund und Motivation . . . 10
- 1.2 Problemstellung . . . 10
- 1.3 Zielsetzung der Arbeit . . . 11
- 1.4 Forschungsfragen . . . 11
- 1.5 Vorgehensweise und Methodik . . . 12
- 1.6 Aufbau der Arbeit . . . 12
- 1.7 Ergebnisse in Kürze – Antworten auf die Forschungsfragen . . . 12

2 Grundlagen **14**
- 2.1 Laravel im Kontext von DevOps . . . 14
- 2.2 DevOps und DevSecOps – Prinzipien und Praktiken . . . 15
- 2.3 CI/CD und GitOps als Basis moderner Deployments . . . 16
- 2.4 Containerisierung und Orchestrierung mit Kubernetes . . . 18

3 Anforderungsanalyse **21**
- 3.1 Ausgangslage bei der DLRG . . . 21
- 3.2 Identifizierte Schwächen . . . 25
- 3.3 Abgeleitete Anforderungen . . . 26

4 Transformation der DLRG-Anwendung „Kindersucharmband“ **29**
- 4.1 Einordnung und Zielbild der Anwendung „Kindersucharmband“ . . . 29
  - 4.1.1 Ablauf und Nutzung . . . 29
  - 4.1.2 Bedeutung für die Transformation . . . 31
- 4.2 Zielprozess auf Basis von DevOps-Praktiken . . . 32
  - 4.2.1 Git-Repository und Branching-Strategie . . . 32
  - 4.2.2 Lokale Entwicklungsumgebung mit Containern . . . 32
  - 4.2.3 CI/CD-Pipeline mit Qualitätssicherung und Sicherheitstests . . . 33
  - 4.2.4 Zusammenfassung . . . 38
  - 4.2.5 Deployment nach Kubernetes . . . 40
- 4.3 Produktivumgebung nach der Umstellung . . . 49
  - 4.3.1 Containerisierung mit Multistage-Builds . . . 49
  - 4.3.2 Deploymentstrategien (Rolling Updates, Feature Flags) . . . 51
  - 4.3.3 Geheimnisverwaltung mit HashiCorp Vault . . . 56

4.3.4 Observability und kontinuierliche Laufzeitüberwachung 60
4.3.5 Laufzeitüberwachung und Compliance-Prüfung 63
4.4 Zusammenfassung der Transformation und Vorher/Nachher-Überblick . . 65

5 Evaluation **67**
5.1 Evaluationsprinzip: Anforderungen als Hypothesen 67
5.2 Auswertung der Anforderungen 67
5.2.1 R1: Reproduzierbarer Build (H_{R1}) 67
5.2.2 R2: Signatur & Provenance (H_{R2}) 68
5.2.3 R3: Auditierbarkeit (H_{R3}) 69
5.2.4 R4: GitOps-Deployment (H_{R4}) 70
5.2.5 R5: Zero-Downtime-Rollouts (H_{R5}) 71
5.2.6 R6: Integriertes Secret Management (H_{R6}) 72
5.2.7 R7: Secret-Rotation und Leases (H_{R7}) 73
5.2.8 R8: Netzwerkisolation & Default-Deny (H_{R8}) 75
5.2.9 R9: Restriktiver `securityContext` & PSA *restricted* (H_{R9}) 77
5.2.10 R10: RBAC nach Least-Privilege (H_{R10}) 79
5.2.11 R11: Reduzierte Runtime-Image-Größe (H_{R11}) 80
5.2.12 R12: Reduktion der gesamten CVE-Anzahl im Runtime-Image (H_{R12}) 81
5.2.13 R13: Automatisierter Release-Flow (max. 1 manueller Schritt) (H_{R13}) 82
5.2.14 R14: Automatisierte, idempotente Migration & Seeding (H_{R14}) . . 83
5.2.15 R15: Beobachtbarkeit & SLOs (H_{R15}) 85
5.2.16 R16: Feature-Flag-gestützte Releases (H_{R16}) 86
5.2.17 R17: Rollback-Fähigkeit (H_{R17}) 86
5.2.18 R18: Prozess-Isolation & dedizierte Backing Services (H_{R18}) . . . 88
5.2.19 R19: Modernisierter Entwicklungsfluss (H_{R19}) 89

6 Zusammenfassung und Fazit **92**
6.1 Zusammenfassung der Ergebnisse 92
6.2 Fazit zur Integration von Dev(Sec)Ops in Laravel 92
6.3 Ausblick auf zukünftige Entwicklungen und Forschungsfelder 93

7 Anhang **94**
7.1 Lokale Entwicklungsumgebung 94
7.1.1 Makefile 94
7.1.2 Git Hooks 94
7.2 CI/CD-Workflows und wiederverwendbare Actions (GitHub Actions) . . 95
7.2.1 Workflows 95
7.2.2 Wiederverwendbare Actions 103
7.3 Helm-Auszüge 109
7.3.1 Chart.yaml 109

7.3.2 values.yaml (exemplarisch) . . . 109
7.3.3 Deployment: App (PHP-FPM) . . . 115
7.3.4 Deployment: Worker . . . 118
7.3.5 CronJob: Scheduler . . . 121
7.3.6 Job: Migration (Expand) . . . 123
7.3.7 Job: Seeder (optional, idempotent) . . . 125
7.3.8 Job: Migration (Contract) (destruktiv, nach Stabilisierung) . . . 127
7.3.9 NetworkPolicy App (Zero-Trust, Beispiel) . . . 130
7.3.10 HorizontalPodAutoscaler . . . 131
7.3.11 ServiceMonitor . . . 132
7.3.12 Namespace (Pod Security Standards „restricted“) . . . 133
7.3.13 ClusterImagePolicy . . . 133
7.3.14 HelmRelease . . . 134
7.4 Laravel-spezifische Anpassungen . . . 139
7.4.1 Expand/Contract-Verfahren . . . 139
7.4.2 FaqSeeder . . . 140
7.4.3 Bootstrap-Prozess . . . 141
7.4.4 Dashboard . . . 142
7.5 Dockerfiles . . . 146
7.5.1 PHP-FPM (Laravel) . . . 146
7.5.2 Nginx . . . 148
7.6 Observability . . . 149
7.6.1 Metriken . . . 149
7.6.2 Prometheus-Regeln . . . 150

Abbildungsverzeichnis

2.1 Schematische Darstellung eines typischen Development and Operations (DevOps)-Flows mit Fokus auf kontinuierlicher Integration, Bereitstellung und Überwachung. Der Zyklus verdeutlicht die enge Verzahnung von Entwicklung und Betrieb sowie den iterativen Charakter moderner Softwarebereitstellung (nach [PP24]). 16
2.2 Darstellung eines Development - Security - Operations (DevSecOps)-Flows, bei dem Sicherheitsaspekte integraler Bestandteil jedes Prozessschritts sind. Durch das „Shift-Left"-Prinzip werden Sicherheitsfaktoren frühzeitig und kontinuierlich in Entwicklung, Test, Bereitstellung und Betrieb eingebunden (nach [PP24]). 17
2.3 Zusammenspiel zentraler Kubernetes-Komponenten: Ein Deployment erzeugt ein ReplicaSet, das mehrere Pods verwaltet. Diese Pods sind mit Labels gekennzeichnet, anhand derer ein Service die passenden Pods auswählt und unter einer stabilen IP bündelt. So wird eine Anwendung trotz dynamisch wechselnder Pods zuverlässig erreichbar (nach [The25e]). 19

3.1 Makro-Architektur in der Docker-Phase: je Umgebung ein eigener Server mit Docker und darauf laufenden Laravel-App-Containern. 21
3.2 Feature-Based Development: mehrere langlebige Branches, Integration über `develop` und `release`, separate Testumgebung für Feedback 23
3.3 Einzelner Docker-Container im Ist-Zustand: vereint Web-App, Supervisor/Queues und Cronjobs. Code-Änderungen wurden per Secure File Transfer Protocol (SFTP) vom Entwicklerrechner in die Container-Instanz übertragen. Dies führte zu einer unklaren Trennung der Verantwortlichkeiten im Container, manuellen und fehleranfälligen Deployments sowie eingeschränkter Skalierbarkeit. 24

4.1 Use-Case-Diagramm der Anwendung *„Kindersucharmband"*. Es zeigt die zentralen Akteure (Erziehungsberechtigte, DLRG und System) sowie deren Interaktionen, die die funktionale Grundlage für die spätere DevOps-Transformation bilden. 30

4.2 Trunk-Based Development: mehrere kurzlebige Branches, Integration über Pull Requests (Pull Request (PR)) nach `main`, separate Testumgebung für Feedback entfällt. Features werden intern in Produktiv getestet und ausgerollt. 32
4.3 Die Abbildung veranschaulicht den Ablauf der Git Hooks (Pre-Commit). Das entsprechende Shell-Skript ist dem Anhang zu entnehmen (siehe Listing 7.1.2). 34
4.4 Die Abbildung veranschaulicht den Ablauf der Fastlane (Feature-Branch). Zunächst werden Linting und Regressionstests (Unit- und Feature-Tests) ausgeführt, um syntaktische Fehler zu erkennen und die Kernlogik abzusichern. Anschließend werden Mutations-Tests zur Wirksamkeitskontrolle der Testsuite durchgeführt. Im weiteren Verlauf werden sogenannte Secret- und Vulnerability-Scans (Common Vulnerabilities and Exposures (CVE)) durchgeführt, um das Risiko von Datenlecks und bekannten Schwachstellen zu minimieren. Die entsprechende Definition der *Fastlane* ist dem Anhang zu entnehmen (siehe Anhang 7.2.1). 35
4.5 Die Abbildung veranschaulicht den Ablauf der PR Quality Gates (Pull Requests nach `main`). Zunächst werden Linting mit automatischen Fixes und Regressionstests (Unit- und Feature-Tests) mit einer projektspezifischen Untergrenze (beim „Kindersucharmband“ 80 %) ausgeführt. Anschließend werden Mutations-Tests mit einem Mutation Score Indicator (MSI)-Threshold von 80 % durchlaufen. Im weiteren Verlauf werden Datenbank-Migrations-Checks (Trockenlauf/Validierung), Composer und NPM-Audits, Build-Asset-Test (Frontend), Dockerfile-Linting von App und Nginx (siehe Abschnitt 4.3.1) sowie Secret- und Vulnerability-Scans (CVE) durchgeführt. Die entsprechende Definition des *PR Quality Gates* ist dem Anhang zu entnehmen (siehe Anhang 7.2.1). 36
4.6 Ablauf der Continuous Integration (CI)/Continuous Delivery (CD)-Pipeline: Von Pre-Commit Hooks und Commit über Fastlane in die Quality Gates, automatische Release-Erstellung, Build, Scan und Signatur bis zum Container-Registry (GitHub Container Registry (GHCR)) und GitOps-Deployment. 38
4.7 Überblick über Repositories, CI/CD-Pipeline und Artefakte (GHCR). . . 39
4.8 Monolithische Migration: destruktive Änderung erzeugt ein Inkompatibilitätsfenster während des Rollouts. **(1)** Destruktive Änderung während des Rollouts. **(2)** Alte App v1 erwartet Schema S0, die Datenbank ist bereits S1. **(3)** Kurzzeitige Ausfälle oder Fehlerantworten. 41
4.9 Expand/Contract-Migration: additive Änderung ermöglicht ein Kompatibilitätsfenster während des Rollouts. **(1)** Zunächst wird das Schema nur erweitert (additiv). **(2)** Sowohl App v1 als auch App v2 können mit Schema S1 arbeiten. **(3)** Nach erfolgreichem Rollout entfernt die destruktive Änderung (Contract) die Altlasten. 42

4.10 Zeitlicher Ablauf von Expand/Contract-Migration mit ergänzenden Seeding-Schritten. 43
4.11 Makro-Übersicht erlaubter Verbindungen zwischen zentralen Workloads gemäß NetworkPolicies. 45
4.12 Makro-Darstellung des Kubernetes-Clusters. 49
4.13 Multistage-Build-Pipelines für App (PHP-FPM) und Nginx: Build-Stages *(graue Rechtecke)* erzeugen Assets *(gestrichelte Rechtecke)* bzw. Dependencies. Die finalen Runtime-Images übernehmen nur die benötigten Artefakte *(grüne Rechtecke)*. 50
4.14 Visualisierte Darstellung eines Rolling-Update-Verfahrens. Die Pods *(bunte Kreise)* werden vom Deployment *„Kindersucharmband“ (graue Rechtecke)* schrittweise von *v1* auf *v2* aktualisiert [The25z]. 52
4.15 Dark Launch über Feature Flags 52
4.16 Anhand des beigefügten Screenshots wird das Dashboard veranschaulicht. Im vorliegenden Kontext wird das Feature *SecureSearch* präsentiert. Zusätzlich sind die Zustände für die öffentliche und interne Sichtbarkeit festgelegt. Domänexperten haben jederzeit die Möglichkeit, über das Dashboard das Feature reversibel zu aktivieren oder zu deaktivieren. 54
4.17 Traffic-Mirroring als zukünftige Erweiterung: parallele Verarbeitung ohne Nutzereinfluss 56
4.18 Darstellung der Interaktion um Telemetriedaten zentralisiert in Grafana zu visualisieren. Die Logs der einzelnen Pods im `Namespace` *„Kindersucharmband“* werden von *Promtail* gesammelt und in *Loki* aggregiert und gespeichert [Grad, Grab]. Metrik-Endpunkte werden von *Prometheus* aufgerufen und an *Alertmanager* sowie Grafana weitergeleitet. *Grafana* hat somit *Prometheus*, *Alertmanager* und *Loki* als Datenquelle [Graa, Grac]. 62
4.19 Ablauf der keyless Signierung und Zertifikatsausstellung mit Sigstore (nach [NMTA22]). 64

Tabellenverzeichnis

4.1 Vergleich des Deployments vor und nach der Umstellung 47
4.2 Änderungskarte – Design- und Prozessunterschiede 65
4.3 Vorher/Nachher-Überblick ausgewählter Kriterien 66

5.1 Secret-Scans: Gitleaks (Repo) und Trivy (Runtime-Images) 73
5.2 Lease-Rotation: Hash- und Credential-Änderung der `.env` 74
5.3 R9 Pfadtests: Erwartung vs. Beobachtung 76
5.4 Image-Größen (komprimiert) 81
5.5 Gesamtzahl CVEs (Trivy, alle Schweregrade) 82
5.6 Manuelle Schritte je Release (vorher vs. jetzt) 83
5.7 Idempotenz-Spot-Check `FAQs` (Erstlauf vs. Zweitlauf) 85
5.8 Rollback & Rollforward: Wiederanlaufzeit und Verfügbarkeit 88
5.9 Abdeckung der Forschungsfragen (F#) durch evaluierte Anforderungen (R#) 90

1 Einleitung

1.1 Hintergrund und Motivation

Die Deutsche Lebens-Rettungs-Gesellschaft (DLRG) betreibt zur Unterstützung ihrer Einsatzkräfte an Nord- und Ostsee die Webanwendung *„Kindersucharmband“*. Die Anwendung hilft dabei, Kinder, die sich am Strand verlaufen haben, schnell wieder mit ihren Eltern zusammenzuführen [DLR]. Durch saisonal stark schwankende Nutzung, hohe Verfügbarkeitsanforderungen und die Verarbeitung personenbezogener Daten ergeben sich besondere Anforderungen an Betrieb, Sicherheit und Skalierbarkeit der Anwendung.

Laravel als weit verbreitetes PHP-Framework bietet eine solide Grundlage für die Entwicklung solcher Anwendungen, stellt jedoch keine Werkzeuge für einen durchgängigen, gehärteten Betriebsprozess bereit. Manuelle Deployments, fehlende Automatisierung und unzureichende Sicherheitsmechanismen führen in der Praxis häufig zu Inkonsistenzen, Ausfällen und erhöhter Angriffsfläche. DevOps- und DevSecOps-Praktiken wie Continuous Integration, Continuous Deployment, GitOps, Container-Härtung und sicheres Secret Management adressieren diese Probleme und schaffen einen Rahmen für reproduzierbare, automatisierte und sichere Betriebsprozesse. Diese Arbeit untersucht, wie sich diese Prinzipien im On-Premise-Kubernetes-Betrieb der DLRG wirksam und messbar umsetzen lassen.

1.2 Problemstellung

Die bisherige Bereitstellung der Anwendung war durch manuelle Arbeitsschritte, unzureichende Automatisierung und eine enge Kopplung von Komponenten gekennzeichnet. Container-Images waren groß, enthielten unnötige Abhängigkeiten und damit zahlreiche bekannte Schwachstellen (CVE). Sensible Daten wurden während des Builds in Images integriert, was ein erhebliches Sicherheitsrisiko darstellte. Rollouts führten regelmäßig zu Ausfallzeiten, und es existierten weder Monitoring- noch Logging-Mechanismen zur frühzeitigen Fehlererkennung. Insgesamt fehlte ein systematischer End-to-End-Prozess, der Automatisierung, Sicherheit und Reproduzierbarkeit verbindet und auf Laravel-Anwendungen übertragbar ist.

1.3 Zielsetzung der Arbeit

Ziel der Arbeit ist die Konzeption und prototypische Umsetzung eines Dev(Sec)Ops-basierten Referenzprozesses für Laravel-Anwendungen auf Kubernetes im Kontext der DLRG. Der Prozess soll:

- Builds und Deployments standardisieren,
- die Lieferkettensicherheit durch Software Bill of Materials (SBOM) und Image-Signaturen gewährleisten,
- Secrets sicher zur Laufzeit bereitstellen,
- horizontale Skalierung und Zero-Downtime-Rollouts ermöglichen sowie
- durch moderne Observability-Werkzeuge Transparenz und Nachvollziehbarkeit im Betrieb erhöhen.

Die Wirksamkeit wird anhand quantitativer und qualitativer Kriterien gegenüber dem Ist-Zustand evaluiert.

1.4 Forschungsfragen

Auf Grundlage der beschriebenen Ausgangssituation ergeben sich folgende Forschungsfragen:

F1: Wie können DevSecOps-Prinzipien so auf Laravel-Anwendungen übertragen werden, dass Build, Test, Härtung, Signatur und Deployment reproduzierbar und auditierbar erfolgen?

F2: Welche Kombination aus Vault-basiertem Secret Management, NetworkPolicies und restriktiven `securityContext`-Vorgaben minimiert die Angriffsfläche im Kubernetes-Cluster, ohne die Betriebseffizienz zu beeinträchtigen?

F3: Welche messbaren Verbesserungen ergeben sich im Vergleich zum Ist-Zustand, etwa in Bezug auf Imagegröße, Anzahl und Schwere von CVEs, Ausfallzeiten oder manuelle Eingriffe im Deployment-Prozess?

F4: Wie lassen sich datenbankbezogene Änderungsprozesse (Migrationen und Seeder) in CI/GitOps integrieren, sodass sie ohne manuelle Schritte und ohne Downtime ablaufen und dabei auditierbar bleiben?

1.5 Vorgehensweise und Methodik

Die Arbeit folgt einem Design-Science-Ansatz, der praxisorientierte Systementwicklung mit wissenschaftlicher Evaluation verbindet. Dazu werden zunächst Ist-Zustand und Schwächen analysiert und daraus Anforderungen abgeleitet. Im Anschluss erfolgt die Konzeption und Implementierung einer Zielarchitektur, die zentrale DevOps- und DevSecOps-Praktiken integriert. Darunter Trunk-Based Development, automatisierte Quality Gates, signierte Container-Artefakte, GitOps-Deployments und Vault-gestütztes Secret Management. Abschließend werden die Ergebnisse anhand definierter Metriken evaluiert und kritisch diskutiert.

1.6 Aufbau der Arbeit

Kapitel 2 erläutert die theoretischen und technischen Grundlagen zu Laravel, DevOps, DevSecOps, CI/CD, GitOps und Kubernetes. Kapitel 3 analysiert die Ausgangssituation der DLRG und leitet daraus konkrete Anforderungen ab. Kapitel 4 beschreibt den Entwurf und die Umsetzung der Zielarchitektur. Kapitel 5 bewertet die Ergebnisse anhand der Forschungsfragen. Kapitel 6 fasst die Arbeit zusammen und gibt einen Ausblick auf zukünftige Entwicklungen.

1.7 Ergebnisse in Kürze – Antworten auf die Forschungsfragen

F1 – DevSecOps für Laravel reproduzierbar & auditierbar: Die Pipeline liefert funktional deterministische Runtime-Inhalte, durchgängig signierte Artefakte mit verifizierbarer Provenance und eine lückenlose Auditkette vom Commit bis zum Pod. Deployments erfolgen deklarativ via GitOps mit Zero-Downtime-Rollouts (*vgl.* R1 (5.2.1), R3 (5.2.3), R4 (5.2.4), R5 (5.2.5).

F2 – Minimierte Angriffsfläche bei effizientem Betrieb: Secrets sind weder im Repository noch in Images. Rotation erfolgt zur Laufzeit. „Default deny"-NetworkPolicies, Pod Security Admission (PSA) `restricted`, restriktiver `securityContext` und Least-Privilege-RBAC reduzieren die Angriffsfläche ohne Funktionsverlust (*vgl.* R6 (5.2.6)–R10 (5.2.10)).

F3 – Messbare Verbesserungen gegenüber dem Ist-Zustand: Runtime-Images schrumpfen kombiniert um 83,2 % (App 88,4 %, Nginx 94,8 %), die Gesamtzahl an CVEs sinkt um 99,8 % (2441 → 5), die p95-Downtime liegt bei 0 s, manuelle Release-Schritte

reduzieren sich von 6 auf 1, Time-to-Recover beträgt 34 s/28 s (Rollforward/Rollback) (*vgl.* R11 (5.2.11), R12 (5.2.12), R5 (5.2.5), R13 (5.2.13), R17 (5.2.17)).

F4 – DB-Änderungen CI/GitOps-integriert, ohne Downtime, auditierbar: Laravel-Migrationen und -Seeding laufen automatisiert, idempotent und orchestriert im Release-Flow über das konstruierte Helm-Chart. Änderungen sind revisionssicher eingebettet (*vgl.* R14 (5.2.14), R3 (5.2.3)).

Einschränkungen: Byte-genaue Reproduzierbarkeit des Image-Digests wurde nicht stabil erreicht. Für Branch-Lebensdauer und Pre-Commit-Pass-Rate liegen (noch) keine belastbaren Kennzahlen vor.

Nachweise: Ausführliche Messungen und Nachweise in Kapitel 5
(R1 (5.2.1) – R19 (5.2.19)).

2 Grundlagen

2.1 Laravel im Kontext von DevOps

Laravel ist ein PHP-Framework für Webanwendungen. Die Syntax, die Handhabung und eine aktive Community können eine schnelle Umsetzung fördern [Docc]. In vielen Projekten wird Laravel als monolithische Anwendung eingesetzt, in der Entwicklungsaspekte dominieren, während Betrieb, Deployment und Skalierbarkeit häufig nachgeordnet sind. Im Zusammenspiel mit DevOps entstehen daraus spezifische Fragestellungen, die über den klassischen Frameworkeinsatz hinausgehen.

Einordnung und Entwicklungsmodell Laravel adressiert insbesondere serverseitige Webentwicklung mit MVC-Struktur, Routing, Object-Relational Mapping (ORM) (Eloquent), Template-Engine (Blade) sowie einer CLI-Unterstützung [Docf, Docb, Doca, Lar25a]. In der Praxis erfolgt die lokale Entwicklung oft mit PHP, Composer, Node.js (npm) und einer relationalen Datenbank.

Betriebsrelevante Eigenschaften Mehrere Frameworkfunktionen wirken sich direkt auf den Betrieb aus und verdienen besondere Beachtung:

- **Scheduler:** Wiederkehrende Aufgaben werden deklarativ im Code (`app/Console/Kernel.php`) definiert und typischerweise durch einen Cron-Eintrag im Minutentakt ausgelöst [Lar25g]. Damit verbunden sind Fragen nach Laufzeitverhalten, Idempotenz und Fehlerbehandlung.
- **Queueing:** Für asynchrone Verarbeitung bietet Laravel eine Abstraktion mit Treibern wie *Redis*, *Beanstalkd*, *SQS* oder *Database* [Lar25f]. Jobs werden über *Queue-Worker* (`php artisan queue:work`) ausgeführt. Relevant sind unter anderem Start-/Stop-Verhalten, Wiederholungen (*retries*) und Sichtbarkeit fehlgeschlagener Jobs.
- **Datenbankmigrationen und Seeder:** Schemaänderungen werden mit *Migrations* versioniert (`database/migrations`) und per `php artisan migrate` angewendet. *Seeders* initialisieren Referenzdaten (`php artisan db:seed`) [Lar25c, Lar25d].

Daraus ergeben sich zeitliche Abhängigkeiten zum Deployment und Anforderungen an Kompatibilität von Änderungen.

- **Konfiguration und Secrets:** Konfigurationswerte stammen aus `config/*` und Umgebungsvariablen, lokal oft über `.env` bereitgestellt [Docc]. Die `.env`-Datei wird nicht versioniert.

Typische Herausforderungen im DevOps-Kontext Aus diesen Eigenschaften leiten sich wiederkehrende Herausforderungen für Betrieb und Skalierung ab:

- **Prozessmodellierung:** Webprozess, Scheduler und Queue-Worker folgen unterschiedlichen Lebenszyklen.
- **Zustand und Speicher:** Sitzungen, Cache und Dateispeicher können lokal oder extern verwaltet werden. Für horizontale Skalierung gewinnt die Auslagerung zustandsbehafteter Komponenten (z. B. Sessions in Redis, Dateien in Objektspeichern) an Bedeutung.
- **Datenbankänderungen:** Migrations binden Anwendungs- und Datenbankschema zusammen. Betriebsseitig stellen sich Fragen der Kompatibilität über Versionen hinweg, der Laufzeit von Migrationen und der Wiederholbarkeit.
- **Fehlertransparenz und Beobachtbarkeit:** Scheduler- und Queue-Aktivitäten finden außerhalb des synchronen Request-Zyklus statt. Sichtbarkeit von Fehlersituationen, Metriken zu Latenz, Durchsatz und Fehlerraten sowie Nachverfolgbarkeit von *failed jobs* sind grundlegend für den stabilen Betrieb.
- **Konfiguration und Geheimnisse:** Die Trennung von Artefakt und Umgebungskonfiguration ist wesentlich. Sensible Werte werden nicht im Code verwaltet. Die Bereitstellung zur Laufzeit beeinflusst Build- und Startphasen.
- **Deployment-Timing:** Das Zusammenspiel von Deployments mit langlaufenden Jobs, Cron-gestützten Tasks und Datenbankmigrationen erzeugt zeitliche Kopplungen, die bei der Veröffentlichung neuer Versionen zu beachten sind.

2.2 DevOps und DevSecOps – Prinzipien und Praktiken

DevOps ist ein Ansatz, der die Zusammenarbeit von Entwicklung – Developer (Dev) und Betrieb – Operations (Ops) verbessern soll [KHDW16]. Ziel ist es, schnellere Releasezyklen, höhere Softwarequalität und eine engere Kollaboration zu erreichen [KHDW16, HF10]. Zentrale Prinzipien sind die Automatisierung von Build, Test und Deployment, die konsequente Anwendung von Continuous Integration (CI) und Continuous Delivery,

der Einsatz von Infrastructure as Code (IaC) sowie ein kontinuierliches Monitoring mit Feedbackschleifen [HF10, Kri22].

Im Alltag zeigt sich DevOps in Praktiken wie Versionskontrolle mit Git, automatisierten Tests, Containerisierung oder Orchestrierung mit Kubernetes. Gleichzeitig bringt der Ansatz Herausforderungen mit sich, etwa in Form von Kulturwandel, Tool-Integration oder der sicheren Verwaltung von Konfigurationen und Secrets [FHK18, Kri22].

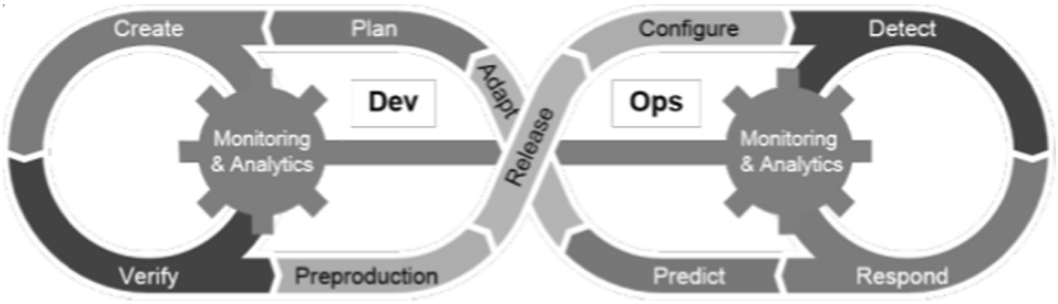

Abbildung 2.1: Schematische Darstellung eines typischen DevOps-Flows mit Fokus auf kontinuierlicher Integration, Bereitstellung und Überwachung. Der Zyklus verdeutlicht die enge Verzahnung von Entwicklung und Betrieb sowie den iterativen Charakter moderner Softwarebereitstellung (nach [PP24]).

DevSecOps erweitert das DevOps-Paradigma um den Aspekt der „Security as Code“. Sicherheitsprüfungen sollen nicht nachgelagert erfolgen, sondern möglichst früh im Entwicklungsprozess integriert werden (Shift Left Security) [Seh23]. Dazu zählen automatisierte Scans in CI/CD-Pipelines, beispielsweise Code-Analyse, Dependency-Scanning oder Container-Scanning. In aktuellen Studien wird zudem gezeigt, dass CI/CD-Pipelines selbst ein attraktives Angriffsziel darstellen und vielfältige Bedrohungen bergen [PSW+24]. Weitere Prinzipien sind Geheimnisverwaltung und Richtlinienkontrolle durch Policy as Code. Besonders im Kontext von Laravel-Anwendungen ist DevSecOps relevant, da hier die sichere Pipeline-Konfiguration, Container-Härtung und der Schutz sensibler Daten essentiell sind [Sta19].

2.3 CI/CD und GitOps als Basis moderner Deployments

Unter Continuous Integration (CI) versteht man das regelmäßige Einspielen von Codeänderungen in ein zentrales Repository. Auf diese Weise lassen sich Integrationsproble-

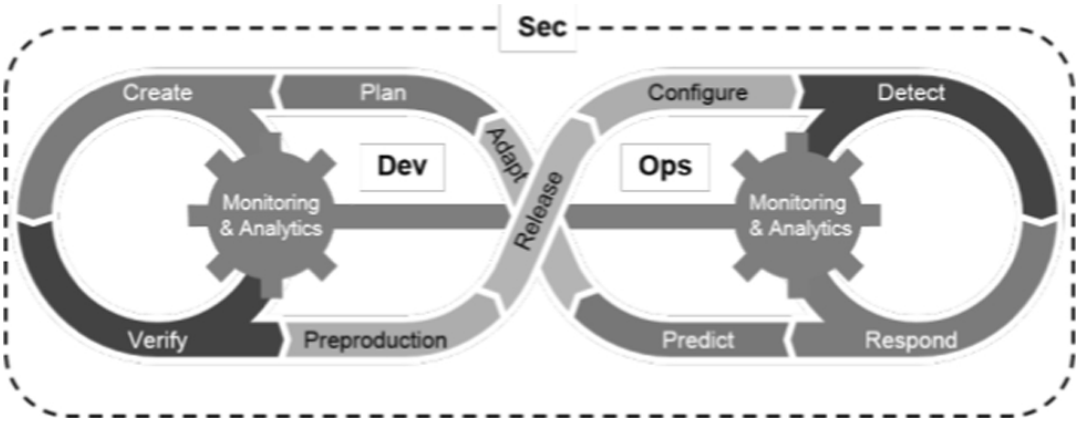

Abbildung 2.2: Darstellung eines DevSecOps-Flows, bei dem Sicherheitsaspekte integraler Bestandteil jedes Prozessschritts sind. Durch das „Shift-Left“-Prinzip werden Sicherheitsfaktoren frühzeitig und kontinuierlich in Entwicklung, Test, Bereitstellung und Betrieb eingebunden (nach [PP24]).

me frühzeitig erkennen, bevor sie sich im Projekt verfestigen. Typisch ist dabei die Einbindung automatisierter Tests, Code-Checks wie Linting sowie das Erstellen lauffähiger Builds [SBZ17].

Continuous Delivery bzw. Continuous Deployment (CD) baut auf diesen Mechanismen auf. Ziel ist es, neue Versionen nicht nur zu integrieren, sondern auch möglichst schnell und zuverlässig in Test- oder sogar Produktionsumgebungen bereitzustellen. Praktisch äußert sich das etwa in automatisierten Deployments, die im Fehlerfall auch wieder zurückgerollt werden können [SMS24].

Gerade bei Laravel-Projekten ist dieser Ansatz von Bedeutung. Jede Codeänderung muss testbar und im Idealfall sofort deployfähig sein [TP20]. Neben den klassischen Tests gehören dazu auch Dinge wie das automatische Bauen von Assets oder das Durchführen von Security-Scans, um Schwachstellen in Abhängigkeiten oder Containern frühzeitig aufzudecken.

GitOps als Erweiterung von CI/CD

Während CI/CD in erster Linie den Entwicklungs- und Releaseprozess einer Anwendung abbildet, greift GitOps weiter [Red25]. Hier wird Git nicht nur als Versionsverwaltung für den Code genutzt, sondern auch als Steuerungszentrale für Infrastruktur und Betrieb. Änderungen an Deployments oder Systemkonfigurationen werden deklarativ in Git abgelegt und anschließend automatisch mit der laufenden Umgebung synchronisiert [BH22].

Die Idee dahinter ist, dass das Repository den gewünschten Zielzustand beschreibt, während Operatoren im Cluster diesen Zustand kontinuierlich mit der Realität abgleichen. Besonders im Zusammenspiel mit Kubernetes hat sich dieses Vorgehen etabliert [Red25].

Der Nutzen von GitOps zeigt sich in mehreren Punkten: Alle Änderungen sind über Git nachvollziehbar, ein Rollback ist durch einen simplen Revert möglich. Sowohl Entwicklung als auch Betrieb arbeiten mit denselben Prozessen, was die Zusammenarbeit erleichtert. Außerdem ist der Systemzustand klar dokumentiert und jederzeit reproduzierbar. In Szenarien wie einem Ausfall oder bei der Wiederherstellung nach einem Fehler kann ein Cluster komplett aus dem Repository neu aufgebaut werden. Hinzu kommt der Sicherheitsaspekt: Da Änderungen ausschließlich über Git-Workflows wie Pull Requests und Code Reviews erfolgen, entsteht automatisch ein revisionssicheres Audit-Log.

2.4 Containerisierung und Orchestrierung mit Kubernetes

Die Containerisierung hat sich in den letzten Jahren in der Softwarebereitstellung weit verbreitet [SH25, Dat23]. Sie ermöglicht die Erstellung einheitlicher, isolierter und portabler Laufzeitumgebungen, die unabhängig von der zugrunde liegenden Infrastruktur betrieben werden können. Dadurch lassen sich Anwendungen reproduzierbar entwickeln, testen und in verschiedenen Umgebungen ausrollen.

Für die Verwaltung und Koordination einer großen Zahl von Containern hat sich Kubernetes als De-facto-Standard etabliert [Car22]. Die Plattform bietet Mechanismen zur Orchestrierung, die über das Starten einzelner Container hinausgehen [The25n]. Im Kern basiert Kubernetes auf einer Reihe von Konzepten, die das Zusammenspiel von Anwendungen und Infrastruktur regeln [The25f].

In Kubernetes bildet der *Pod* die kleinste Einheit, die eine Anwendung oder Teile davon ausführt [The25q]. Mehrere *Pods* können über ein *ReplicaSet* organisiert werden, wodurch eine bestimmte Anzahl parallel laufender Instanzen sichergestellt wird [The25r]. Änderungen am System, etwa ein Update oder ein Rollback, werden über sogenannte *Deployments* gesteuert, die die *ReplicaSets* verwalten [The25j, The25r]. Damit Anwendungen trotz wechselnder Instanzen erreichbar bleiben, sorgt ein *Service* für eine stabile Zugriffsschicht und übernimmt gleichzeitig die Verteilung von Anfragen [The25u]. Darüber hinaus dienen *ConfigMaps* und *Secrets* der Verwaltung von Konfigurationen und vertraulichen Informationen, während *Ingress*-Ressourcen den externen Zugriff auf Dienste steuern [The25u].

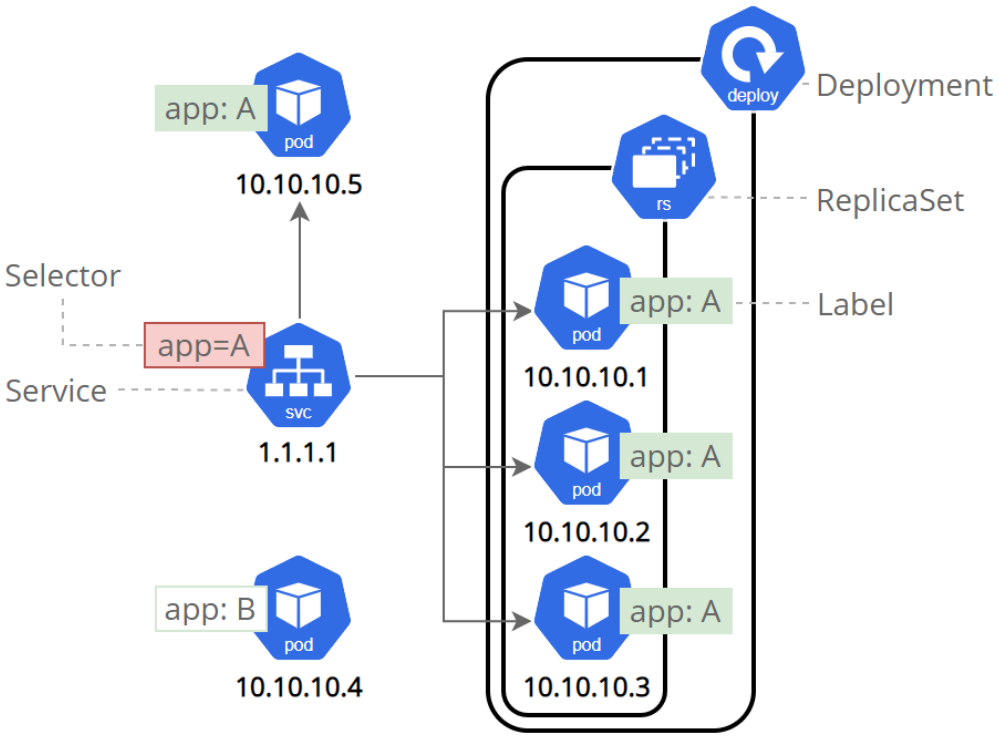

Abbildung 2.3: Zusammenspiel zentraler Kubernetes-Komponenten: Ein Deployment erzeugt ein ReplicaSet, das mehrere Pods verwaltet. Diese Pods sind mit Labels gekennzeichnet, anhand derer ein Service die passenden Pods auswählt und unter einer stabilen IP bündelt. So wird eine Anwendung trotz dynamisch wechselnder Pods zuverlässig erreichbar (nach [The25e]).

Kubernetes stellt Mechanismen für horizontale Skalierung, Ausfallsicherheit, Self-Healing sowie für automatisierte Rollouts und Rollbacks bereit [The25d, The25z]. Der Einsatz ist jedoch mit einer erhöhten Komplexität verbunden, da zahlreiche Konfigurationsoptionen zu berücksichtigen sind und sicherheitsrelevante Aspekte wie Zugriffskontrolle und Geheimnisverwaltung besondere Aufmerksamkeit erfordern. Im vorliegenden Projektkontext wird Kubernetes als Zielumgebung für produktive Deployments der Laravel-Anwendungen der DLRG genutzt und bildet damit die technische Basis für Automatisierung und Sicherheit.

Helm als Paketmanager für Kubernetes

Kubernetes bringt viele Charakteristika mit sich, etwa Skalierbarkeit, Ausfallsicherheit und die Möglichkeit automatisierter Deployments [The25n]. Gleichzeitig steigt mit der

Anzahl der eingesetzten Ressourcen jedoch auch die Komplexität, da selbst einfache Anwendungen aus zahlreichen Konfigurationsdateien (z.B. `Deployment`, `Service` oder `Ingress`) bestehen können. Um diese Komplexität zu verringern, hat sich Helm als Paketmanager für Kubernetes etabliert [The25a].

Helm ermöglicht es, mehrere Ressourcen in sogenannten *Charts* zu bündeln und diese als wiederverwendbare Pakete bereitzustellen. Ein Chart enthält alle notwendigen Definitionen, um eine Anwendung vollständig in Kubernetes zu betreiben, und ermöglicht damit sowohl die Bereitstellung als auch die Standardisierung von Deployments [The25a].

Eine wesentliche Eigenschaft von Helm ist die Möglichkeit, Konfigurationen über *Values*-Dateien flexibel zu parametrisieren, ohne die zugrunde liegenden Templates ändern zu müssen [The25a]. Darüber hinaus unterstützt Helm die Versionsverwaltung und ermöglicht so sowohl Updates als auch Rollbacks von Anwendungen. Durch die Integration in CI/CD- und GitOps-Workflows lassen sich Deployments automatisiert, reproduzierbar und revisionssicher durchführen.

Kritische Betrachtung: In früheren Versionen (bis Helm v2) war Helm aufgrund der Komponente *Tiller* umstritten. Tiller lief als zusätzlicher Serverprozess im Cluster und benötigte weitreichende Berechtigungen, was sicherheitskritische Angriffsflächen eröffnete und die Komplexität erhöhte. Mit Helm v3 wurde Tiller entfernt, sodass Helm seither direkt clientseitig mit dem Kubernetes-API-Server kommuniziert und die Berechtigungen des jeweiligen Nutzers nutzt [The25c]. Dadurch haben sich die Sicherheitsbedenken weitgehend erledigt und Helm konnte sich als De-facto-Standard für Kubernetes-Paketierung etablieren [SH25].

Damit übernimmt Helm die Rolle eines Bindeglieds zwischen Anwendungsentwicklung und Kubernetes-Orchestrierung. Es trägt wesentlich zur Vereinfachung, Wiederverwendbarkeit und Automatisierung von Deployments bei und ist deshalb ein zentraler Bestandteil moderner Kubernetes-Umgebungen.

3 Anforderungsanalyse

3.1 Ausgangslage bei der DLRG

Die Ausgangssituation bei der DLRG war über einen längeren Zeitraum durch eine heterogene und überwiegend manuell aufgebaute Systemlandschaft geprägt. Laravel-Anwendungen wurden auf dedizierten Servern betrieben, die jeweils separat für Entwicklungs-, Test- und Produktionsumgebungen konfiguriert waren. Zentrale Komponenten wie Webserver oder Datenbanken wurden in diesem Kontext manuell installiert und verwaltet. Eine konsistente Dokumentation oder einheitliche Automatisierung fehlten weitgehend, sodass es regelmäßig zu Abweichungen zwischen den Umgebungen („Configuration Drift“) kam. Diese Unterschiede beeinträchtigten die Betriebssicherheit und führten zu einem erhöhten Wartungsaufwand.

Mit der Einführung von Docker konnten wesentliche Teile der Bereitstellungsprozesse erstmals standardisiert und reproduzierbar gestaltet werden [Doc25c]. Anwendungen ließen sich dadurch schneller ausrollen und einfacher in getrennten Umgebungen betreiben. Parallel wurde eine erste CI/CD-Pipeline aufgebaut, die grundlegende Automatisierung ermöglichte. Allerdings fehlten dort wichtige Prüfmechanismen wie Fail-fast-Checks oder Quality Gates, sodass fehlerhafte Builds nicht zuverlässig abgefangen wurden. In der Praxis wurde für jede Umgebung (Entwicklung, Test und Produktion) ein eigener Server mit Docker betrieben.

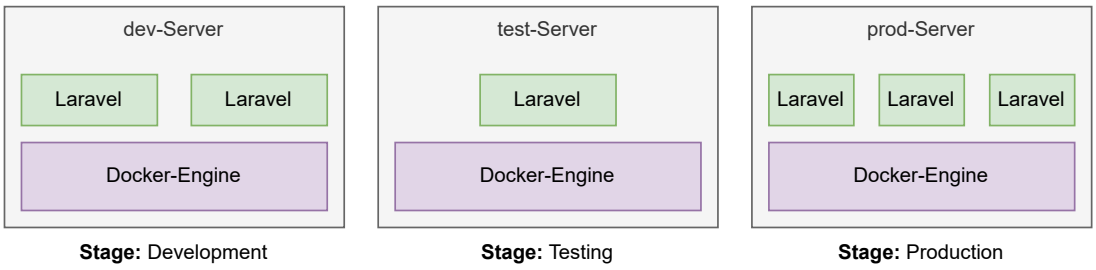

Abbildung 3.1: Makro-Architektur in der Docker-Phase: je Umgebung ein eigener Server mit Docker und darauf laufenden Laravel-App-Containern.

Diese Architektur erlaubte zwar eine klare Trennung, bot aber keine Funktionen wie automatische Lastverteilung, Selbstheilung oder planbare Rollouts. Hinzu kam, dass die Docker-Images sämtliche Build-Abhängigkeiten (Composer, Node (npm)) enthielten und dadurch eine Größe von über einem Gigabyte erreichten. Die große Angriffsfläche durch bekannte Sicherheitslücken (CVE) sowie das unsichere Handling von Konfigurationsdateien, die bereits beim Build in die Images integriert wurden, verschärften die Problematik [MIT25]. Zusätzlich war die Entwicklungsumgebung zentral organisiert: Auf einem Server liefen mehrere Projekte parallel, für jeden Entwickler jeweils eine eigene Instanz. Wer nicht aktiv mitarbeitete, griff dabei häufig auf veraltete Remote-Stände zurück. Zudem erforderte dieses Modell eine ständige Internetverbindung, wodurch sowohl laufende Kosten entstanden als auch ein permanenter Absicherungsaufwand für den Entwicklungsserver notwendig war.

Die Entwicklung von Anwendungen erfolgte zunächst lokal auf den Rechnern der Entwickler. Änderungen am Quellcode wurden anschließend per SFTP in das Projektverzeichnis auf dem zentralen Entwicklungsserver übertragen (siehe Abbildung 3.3). Über eine zugewiesene URL ließ sich der neue Stand unmittelbar testen.

Um paralleles Arbeiten zu ermöglichen, stellte der Entwicklungsserver für jeden Entwickler eine eigene Remote-Instanz bereit, die in einem Docker-Container betrieben wurde. Da diese Instanzen nicht immer gleichzeitig aktualisiert wurden, standen sie teilweise auf unterschiedlichen Versionsständen, was die Zusammenarbeit erschwerte.

Die Branch-Strategie war featurebasiert [Fow20]. Neue Funktionen wurden in separaten Branches von `develop` entwickelt. Nach Fertigstellung eines Features erfolgte ein Pull Request zurück nach `develop`, wodurch die CI/CD-Pipeline ausgelöst wurde. Diese beschränkte sich jedoch auf grundlegende Schritte wie Container-Build und Deployment. Automatisierte Tests, Code-Analysen oder Security-Scans waren nicht integriert. Nach dem Merge in `develop` wurde ein weiterer Pull Request nach `release` erstellt, sodass die neuen Funktionen auf einer Testinstanz geprüft werden konnten. Dort gaben Stakeholder ihr Feedback, bevor je nach Bedarf ein Merge nach `main` durchgeführt und die Änderungen produktiv geschaltet wurden. Eine feste Release-Planung existierte nicht, der Zeitpunkt hing vom konkreten Anwendungsfall ab.

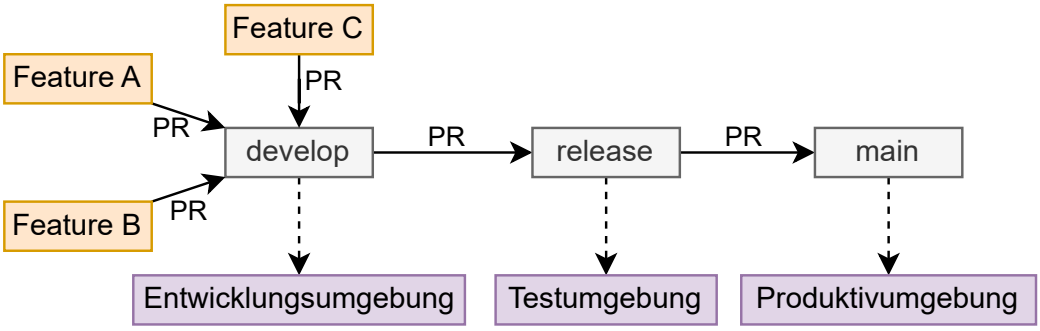

Abbildung 3.2: Feature-Based Development: mehrere langlebige Branches, Integration über `develop` und `release`, separate Testumgebung für Feedback

Das Deployment war nur teilweise automatisiert. Zwar stellte die Pipeline beim Merge nach `main` einen neuen Container bereit, weitere Schritte mussten jedoch manuell erfolgen. Besonders aufwendig waren Datenbankmigrationen: Nach jedem Deployment war ein manuelles Einloggen notwendig, woraufhin das Kommando php artisan migrate ausgeführt werden musste. Zusätzlich mussten auch Seeder manuell gestartet werden, um die Datenbank mit Basis- oder Konfigurationsdaten zu befüllen. Die konsequente Durchführung der beiden Schritte war nicht durchgängig gewährleistet, sodass Umgebungen teilweise auf unterschiedlichen Datenständen basierten. Damit war ein konsistentes Testen erschwert, und auch im Produktivbetrieb bestand die Gefahr, dass neue Funktionen ohne die dafür erforderlichen Daten ausgeliefert wurden. Hinzu kam, dass die Container eine relativ lange Startzeit von rund einer Minute hatten. Während dieser Phase war die Anwendung nicht erreichbar, was den Betrieb deutlich beeinträchtigte.

Auch im Hintergrundbetrieb zeigte sich eine enge Kopplung. Asynchrone Jobs wurden über das Laravel-Queue-System mit dem Datenbanktreiber abgewickelt [Lar25f]. Diese Jobs waren für zentrale Funktionen, wie zum Beispiel, der Versand von Benachrichtigungen (bspw. Mail oder SMS) der Anwendung zuständig.

Zusätzlich griffen die Anwendungen für Session-Handling und Caching ebenfalls auf die Datenbank zurück [Lar25b]. Damit übernahm die Datenbank gleich mehrere Rollen und wurde entsprechend stark belastet.

Für die Abarbeitung der Queue-Worker (siehe Abschnitt 2.1) wurde innerhalb des Containers ein Prozessmanager (`supervisord`) eingebunden [Lar25f]. Darüber hinaus war ein Crontab eingerichtet, der jede Minute `php artisan schedule:run` ausführte, um geplante Aufgaben wie die Löschung abgelaufener Einträge und die Benachrichtigung der Erziehungsberechtigten auszuführen [Lar25g]. Supervisor und Cronjobs im selben

Container entsprachen zwar den Empfehlungen für klassische Serverumgebungen, widersprachen jedoch den Best Practices im Containerumfeld [Doc25e, Doc25a]. Die offizielle Docker-Dokumentation empfiehlt hierzu:

> *„It's best practice to separate areas of concern by using one service per container."* [Doc25e]

Container sollten möglichst nur einen klar abgegrenzten Prozess enthalten [Doc25e].

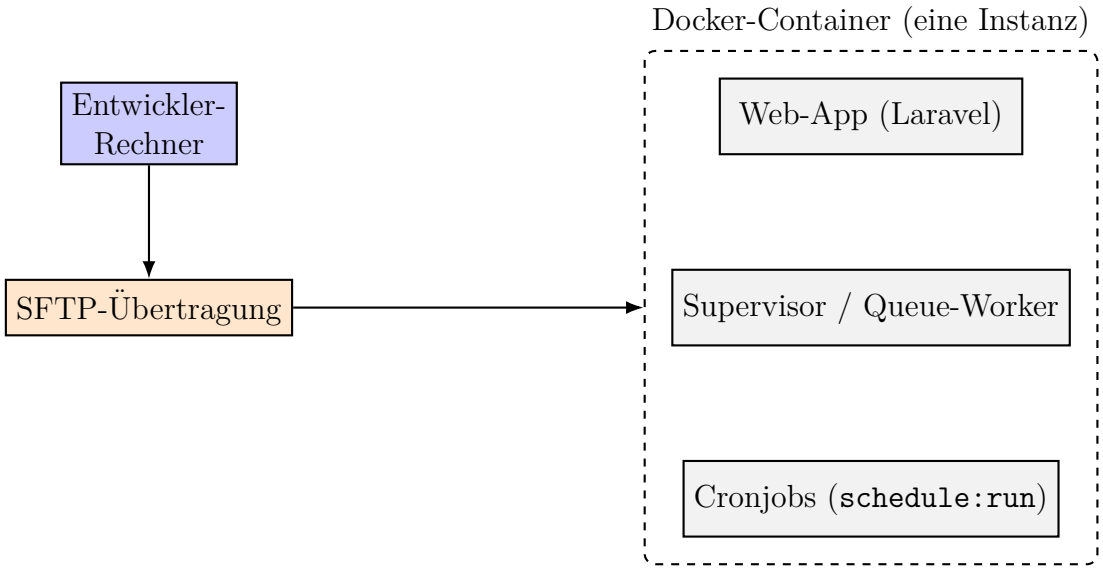

Abbildung 3.3: Einzelner Docker-Container im Ist-Zustand: vereint Web-App, Supervisor/Queues und Cronjobs. Code-Änderungen wurden per SFTP vom Entwicklerrechner in die Container-Instanz übertragen. Dies führte zu einer unklaren Trennung der Verantwortlichkeiten im Container, manuellen und fehleranfälligen Deployments sowie eingeschränkter Skalierbarkeit.

Ein systematisches Logging oder Monitoring war nicht eingerichtet. Logs wurden lediglich im Container gespeichert. Eine zentrale Auswertung oder Überwachung der Anwendung war damit nicht möglich. Ebenso wenig gab es Mechanismen für Rollbacks: Wenn ein Deployment fehlerhaft war, mussten Probleme manuell behoben werden, was zwangsläufig zu Ausfallzeiten führte.

Hinzu kamen sicherheitsrelevante Schwächen. Entwicklungscontainer wurden unverändert auch in der Produktion eingesetzt. Die Container konnten ohne Einschränkungen untereinander kommunizieren (Ost-West), da keine Netzwerktrennung vorgesehen war. Zudem

waren die Images nicht gehärtet und enthielten zahlreiche Abhängigkeiten, die im Betrieb nicht benötigt wurden. Dadurch stieg sowohl die Größe der Images als auch die Angriffsfläche für potenzielle Sicherheitslücken.

Insgesamt funktionierte der Prozess zwar, war jedoch durch manuelle Eingriffe, eingeschränkte Automatisierung und die Mehrfachbelastung der Datenbank fehleranfällig und nur bedingt skalierbar.

3.2 Identifizierte Schwächen

Auf Basis der Ausgangslage wurden die folgenden, systematischen Schwächen identifiziert:

S1: Manuelle Datenbankänderungen: Migrationen/Seeder wurden nach Deployments manuell ausgeführt und führten zu inkonsistenten Datenstände zwischen Umgebungen.

S2: Unvollständige CI/CD-Prüfungen: Pipeline ohne Tests, Analysen, Sicherheits- und Lieferkettenprüfungen. Fehlerhafte Artefakte gelangten weiter.

S3: Gekoppelte Nebenprozesse: Queues, Cron/Scheduler und Web-App liefen im selben Container. Datenbank diente zusätzlich als Queue/Session/Cache-Backend.

S4: Begrenzte Skalierbarkeit und langsamer Start: Enge Kopplung an die Datenbank. Container-Startzeiten von ca. 1 min erschwerten horizontale Skalierung.

S5: Verfügbarkeit und Release-Governance: Neustarts führten zu Downtime. Releases erfolgten ohne klare Steuerung.

S6: Sicherheitsdefizite in Images und Netzwerk: Große, ungehärtete Images. Fehlende Netzisolation (Ost-West). Entwicklungs-Images in Produktion.

S7: Fehlende Observability: Logs nur lokal im Container, kein zentrales Logging/Monitoring/Alerting.

S8: Organisatorischer Single Point of Failure (SPoF) in der Entwicklung: Zentraler Dev-Server. Arbeit ohne Internet nicht möglich. Divergierende Remote-Stände.

S9: Keine Rollback-Strategie: Rücksprung auf stabile Versionen nicht definiert. Manuelle Eingriffe nötig.

S10: Configuration Drift: Manuell gepflegte Server führten zu abweichenden Zuständen zwischen Umgebungen.

S11: Ineffiziente Branch-Strategie: Langlebige Feature-Branches mit später Integration (`develop`/`release`/`main`). Erhöhte Merge-Risiken.

S12: Gekoppelte fachliche Releases an Deployments: Neue Funktionen konnten nur durch ein Produktiv-Deployment aktiviert oder deaktiviert werden.

3.3 Abgeleitete Anforderungen

Aus den identifizierten Schwächen (S) lassen sich folgende Anforderungen ableiten:

R1: Reproduzierbarer Build: Alle Container-Images werden via Multi-Stage-Dockerfiles deterministisch gebaut. Gleicher Commit ⇒ identischer Digest.
Kriterium: CI-Build aus gleichem Commit erzeugt stabilen SHA256-Digest.

R2: Signatur & Provenance: Container-Images sind kryptografisch signiert, lieferketten-konforme Provenance/Attestations werden erzeugt und verifizierbar bereitgestellt.
Kriterium: Verifikation der Signatur und der Attestations ist automatisiert prüfbar.

R3: Auditierbarkeit: Jede Build-/Deploy-Aktion erzeugt unveränderbare Audit-Events (Who/What/When/Commit/Digest) in einem zentralen System.
Kriterium: Durchsuchbare, korrelierbare Logs/Events mit Referenz auf Commit/Release vorhanden.

R4: GitOps-Deployment: Deployments erfolgen ausschließlich deklarativ über versionskontrollierte Manifeste und einem GitOps-Controller.
Kriterium: Änderungen laufen nur über Pull Request (PR)/Merge. Drift wird automatisch erkannt und korrigiert.

R5: Zero-Downtime-Rollouts: Standard-Strategie ist Rolling Update mit Health-/Readiness-Gates.
Kriterium: 95%-Quantil der Downtime pro Release = 0 s in Tests.

R6: Integriertes Secret Management: Keine Secrets in Repository/Images/-ConfigMaps. Abruf zur Laufzeit aus einem zentralen, revisionssicheren Secret-Management-System (statisch/dynamisch).
Kriterium: Secret-Scans finden 0 Treffer im Repo/Runtime-Image.

R7: Secret-Rotation: Zeit- oder ereignisbasierte Rotation kritischer Secrets (z. B. Datenbankzugänge).
Kriterium: Dokumentierter Rotationsplan und erfolgreich validierter Probelauf.

R8: NetworkPolicies: Default Deny: Namespace-weit eingehend/ausgehend *deny*, Freigaben strikt nach Least-Privilege.
Kriterium: Netzwerktests blocken unerlaubte Pfade.

R9: Restriktiver `securityContext`: `runAsNonRoot`, `readOnlyRootFilesystem`, systemnahe Härtung, Capability-Reduktion.
Kriterium: Pod-Sicherheitsstandard *Restricted* erfüllt.

R10: Least-Privilege-RBAC: Spezifische ServiceAccounts mit minimalen Rechten. Kein Default-SA.
Kriterium: Berechtigungsprüfung zeigt keine unnötigen Verben/Ressourcen.

R11: Kleinere Images: Reduktion der Runtime-Imagegröße um $\geq 80\,\%$ gegenüber Ist-Zustand.
Kriterium: Vergleich in MB pro Image (vor/nach).

R12: Weniger CVE: $\geq 90\,\%$ weniger Schwachstellen in Runtime-Images.
Kriterium: Vor-/Nach-Report eines Container-Vulnerability-Scanners.

R13: Weniger manuelle Schritte: Manuelle Eingriffe pro Release von Ist 6 auf $\leq$ 1 (nur Pull Request (PR)).
Kriterium: Release-Runbook mit Schrittzähler.

R14: Automatisierte Datenbank-Änderungen: Datenbank-Migrationen plus Seeder laufen orchestriert und reproduzierbar.
Kriterium: Grüne Pipeline.

R15: Beobachtbarkeit: Zentrale, strukturierte Logs/Metriken/Traces. Definierte Service Level Objective (SLO).
Kriterium: Alerts vorhanden und regelmäßige SLO-Reports.

R16: Feature-Flag-gestützte Releases: Deployments sind vom fachlichen Release entkoppelt. Neue Funktionen werden über Feature Flags/dynamische Konfiguration aktiviert.
Kriterium: Flags zentral steuerbar (Enable/Disable, Scoping), Status versioniert/auditierbar. Deaktivierung ohne Redeploy möglich.

R17: Rollback-Fähigkeit: Deployments sind jederzeit reversibel. Version-Pinning per GitOps und reproduzierbare Artefakte erlauben Rücksprung auf eine vorherige, bekannte Version.
Kriterium: Erfolgreicher Probelauf (Rollback & Rollforward). Wiederanlaufzeit ≤ 5 Minuten.

R18: Prozess-Isolation & dedizierte Backing Services. Ein Service pro Container/Workload. Cron/Scheduler, Worker, Web getrennt. Queue/Session/Cache über spezialisierte Systeme (z. B. Redis), nicht über die Primärdatenbank.
Kriterium: Architektur-Review bestätigt Trennung. Keine Datenbank-Backends für Queue/Session/Cache. Separate Deploy-/Scale-Einheiten vorhanden.

R19: Modernisierter Entwicklungsfluss: Trunk-Based Development mit kurzlebigen Branches. Pre-Commit-Checks (Format/Lint/Tests/Secrets) & reproduzierbares, containerisiertes Dev-Setup.
Kriterium: Branch-Lebensdauer verkürzen. Onboarding (lokales Setup) in ≤ 10 Minuten reproduzierbar.

4 Transformation der DLRG-Anwendung „Kindersucharmband“

4.1 Einordnung und Zielbild der Anwendung „Kindersucharmband“

Das Projekt *„Kindersucharmband“* hilft den Einsatzkräften der DLRG an Nord- und Ostsee dabei, Kinder, die sich verlaufen haben, schnell wieder mit ihren Eltern zusammenzuführen [DLR]. Es ergänzt die klassischen, physischen Armbänder, die an den Wachstationen ausgegeben werden, um eine einfache digitale Lösung. Das *„Kindersucharmband“* dient dazu, das Projekt so zu transformieren, dass im Idealfall, alle abgeleiteten Anforderungen (R) erfüllt sind.

4.1.1 Ablauf und Nutzung

Vor Ort erhalten Erziehungsberechtigte an einer DLRG-Wachstation ein Armband mit einer eindeutigen Nummer [DLR]. Über die Website des Projekts können sie diese Nummer anschließend selbst registrieren. Dabei werden nur wenige Angaben benötigt: der Vorname des Kindes, eine Mobilnummer, die Nummer des Armbands und das geplante Urlaubsende. Nach der Registrierung verschickt das System eine SMS mit einem Bestätigungslink. Erst wenn diese SMS bestätigt wurde, wird der Eintrag aktiv und die Armbandnummer ist im System hinterlegt.

Kommt es vor, dass ein Kind verloren geht und von einer DLRG-Einsatzkraft gefunden wird, kann die Helferin oder der Helfer die Nummer des Armbands in einer geschützten Suchmaske eingeben. Daraufhin erscheint die hinterlegte Mobilnummer der Erziehungsberechtigten, sodass sie direkt angerufen werden können. Gleichzeitig wird dieser Vorgang im System protokolliert, und eine automatische SMS informiert die Eltern darüber, dass ihr Kind gefunden wurde und in Kürze ein Anruf der DLRG erfolgt.

Nach Ende des angegebenen Urlaubs werden alle Daten automatisch gelöscht. Auch über diese Löschung werden die Eltern per SMS informiert. Alternativ können sie ihre Daten schon früher entfernen, indem sie über ihr persönliches Profil, das nach der SMS-Bestätigung sichtbar ist, eine Löschung anstoßen.

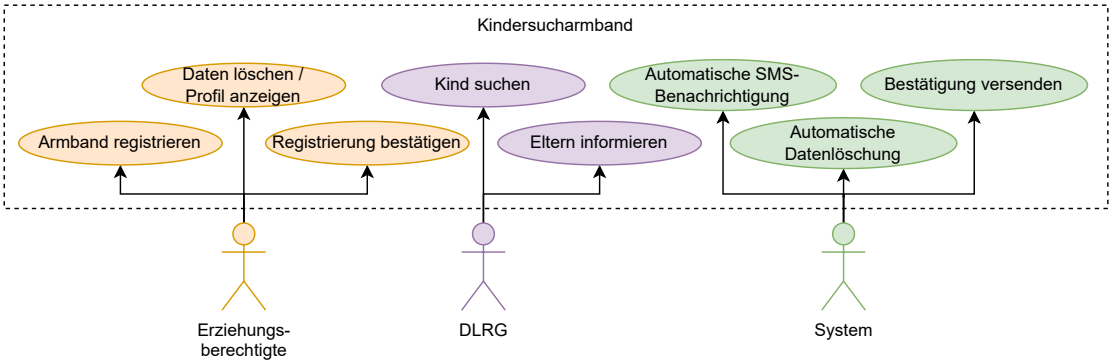

Abbildung 4.1: Use-Case-Diagramm der Anwendung *„Kindersucharmband“*. Es zeigt die zentralen Akteure (Erziehungsberechtigte, DLRG und System) sowie deren Interaktionen, die die funktionale Grundlage für die spätere DevOps-Transformation bilden.

Beteiligte und Rollen:

- **Erziehungsberechtigte:** Sie registrieren das Armband, bestätigen ihre Daten und können sie jederzeit einsehen oder löschen.
- **DLRG-Einsatzkräfte:** Sie greifen im Bedarfsfall über die geschützte Suchmaske auf die hinterlegten Kontaktdaten zu, um die Eltern zu informieren.
- **System:** Verantwortlich für automatisierte Datenlöschung sowie das Versenden von Benachrichtigungen.

Technische und organisatorische Anforderungen:

- **Datenschutz:** Es werden nur die notwendigsten Informationen erhoben, und alle Daten werden nach dem Urlaub automatisch gelöscht.
- **Zuverlässigkeit:** Gerade während der Urlaubssaison muss die Anwendung stabil und performant bleiben.
- **Nachvollziehbarkeit:** Jede Suchanfrage wird dokumentiert, ohne dass unnötig personenbezogene Daten offengelegt werden.

Technische Daten

Nachfolgend sind alle für die Transformation von der Anwendung *„Kindersucharmband“* benötigten Versionen und Abhängigkeiten aufgelistet:

Komponente	Version / Details
Framework	Laravel 12
PHP	8.4.13
Composer	2.8.12
Node	24.9.0
Composer (Kern)	`laravel/pennant` v1.16.1; `spatie/laravel-prometheus` v1.3.0
Composer (DEV)	`laravel/pint` 1.13.0; `laravel/sail` 1.26.0; `infection/infection` 0.31.2; `phpunit/phpunit` 12.0.0
Datenbanken	MySQL 9.4.0; Redis 8.2.1
Versionsverwaltung (VCS)	GitHub
Container Registry	GitHub Container Registry (GHCR)
CI/CD-Pipeline	GitHub Actions Workflows
GitOps-Operator	FluxCD

4.1.2 Bedeutung für die Transformation

Das „Kindersucharmband“ verbindet eine Webanwendung mit Backend-Logik, Datenbank, Hintergrundprozessen (z. B. Queues und Scheduler) und automatisierten Benachrichtigungen. Gleichzeitig unterliegt es strengen Datenschutzregeln und saisonalen Nutzungsschwankungen. Im Sommer herrscht sehr hoher Andrang, im Winter fast keiner.

Diese Mischung aus technischer Komplexität, Sicherheitsanforderungen und variabler Last macht die Anwendung zu einem idealen Beispiel, um moderne DevOps- und DevSecOps-Praktiken in der Praxis einzusetzen. Dazu zählen automatisierte Tests und Deployments, Containerisierung, sicheres Secret Management, Monitoring sowie ein vollständig reproduzierbarer Betrieb in Kubernetes.

Im weiteren Verlauf wird gezeigt, wie die ursprüngliche Anwendung technisch überarbeitet wurde, um Anforderungen (R) zu erfüllen und langfristig einen stabilen, sicheren Betrieb zu gewährleisten.

4.2 Zielprozess auf Basis von DevOps-Praktiken

4.2.1 Git-Repository und Branching-Strategie

Die bisherige featurebasierte Branch-Strategie mit dedizierter Testumgebung (siehe Abbildung 3.2) führte häufig zu unterschiedlichen Versionsständen und erschwerte die Zusammenarbeit. Im Zielprozess wird daher auf ein vereinfachtes Vorgehen nach dem Prinzip des *Trunk-Based Development* gesetzt [FHK18, Ham].

Anstelle langfristig bestehender Feature-Branches arbeiten die Entwickler in kurzen Zyklen direkt auf dem Hauptzweig `main`. Pull Requests dienen dabei in erster Linie der Code-Review, nicht der verzögerten Integration. Durch die geringe Lebensdauer der Branches werden Integrationskonflikte minimiert, und neue Änderungen stehen unmittelbar in einer stabilen Version zur Verfügung.

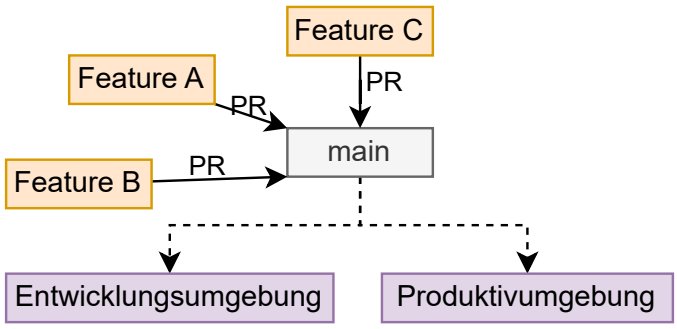

Abbildung 4.2: Trunk-Based Development: mehrere kurzlebige Branches, Integration über Pull Requests (PR) nach `main`, separate Testumgebung für Feedback entfällt. Features werden intern in Produktiv getestet und ausgerollt.

Die Rolle der bisherigen Testumgebung entfällt. An ihre Stelle tritt ein Feature-Flag-Konzept, das neue Funktionen bereits in der Produktivumgebung ausrollt, dort jedoch zunächst nur intern sichtbar macht (mehr Informationen dazu im Abschnitt 4.3.2).

4.2.2 Lokale Entwicklungsumgebung mit Containern

Für die lokale Entwicklung wird im Zielprozess *Laravel Sail* eingesetzt. Sail stellt eine containerisierte Entwicklungsumgebung bereit, die alle für Laravel typischen Komponenten umfasst und damit eine enge Annäherung an die Produktionsumgebung ermöglicht

[Doce]. Entwickler müssen dadurch weder PHP noch Datenbank- oder Caching-Systeme lokal installieren. Stattdessen kann eine reproduzierbare Umgebung mit einem einzigen Befehl bereitgestellt werden. Dieser Ansatz adressiert ein zentrales DevOps-Prinzip, nämlich die Minimierung von Abweichungen zwischen Entwicklungs- und Betriebsumgebungen [KHDW16, HF10].

Über die Option `php artisan sail:install` lassen sich unterschiedliche Dienste integrieren, etwa relationale und NoSQL-Datenbanken (MySQL, PostgreSQL, MariaDB, MongoDB), Caching-Systeme (Redis, Valkey, Memcached), Suchindizes (Meilisearch, Typesense), ein Storage-Emulator (MinIO), Messaging-Systeme (RabbitMQ), Testwerkzeuge (Mailpit, Selenium) sowie ein WebSocket-Server (Soketi) [Doce]. Im konkreten Projektkontext wurden lediglich MySQL und Redis benötigt, da sie die Kernanforderungen der Anwendung abdecken.

Konfigurationen und Secrets werden lokal über eine `.env`-Datei bereitgestellt, während in der Produktion ein dediziertes Geheimnismanagement *HashiCorp Vault* (siehe Abschnitt 4.3.3) zum Einsatz kommt. Dadurch wird ein klarer Unterschied zwischen Entwicklungs- und Betriebsumgebung etabliert, ohne dass Entwickler auf Komfort verzichten müssen. Dies entspricht der in DevOps verankerten Trennung von Zuständigkeiten bei gleichzeitiger Standardisierung der Schnittstellen [Kri22].

Darüber hinaus wird ein `Makefile` eingesetzt, um den initialen Projektaufbau und wiederkehrende Aufgaben zu automatisieren [AKNN+12]. Anstatt dass Entwickler einzelne Schritte wie die Installation von Abhängigkeiten, das Anlegen der `.env`-Datei, die Generierung des Anwendungsschlüssels via `php artisan key:generate`, das Starten der Container, das Ausführen von Migrationen und Seedern sowie Frontend-Builds manuell ausführen müssen, werden diese Prozesse in einem einzigen Befehl (`make setup`) gebündelt. Dies erleichtert insbesondere das Onboarding neuer Teammitglieder und stellt sicher, dass alle Entwickler schnell über eine konsistente, funktionsfähige Umgebung verfügen. Das `Makefile` ist in Listing 7.1.1 dargestellt.

Auf diese Weise wird *Fail-Fast*-Prinzip etabliert und sichergestellt, dass Fehler frühzeitig sichtbar werden.

4.2.3 CI/CD-Pipeline mit Qualitätssicherung und Sicherheitstests

Die im Projekt verwendete CI/CD-Pipeline folgt einem zweistufigen Ansatz, der schnelle Rückmeldungen für Entwickler mit strikten Qualitätsanforderungen vor dem Merge in den Hauptzweig `main` kombiniert. Ziel ist es, Fehler *(Fail-Fast)* und Sicherheitsrisiken frühzeitig zu erkennen (*Shift Left*) und gleichzeitig die Release-Stabilität zu erhöhen. Die Pipeline ist so gestaltet, dass jedes Commit prinzipiell deploybar ist, während Pull Requests gegen den Hauptzweig zusätzliche Hürden (Quality Gates) passieren müssen.

Frühe Checks durch Git Hooks

Noch vor der eigentlichen Pipeline greifen lokale Git Hooks [Git]. Diese erzwingen grundlegende Prüfungen wie Linting, Code-Formatierung, Unit-Tests oder Secret-Scans bereits bei der Ausführung, in diesem Fall, eines Commits (`git commit`). Entwickler erhalten damit direktes Feedback, sodass fehlerhafte Änderungen gar nicht erst in die zentrale Pipeline gelangen. Auf diese Weise wird ein *Fail-Fast*-Prinzip gestärkt. Probleme werden direkt lokal sichtbar und gelangen gar nicht erst in die zentrale CI/CD-Pipeline.

Da der standardmäßige Ordner für Git Hooks im Verzeichnis `.git/hooks` liegt, und dieser nicht versioniert wird, muss ein separater Ordner `.githooks` im Projektverzeichnis angelegt werden, und `git` so konfiguriert werden, dass dieser `.githooks`-Ordner standardmäßig verwendet wird. Um dies umzusetzen, wird `direnv` verwendet [dir25]. Das Tool ermöglicht es, bestimmte Befehle ausführen zu lassen sobald auf einen Ordner, in diesem Fall der Projektordner, zugegriffen wird. Hierfür benötigt das Tool im Projektordner die Datei `.envrc` mit den auszuführenden Befehlen, die den Pfad des Git Hooks-Ordner angeben:

```
export GIT_HOOKS_DIR=$PWD/.githooks
git config --local core.hooksPath "$GIT_HOOKS_DIR"
```

Listing 4.1: Die .envrc-Datei wird ausgeführt, sobald auf das Projektverzeichnis zugegriffen wird, wodurch der aktualisierte Pfad der Git Hooks konfiguriert wird.

Sobald ein `git commit`-Befehl ausgeführt wird, erfolgt die Ausführung des folgenden Ablaufs:

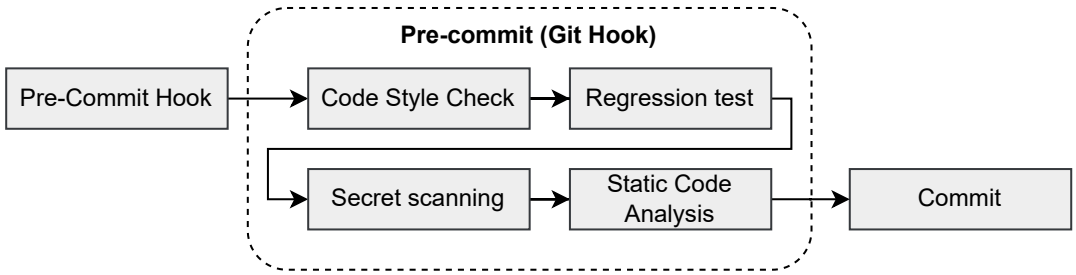

Abbildung 4.3: Die Abbildung veranschaulicht den Ablauf der Git Hooks (Pre-Commit). Das entsprechende Shell-Skript ist dem Anhang zu entnehmen (siehe Listing 7.1.2).

Sollte jenes Skript (siehe Listing 7.1.2) einen Exit-Code $\neq 0$ zurückgeben, wird der `git commit`-Befehl mit einem Fehler abgebrochen [Git]. Somit wird sichergestellt, dass keine

fehlerhaften Artefakte in das Versionskontrollsystem gelangen. Weitere lokale Git Hooks sind `prepare-commit-msg`, `commit-msg` und `post-commit` [Git].

Schnelle Rückmeldung: die Fastlane (Feature-Branches)

Bei jedem Push auf einen Feature-Branch (z.B. `feature/*` und nicht-`main`) wird über GitHub Actions die Fastlane ausgelöst (siehe Anhang 7.2.1). Sie dient der kurzfristigen, iterativen Rückmeldung ohne lange Wartezeiten und bündelt die zentralen Qualitätssignale:

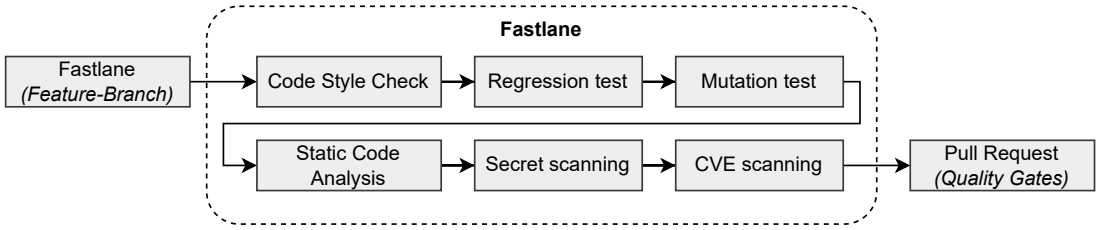

Abbildung 4.4: Die Abbildung veranschaulicht den Ablauf der Fastlane (Feature-Branch). Zunächst werden Linting und Regressionstests (Unit- und Feature-Tests) ausgeführt, um syntaktische Fehler zu erkennen und die Kernlogik abzusichern. Anschließend werden Mutations-Tests zur Wirksamkeitskontrolle der Testsuite durchgeführt. Im weiteren Verlauf werden sogenannte Secret- und Vulnerability-Scans (CVE) durchgeführt, um das Risiko von Datenlecks und bekannten Schwachstellen zu minimieren. Die entsprechende Definition der *Fastlane* ist dem Anhang zu entnehmen (siehe Anhang 7.2.1).

Die Fastlane erzwingt noch keine harten Schwellwerte (Quality Gates), erzeugt jedoch klare Signale und Berichte, die Entwickler unmittelbar nutzen können.

Merge-Sicherheit: PR Quality Gates (Pull Requests nach `main`)

Wird ein Pull Request gegen den Hauptzweig `main` eröffnet, greift eine intensivere Pipeline mit formalen Eintrittskriterien. Zusätzlich zu den Schritten der Fastlane werden hier durchgeführt:

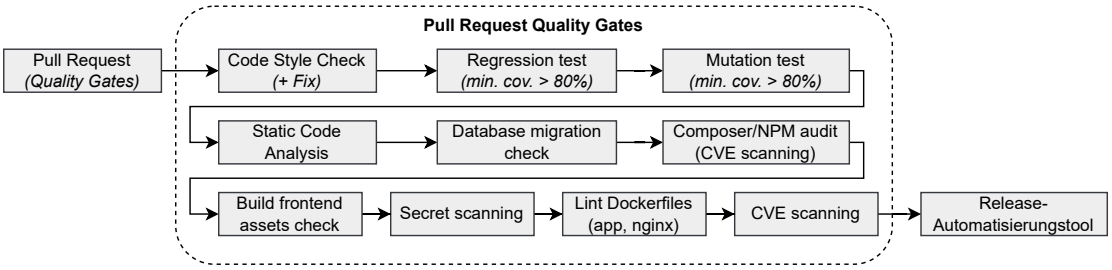

Abbildung 4.5: Die Abbildung veranschaulicht den Ablauf der PR Quality Gates (Pull Requests nach `main`). Zunächst werden Linting mit automatischen Fixes und Regressionstests (Unit- und Feature-Tests) mit einer projektspezifischen Untergrenze (beim „Kindersucharmband“ 80 %) ausgeführt. Anschließend werden Mutations-Tests mit einem MSI-Threshold von 80 % durchlaufen. Im weiteren Verlauf werden Datenbank-Migrations-Checks (Trockenlauf/Validierung), Composer und NPM-Audits, Build-Asset-Test (Frontend), Dockerfile-Linting von App und Nginx (siehe Abschnitt 4.3.1) sowie Secret- und Vulnerability-Scans (CVE) durchgeführt. Die entsprechende Definition des *PR Quality Gates* ist dem Anhang zu entnehmen (siehe Anhang 7.2.1).

Nur wenn alle Quality Gates erfolgreich durchlaufen werden, ist ein Merge möglich. Damit wird verhindert, dass fehlerhafte oder unsichere Artefakte in den Hauptzweig `main` gelangen.

Begründung der Metriken: Die Wahl einer Testabdeckung von 80 % folgt einem pragmatischen Kosten-Nutzen-Verhältnis, das dem Pareto-Prinzip (80/20-Regel) entspricht [Jur54]. Empirisch zeigt sich, dass der überwiegende Anteil potenzieller Fehler bereits durch eine Abdeckung von etwa 70–80 % adressiert wird, während der Aufwand zur Erhöhung auf nahezu vollständige Abdeckung exponentiell steigt [IH14, MNDT09]. Der gewählte Schwellenwert repräsentiert somit eine Balance zwischen Risikoreduktion und ökonomischer Effizienz der Qualitätssicherung.

Automatisierungsdetails und Artefakte: Die gesamte Pipeline ist in *GitHub Actions* umgesetzt. Workflows sind in YAML-Dateien versioniert und können damit wie der Anwendungscode selbst nachvollzogen und auditiert werden (siehe Kapitel 7.2). Sowohl Fastlane als auch Quality Gates nutzen Caching (Composer, npm) zur Laufzeitoptimierung und erzeugen standardisierte Reports (Linting, SCA, Coverage, MSI, Audits), die automatisiert an Pull Requests angehängt werden. Build-Schritte validieren, dass

Frontend-Assets reproduzierbar sind. Security-Reports werden als Pflichtbefunde gewertet. Bekannte kritische Schwachstellen führen zum Abbruch der Pipeline.

Automatisches Versionieren und Release-Erstellung

Nach erfolgreichem Merge in den Hauptzweig `main` wird das Release-Automatisierungstool *release-please* getriggert [Gooa]. Das Tool generiert auf Basis der getätigten Commit-Nachrichten eine semantische Version, die beispielsweise die Bezeichnung `v1.2.3` trägt, erstellt den zugehörigen Git-Tag, aktualisiert konsistent alle Stellen, an denen die Version referenziert wird (z.B. `composer.json`), und legt das Release an [Gooa]. Damit sind Versionierung und Changelog nachvollziehbar automatisiert und revisionssicher.

Die YAML-Definition befindet sich im Anhang unter Abschnitt 7.2.1.

Build, Härtung und Publikation der Container-Artefakte

Sobald ein neuer Git-Tag existiert, werden die Anwendungs- (Laravel) und Nginx-Images gebaut. Danach folgt ein mehrstufiger Security-Gate. Zunächst wird mit dem Tool `Trivy` ein Vulnerability-Scan (CVE) der beiden Images durchgeführt [Tri]. Sollte die Anzahl der gefundenen CVEs oberhalb der definierten Schweregrade liegen, führt dies zum Abbruch des Build-Vorgangs. Zusätzlich werden für beide Images mit dem Tool *cosign* Signaturen generiert sowie eine Software Bill of Materials (SBOM) erzeugt und diese ebenfalls signiert (Attestierung) und im GHCR abgelegt [Sig25b, Fed]. Alle Artefakte werden in die GHCR gepusht und eindeutig mit Commit-Hash und Release-Tag gekennzeichnet. So ist jede veröffentlichte Version reproduzierbar und auditierbar.

Durch dieses Vorgehen wird die Software-Lieferkette gehärtet. SBOMs schaffen Transparenz, welche Bibliotheken und Abhängigkeiten in den Images enthalten sind. Signaturen gewährleisten die Herkunft der Artefakte (Supply Chain Integrity). Vulnerability-Scans verhindern, dass bekannte Sicherheitslücken (CVE) in Produktion gelangen. Dieser Ablauf macht Security zu einem integralen Bestandteil des Delivery-Prozesses.

Die YAML-Definition befindet sich im Anhang unter Abschnitt 7.2.1.

Helm-Chart als OCI-Artefakt

Das Helm-Chart (siehe Abschnitt 4.2.5) wird in einem eigenständigen Repository gepflegt und besitzt eine separate Release-Automatisierung. Veröffentlichungen erfolgen als Open Container Initiative (OCI)-Artefakt in der Container-Registry (GHCR). Der Git-Tag entspricht jeweils der *Chart*-Release-Version. Das Anwendungs-Repository des

Kindersucharmbands ist davon unabhängig, d. h. Chart- und Image-Versionen sind entkoppelt und folgen eigenen Release-Zyklen.

Weitere Informationen zum Helm-Chart sind dem Abschnitt 4.2.5 zu entnehmen.

Einbindung in den GitOps-Flow

Das veröffentlichte Helm-Chart und die Container-Images bilden gemeinsam die *Single Source of Truth* für die Auslieferung. Beide Artefakte werden unabhängig versioniert und über Git referenziert. Der GitOps-Operator (FluxCD) überwacht das GitOps-Repository und synchronisiert die darin definierten Spezifikationen des `HelmRelease`-Objekts mit den Zielumgebungen (siehe Abschnitt 2.3). Änderungen an diesen Manifesten, wie z. B. neue Chart- oder Image-Versionen werden automatisch erkannt und im Cluster angewendet (siehe Abschnitt 2.3).

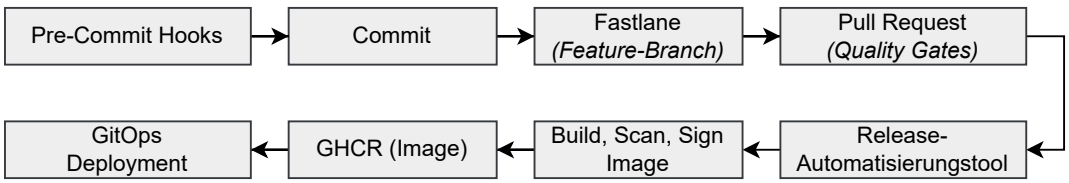

Abbildung 4.6: Ablauf der CI/CD-Pipeline: Von Pre-Commit Hooks und Commit über Fastlane in die Quality Gates, automatische Release-Erstellung, Build, Scan und Signatur bis zum Container-Registry (GHCR) und GitOps-Deployment.

Zur Sicherstellung, dass die Definitionen stets aktuelle Artefaktversionen referenzieren, wird das Tool *Renovate* eingesetzt [Ren]. Es prüft in regelmäßigen Abständen die im `HelmRelease` angegebenen Tags und vergleicht sie mit den im GHCR veröffentlichten Versionen. Erkennt Renovate eine neue Version, erstellt es automatisch einen Pull-Request im GitOps-Repository, der die Tag-Angaben aktualisiert. Nach Merge des Pull-Requests erkennt der GitOps-Operator die geänderte Manifestversion und führt den entsprechenden Rollout durch.

Dieses Zusammenspiel aus *Renovate* (Automatisierung der Versionspflege) und *FluxCD* (deklarative Auslieferung) stellt sicher, dass Deployments nachvollziehbar, reproduzierbar und auditierbar bleiben, ohne manuelle Eingriffe im Cluster vorzunehmen.

4.2.4 Zusammenfassung

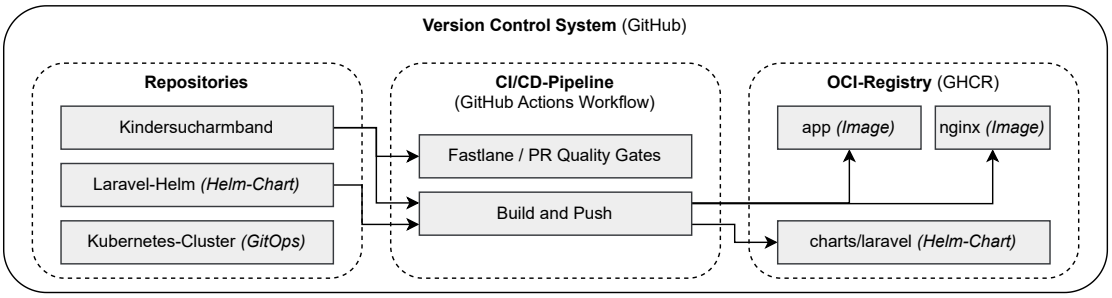

Abbildung 4.7: Überblick über Repositories, CI/CD-Pipeline und Artefakte (GHCR).

Entwicklungsfluss (Repo *„Kindersucharmband“*)

- **Feature-Branches:** Entwicklung auf separaten Branches.
- **Sofortiges Feedback:** Git Hooks prüfen Commits (*Fail-Fast*).
- **Push:** `Fastlane` startet automatisch.
- **Pull-Request gegen `main`:** strengere *Quality Gates*.
- **Nach dem Merge:** Release-Automatisierung erzeugt ein Release und triggert Build von den Container-Images *App* und *Nginx*.

Cluster & GitOps (Repo *Kubernetes-Cluster*)

- Alle Kubernetes-Manifeste liegen im Repository, inkl. `HelmRelease` für *„Kindersucharmband“*.
- CI/CD des Clusters führt `Renovate` in Intervallen aus:
  - prüft auf neue Versionen / Image-Tags (z. B. App, `Nginx`),
  - erstellt automatisch Pull Requests (PR) mit den Updates.
- Das Kubernetes-Cluster synchronisiert sich kontinuierlich mit dem Repo:
  - freigegebene Änderungen werden automatisiert übernommen und ausgerollt.

Helm-Chart

- Unabhängig vom Projekt *„Kindersucharmband“*.
- Dient als generische Hülle. Kompatibel auch für andere Laravel-Anwendungen (siehe Abschnitt 4.2.5).

4.2.5 Deployment nach Kubernetes

Das Deployment in das Kubernetes-Cluster erfolgt deklarativ über ein Helm-Chart, das sämtliche Anwendungskomponenten und Infrastrukturressourcen umfasst. Die Chart-Artefakte
(Anwendungs- und Nginx-Images sowie das Chart selbst) werden versioniert in der Container Registry (GHCR) bereitgestellt und im Rahmen des GitOps-Prozesses in die Zielumgebungen synchronisiert. Umgebungsunterschiede werden ausschließlich über das `HelmRelease` parametrisiert (siehe Listing 7.3.14). Die Templates bleiben identisch und damit reproduzierbar. Ziel ist es, eine Betriebsumgebung zu schaffen, die automatisiert, skalierbar und sicher betrieben werden kann.

Exemplarische Chart-Auszüge sind im Anhang unter Abschnitt 7.3 zu finden.

Anwendungs-Layer (PHP-FPM, Nginx) und horizontale Skalierung

Die Laravel-Anwendung läuft als `Deployment` („app“) auf PHP-FastCGI Process Manager (FPM). Ein separates `Deployment` für Nginx übernimmt die HTTP-Serving-Schicht. Alle drei Deployments (App, Nginx und Worker) starten mit jeweils **2 Replikas** und sind über Horizontal Pod Autoscaling (HPA) dynamisch skalierbar [The25d].

Für App und Worker ist eine Skalierung zwischen 2 und 8 Replikas möglich (siehe Listing 7.3.10). Die Zielauslastung liegt bei 60 % CPU und 70 % Speicher, sodass Lastspitzen frühzeitig abgefangen werden können. Nginx skaliert ebenfalls zwischen 2 und 8 Replikas, allerdings ausschließlich auf Basis der CPU-Auslastung. Nginx wird über einen `Service` exponiert und ist per `Ingress` erreichbar [The25u, The25k]. Die Transport Layer Security (TLS)-Terminierung findet am Ingress statt, während Nginx intern HTTP auf Port 80 spricht (siehe Abbildung 4.11). Laufzeitkonfigurationen werden über dedizierte `ConfigMaps` getrennt von den Build-Artefakten verwaltet [The25g].

Laravel-spezifische Betriebsaufgaben als eigenständige Ressourcen

Die früher im Container gebündelten Nebenprozesse (siehe Abbildung 3.3) wurden in eigenständige Kubernetes-Workloads überführt:

- **Scheduler:** Ein `CronJob` triggert `php artisan schedule:run` jede Minute [The25h, Lar25g]. Parallelausführung ist deaktiviert, `startingDeadlineSeconds` ist auf 60 s gesetzt (siehe Listing 7.3.5).
- **Queue-Worker:** Ein separates `Deployment` („worker“) verarbeitet asynchrone Jobs [The25j, Lar25f]. Es startet mit 2 Replikas und kann dynamisch bis zu 8 Replikas skalieren (siehe Listing 7.3.4).
- **Migrationen und Seeder:** Datenbankmigrationen laufen automatisiert als `Job` mit den Helm-Hooks `pre-upgrade` und `post-install` (siehe Listing 7.3.6 & 7.3.8) [The25b, Lar25c]. Seeder werden nachgelagert als eigener `Job` ausgeführt (siehe Listing 7.3.7) [Lar25d]. Beide Jobs sind idempotent und blockieren bei Fehlern das Release, sodass fehlerhafte Deployments gar nicht erst in Betrieb gehen [The25l].

Besonderes Augenmerk liegt auf dem Umgang mit Schemaänderungen. Klassische, monolithische Migrationen führen während des Rollouts zu einem sogenannten Inkompatibilitätsfenster [Wel18]. Die Datenbank befindet sich bereits im neuen Schema, während noch alte Applikationsinstanzen laufen, die das alte Schema erwarten. Dadurch entstehen kurzzeitige Ausfälle oder fehlerhafte Antworten (vgl. Abbildung 4.8). Der Rollout ist genau jener Zeitraum, in dem Migrationen und Applikationswechsel parallel stattfinden, und in dem Inkonsistenzen auftreten können [Wel18, Lar25c].

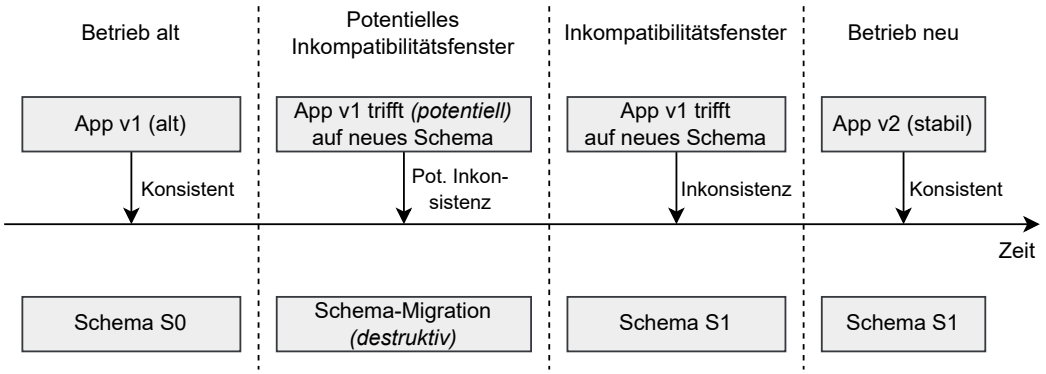

Abbildung 4.8: Monolithische Migration: destruktive Änderung erzeugt ein Inkompatibilitätsfenster während des Rollouts. **(1)** Destruktive Änderung während des Rollouts. **(2)** Alte App v1 erwartet Schema S0, die Datenbank ist bereits S1. **(3)** Kurzzeitige Ausfälle oder Fehlerantworten.

Um dieses Problem zu vermeiden, wurde das *Expand/Contract*-Verfahren eingeführt [Wel18]. Dieses teilt Änderungen in zwei Phasen auf:

1. **Expand (additiv):** Zunächst werden nur erweiternde Änderungen am Schema vorgenommen, die sowohl von der alten als auch von der neuen Applikationsversion verarbeitet werden können [Wel18]. Beispiele sind das Hinzufügen neuer Spalten oder Tabellen.
2. **Contract (destruktiv):** Erst wenn die neue Version stabil läuft und alle Altinstanzen außer Betrieb genommen wurden, werden überflüssige Strukturen in einem nachgelagerten Schritt entfernt [Wel18].

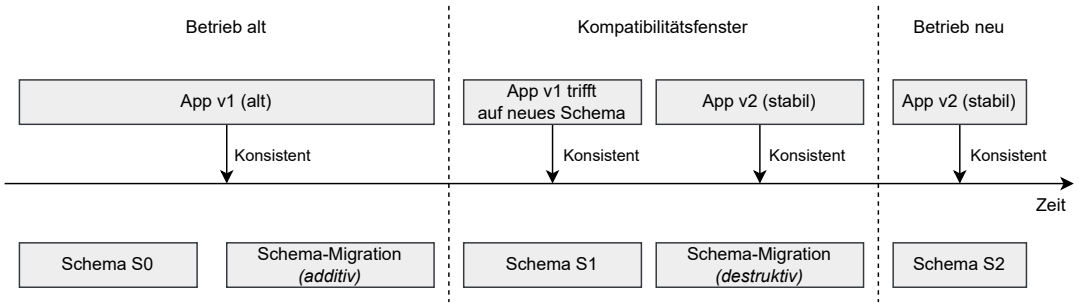

Abbildung 4.9: Expand/Contract-Migration: additive Änderung ermöglicht ein Kompatibilitätsfenster während des Rollouts. **(1)** Zunächst wird das Schema nur erweitert (additiv). **(2)** Sowohl App v1 als auch App v2 können mit Schema S1 arbeiten. **(3)** Nach erfolgreichem Rollout entfernt die destruktive Änderung (Contract) die Altlasten.

Da Laravel ein solches Verfahren nicht nativ unterstützt, wurden zwei zusätzliche Artisan-Commands eingeführt: `migrate:expand` und `migrate:contract` [Lar25a]. Intern sind dies lediglich Proxy-Befehle, die `php artisan migrate --path` für die jeweiligen Verzeichnisse `database/migrations/expand` und `database/migrations/contract` aufrufen (siehe Anhang 7.4.1) [Lar25a, Lar25c]. Dadurch bleibt die Codekomplexität gering, während eine klare Trennung additiver und destruktiver Änderungen gewährleistet ist.

Zwischen diesen beiden Phasen werden Seeder-Prozesse ausgeführt, um Daten in die neu angelegten Strukturen zu übernehmen. Dabei ergibt sich eine Analogie zu den beschriebenen monolithischen Migrationen: Auch beim Seeding existiert ein Inkompatibilitätsfenster. Während App v2 bereits auf die neuen Strukturen zugreifen kann, stehen die dazugehörigen Inhalte möglicherweise noch nicht oder nur teilweise bereit. Die Anwendung läuft in diesem Zeitraum zwar technisch fehlerfrei, zeigt jedoch veraltete oder unvollständige Daten an.

Theoretisch ließe sich dieses Problem analog zu Expand/Contract entschärfen, indem zunächst ausschließlich additive Seeding-Schritte ausgeführt werden, danach der Wechsel auf App v2 erfolgt und erst anschließend destruktives Seeding stattfindet, bevor im letzten Schritt die Contract-Migration durchgeführt wird.

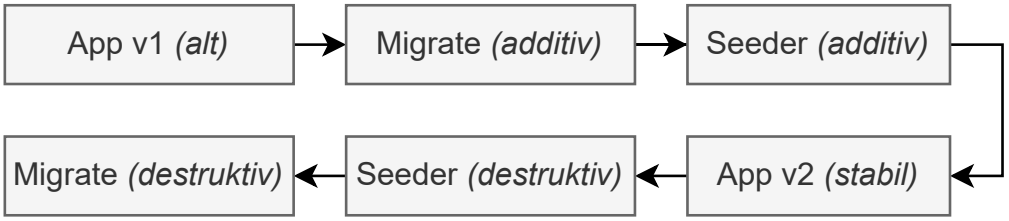

Abbildung 4.10: Zeitlicher Ablauf von Expand/Contract-Migration mit ergänzenden Seeding-Schritten.

In der Praxis würde ein solches Vorgehen jedoch die Komplexität von Code und Orchestrierung deutlich erhöhen.

Im Anwendungsfall *„Kindersucharmband“* ist das Risiko vernachlässigbar, da ausschließlich FAQ-Einträge aktualisiert werden (siehe Listing 7.4.2). Kommt es während des Rollouts zu Verzögerungen, zeigt App v2 höchstens die vorherigen FAQ-Einträge aus App v1 an, die abwärtskompatibel sind und die Stabilität nicht beeinträchtigen.

Während sich das Expand/Contract-Verfahren für Migrationen praktisch etablieren lässt, bleibt die analoge Anwendung auf Seeder-Prozesse in diesem Kontext ein theoretisches Konstrukt.

Für die Umsetzung im Betrieb konzentriert sich das Deployment daher auf die Migrationen, deren Ablauf in Kubernetes-Umgebungen durch Helm-Hooks gesteuert wird:

- **Expand:** läuft als Hook-Job bei `pre-upgrade` und `post-install` (Gewicht 10) [The25b].
- **Seeder:** läuft nach `post-upgrade` und `post-install` (Gewicht 15) [The25b].
- **Contract:** läuft nach `post-upgrade` (Gewicht 20) [The25b].

Damit ist sichergestellt, dass zuerst die kompatiblen Erweiterungen eingespielt werden, dann die Daten befüllt werden, und erst im Anschluss das Schema von Altlasten bereinigt wird. Der Betrieb bleibt während des gesamten Rollouts konsistent und durchgehend verfügbar.

Daten- und State-Layer

Die Anwendung nutzt MySQL (`StatefulSet`) als Primärdatenbank und Redis (`StatefulSet`) für Caching, Sessions und Queues [The25v, Lar25b]. Beide Komponenten sind über dedizierte `Services` erreichbar und persistieren ihren Datenbestand über ein Persistent Volume Claim (PVC) [The25u, The25o]. Für das *„Kindersucharmband“* selbst ist kein zusätzlicher Speicher notwendig, da weder öffentliche Uploads noch Artefakte anfallen. Das Chart sieht jedoch optional ein `sharedStorage` vor, das perspektivisch von anderen Laravel-Anwendungen genutzt werden kann (siehe Anhang Listing 7.2).

Rollout-Strategien und Verfügbarkeit

Die Deployments (App, Nginx, Worker) verwenden das RollingUpdate-Verfahren [The25z]. Mit `maxUnavailable: 0` wird verhindert, dass Pods abgeschaltet werden, solange keine neue Instanz bereitsteht. `maxSurge: 25%` erlaubt temporär zusätzliche Pods, um Lastspitzen während eines Rollouts abzufangen [The25z]. Ergänzend werden Pod Disruption Budget (PDB) mit `maxUnavailable: 1` eingesetzt, um sicherzustellen, dass bei Wartungen oder Node-Drains mindestens eine Instanz verfügbar bleibt [Kub25b]. Zusammen ermöglichen diese Mechanismen Zero-Downtime-Updates und reduzieren das Risiko von Ausfällen erheblich. Dadurch entsteht ein deutlicher Fortschritt gegenüber dem alten Zustand, in dem Neustarts der Container unmittelbar zu Ausfallzeiten führten.

Netzwerkisolation nach dem Least-Privilege-Prinzip

Strikte `NetworkPolicies` erzwingen eine Zero-Trust-Kommunikation im Cluster [The25m]. Die App darf eingehend ausschließlich von Nginx erreicht werden und kommuniziert ausgehend nur mit Redis und MySQL. Redis akzeptiert ausschließlich Anfragen von App, Worker und Scheduler. MySQL wiederum nimmt nur Verbindungen von App, Worker, Scheduler, Seeder und Migrator entgegen. Seeder und Migrator besitzen keinerlei weitere Netzwerkrechte und können ausschließlich auf die MySQL-Datenbank zugreifen. Worker dürfen nach außen lediglich verschlüsselte HTTPS-Verbindungen aufbauen (z. B. für SMS-Versand). Alle nicht explizit erlaubten Zugriffe sind blockiert. Damit wird die Angriffsfläche reduziert und jede Workload verfügt nur über die minimal notwendigen Verbindungen.

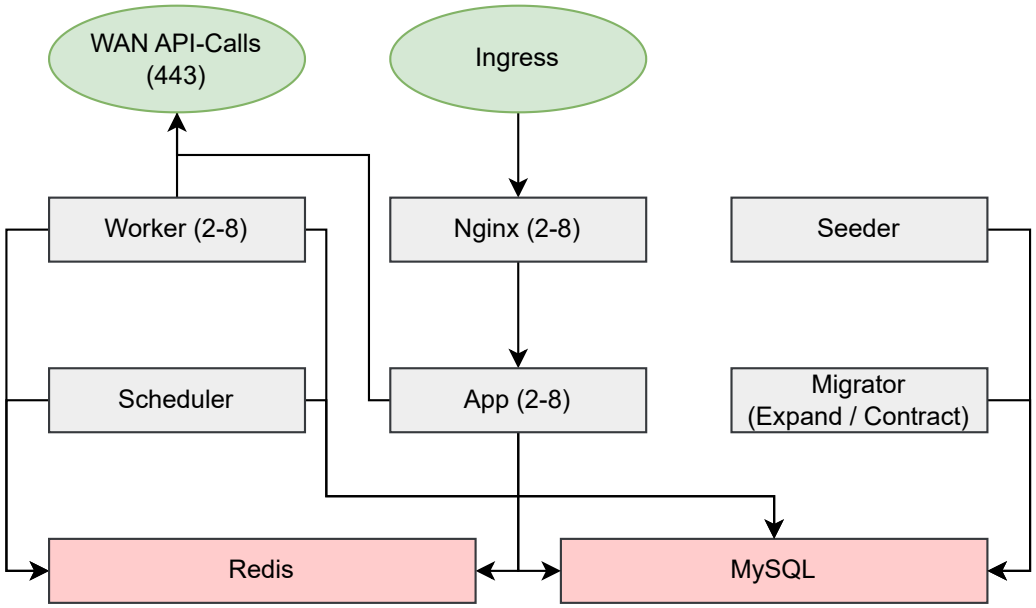

Abbildung 4.11: Makro-Übersicht erlaubter Verbindungen zwischen zentralen Workloads gemäß NetworkPolicies.

Eine exemplarische `NetworkPolicy` für das `Deployment` *App* ist im Anhang in 7.3.9 aufgeführt.

Sicherheitskontext und Härtung

Alle Container (Pods) laufen mit restriktiven `securityContext`-Vorgaben: Prozesse starten als Non-Root unter dedizierten UIDs/GIDs, Linux-Capabilities sind entfernt, Privilege Escalation ist deaktiviert, das Root-Dateisystem ist schreibgeschützt, und das standardisierte Seccomp-Profil (`RuntimeDefault`) ist aktiv [The25x].

```
securityContext:
  runAsUser: 1000          # Non-Root UID
  runAsGroup: 3000         # dedizierte GID
  runAsNonRoot: true
  allowPrivilegeEscalation: false
  readOnlyRootFilesystem: true
  capabilities:
    drop: ["ALL"]
  seccompProfile:
```

```
    type: RuntimeDefault
```

Listing 4.2: Beispielhafter `securityContext` für Pods

Ergänzend nutzen sämtliche Deployments Liveness-, Readiness- und Startup-Probes (siehe Listing 7.3.3) [The25y]. Pods werden nur dann in den Datenpfad aufgenommen, wenn sie betriebsbereit sind, und fehlerhafte Instanzen werden automatisch ersetzt [The25z, The25y]. Im Vergleich zum alten Setup, das ohne Sicherheitsrestriktionen und Probes auskam, bedeutet dies einen klaren Qualitätssprung: Sicherheit ist nun nicht nachträglich ergänzt, sondern von Beginn an integraler Bestandteil („Security by Default“).

Ressourcenmanagement und Startverhalten

Sämtliche Workloads definieren `requests` und `limits` für CPU und Speicher [The25s]. In Kombination mit den Probes wird sichergestellt, dass Pods erst dann Traffic verarbeiten, wenn sie stabil laufen, und dass fehlerhafte Container automatisch ersetzt werden (Self-Healing). Dadurch lassen sich Ausfallzeiten durch langsame Starts oder instabile Instanzen vermeiden.

Logging und Observability

Ein wesentliches Ziel der Transformation war die Integration einer modernen Observability-Plattform. Während Laravel zuvor lokal in `storage/logs` schrieb und Logs damit nur containerintern verfügbar waren, werden Ausgaben nun konsequent nach `stdout/stderr` geleitet. Um dieses Verhalten in Laravel umzusetzen, reicht es aus die Variable in der `.env`-Datei folgendermaßen zu definieren: `LOG_CHANNEL=stderr` zu setzen [Lar25e]. Dies entspricht den Best Practices im Container-Umfeld und erlaubt die zentrale Aggregation durch die Plattform [Doc25f, Kub25a].

Für Monitoring und Auswertung wird ein Open-Source-Stack eingesetzt:

- **Prometheus** sammelt Metriken wie Ressourcenauslastung, Latenzen oder Queue-Längen.
- **Loki** aggregiert Logs, die durch strukturierte Ausgabe (JavaScript Object Notation (JSON)) und Labels (Service, Namespace, Commit-Version) gezielt abgefragt werden können.
- **Grafana** dient als Visualisierungsschicht und ermöglicht die Definition von Dashboards und Alarmierungen auf Basis von Metriken und Logs.

In Verbindung mit Probes entsteht so ein selbstheilendes System, das kontinuierliches Feedback liefert und frühzeitige Reaktionen auf Abweichungen ermöglicht.

Eine detaillierte Erörterung des Themas erfolgt im Abschnitt 4.3.4.

Zusammenfassung

Das Helm-Chart transformiert die Anwendung von einer monolithisch gekoppelten Serverlösung zu einem klar geschnittenen, skalierbaren und sicherheitsgehärteten Kubernetes-Setup. Orchestrierungsaufgaben (Migrationen, Seeder, Scheduler, Worker) sind als eigenständige Ressourcen modelliert, die Netzwerkoberfläche ist minimal, und die Auslieferung erfolgt deklarativ und auditierbar. Persistenter Speicher (PVC) ist für das *„Kindersucharmband“* nicht erforderlich, aber als Option vorgesehen. Durch Rolling-Update-Strategien, Pod Disruption Budgets (PDB) und moderne Observability-Werkzeuge wie Prometheus, Loki und Grafana werden Verfügbarkeit, Transparenz und Betriebssicherheit erheblich gesteigert. Sicherheit, Skalierbarkeit und Automatisierung sind damit im gesamten Deployment-Prozess fest verankert.

Tabelle 4.1: Vergleich des Deployments vor und nach der Umstellung

Aspekt	**Vor der Umstellung**	**Nach der Umstellung**
Rollout-Strategie	Container-Neustarts führten zu Ausfallzeiten von bis zu einer Minute.	RollingUpdate mit `maxUnavailable: 0` und `maxSurge: 25%`. PodDisruptionBudgets sichern Verfügbarkeit (`maxUnavailable: 1`). Zero-Downtime-Updates.
Replikas und Skalierung	Meist eine Instanz pro Service, keine automatische Skalierung.	Alle Deployments starten mit 2 Replikas. HPA skaliert App und Worker zwischen 2–8 Replikas (CPU 60%, RAM 70%). Nginx skaliert zwischen 2–8 Replikas (CPU-basiert).

Aspekt	Vor der Umstellung	Nach der Umstellung
Migrationen	Manuell durch Entwickler nach jedem Deployment. Häufig inkonsistente Datenstände.	Automatisierte Jobs mit Helm-Hooks (`pre-upgrade`, `post-install`). Idempotent, blockieren bei Fehlern den Release. Additive Änderungen problemlos. Destruktive Änderungen erfordern Expand/Contract-Verfahren.
Fehler-Handling bei Migrationen	Fehlerhafte Migrationen führten zu inkonsistenten Systemzuständen.	Fail-Fast: fehlerhafte Migrationen stoppen den Rollout automatisch.
Nebenprozesse	Supervisor und Cronjobs liefen im selben Container wie die Anwendung.	Getrennte Workloads: Scheduler als `CronJob`, Queue-Worker als eigenes `Deployment`. Bessere Wartbarkeit und Skalierbarkeit.
Logging	Logs ausschließlich lokal im Container (`storage/logs`). Keine zentrale Sammlung oder Auswertung.	Logs ausschließlich nach `stdout`/`stderr`. Zentrale Aggregation mit Loki, Korrelation über strukturierte Logs (JSON, Labels).
Monitoring und Observability	Kein zentrales Monitoring, keine Metriken oder Alarmierung.	Vollständiger Stack: Prometheus für Metriken, Loki für Logs, Grafana für Visualisierung und Alerting. Probes (Readiness, Liveness, Startup) für Self-Healing.
Sicherheit	Container liefen ohne Einschränkungen, Root-User, keine Netzwerktrennung.	Restriktive `securityContext`-Vorgaben (Non-Root, keine Capabilities, ReadOnly-RootFS, Seccomp). Strikte `NetworkPolicies`. Security by Default.
Rollback-Strategien	Keine Möglichkeit zum schnellen Zurückspringen auf eine stabile Version.	Rollbacks über Git-Revert oder Version-Pins im GitOps-Workflow jederzeit möglich.

4.3 Produktivumgebung nach der Umstellung

Abbildung 4.12 stellt die Zielarchitektur abstrahiert dar. Anstelle mehrerer Server (siehe Abbildung 3.1) tritt ein gemeinsames Cluster mit logisch getrennten Bereichen. Die Server-Infrastruktur der DLRG weist 6 dedizierte Server auf, weshalb die Wahl der Cluster-Nodes entsprechend gewählt wurde, um das Risiko eines Single Point of Failure (SPoF) zu reduzieren. Eine übergreifende „Observability & Security“-Schicht ergänzt Monitoring, Logging, Geheimnisverwaltung und Richtliniendurchsetzung (siehe Abschnitt 4.3.3 & 4.3.5).

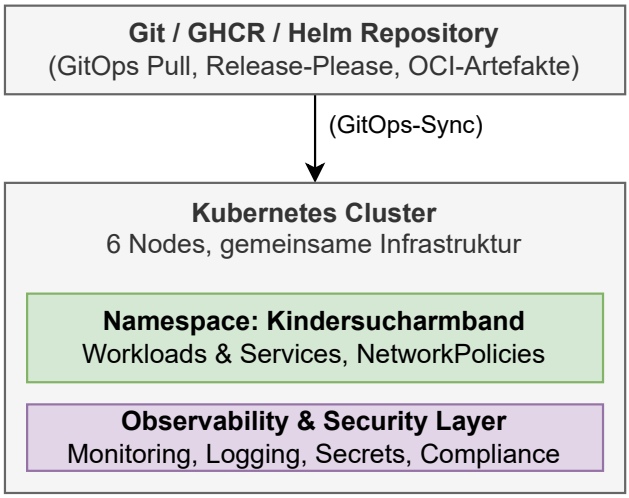

Abbildung 4.12: Makro-Darstellung des Kubernetes-Clusters.

4.3.1 Containerisierung mit Multistage-Builds

In der produktiven Zielumgebung setzt die Anwendung konsequent auf Multistage-Builds [Doc25d]. Dabei werden Build- und Laufzeit werden in mehrere `FROM`-Stufen getrennt [Doc25d]. Artefakte werden per `COPY --from=<stage>` selektiv in das finale Image übernommen, sodass am Ende nur die zur Ausführung benötigten Dateien enthalten sind [Doc25a, OCI25]. Das reduziert die Angriffsfläche (keine Compiler/Paketmanager im Endimage), verkleinert die Image-Größe und beschleunigt Pulls und Rollouts. [Doc25a, Doc25b]

Für das PHP-FPM-Image werden die Frontend-Assets zunächst in einer Node-Stage gebaut und die PHP-Abhängigkeiten in einer Composer-Stage aufgelöst. Das finale Lauf-

zeitimage basiert auf einem schlanken `php:8.4-alpine-FPM` und enthält ausschließlich Anwendungscode, optimierte Vendor-Pakete und vorkompilierte Assets. Compiler, Paketmanager und temporäre Artefakte verbleiben in den frühen Stages. Das Nginx-Image wird analog zweistufig erzeugt und liefert ausschließlich statische Dateien aus. Beide Container laufen unter nicht-privilegierten Benutzern. Die Entkopplung von App (PHP-FPM) und Webserver (Nginx) folgt dem Single-Responsibility-Prinzip und vereinfacht Skalierung, Rollouts und Fehlerdiagnose [Doc25e].

Im Anhang befindet sich sowohl die `Dockerfile` für die App als auch für den Nginx. Diese sind unter dem entsprechenden Abschnitt im Dokument mit der Nummer 7.5 aufgeführt.

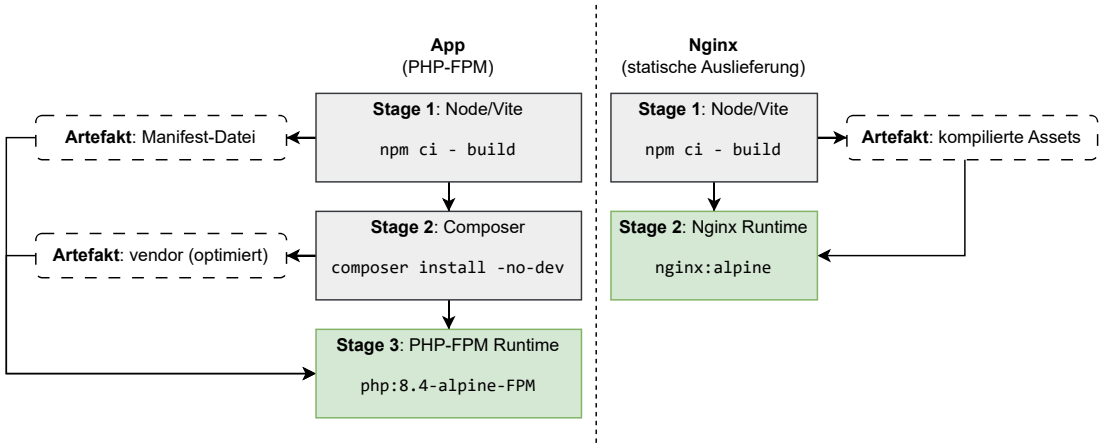

Abbildung 4.13: Multistage-Build-Pipelines für App (PHP-FPM) und Nginx: Build-Stages *(graue Rechtecke)* erzeugen Assets *(gestrichelte Rechtecke)* bzw. Dependencies. Die finalen Runtime-Images übernehmen nur die benötigten Artefakte *(grüne Rechtecke)*.

Messbare Effekte sind kleinere Images und potentiell geringere Angriffsflächen: Das neue App-/Worker-/Scheduler-/Seeder-Image liegt bei 120 MB, das Nginx-Image bei lediglich 54 MB. In Kubernetes resultieren daraus schnellere Pulls, zügigere Rolling Updates, geringer Netzwerk- und Storage-Bedarf und ein deutlich reduzierter Scope für Security-Scans.

Zur Reproduzierbarkeit und Lieferkettensicherheit werden Basisimages und Werkzeuge strikt auf konkrete Versionen und Digests gepinnt (z. B. `node:24.9.0-alpine3.22`, `composer:2.8.12`, `php:8.4.13-fpm-alpine3.22`, `nginx:alpine3.22`, jeweils mit SHA-256-Digest). Dieses „immutable pinning“ verhindert unkontrollierte Tag-Drifts, ermöglicht

deterministische Builds und erzwingt bewusste, auditierbare Updates über Pull Requests [Doc25a].

Damit entsteht eine durchgängige Sicherheitskette: gehärtete Multistage-Images, kryptografisch abgesicherte Artefakte (vgl. 4.2.3), transparente Abhängigkeitslisten (vgl. 4.2.3) und strikte Durchsetzung im Cluster (vgl. 4.3.5).

In Summe verbindet diese Architektur Effizienz und Sicherheit: Die Container sind klein, schnell und ressourcenschonend. Builds sind deterministisch und reproduzierbar und die Lieferkette ist von der CI/CD-Pipeline bis zum Kubernetes-Cluster kryptografisch abgesichert. Damit wird die Containerisierung von einem reinen Betriebsmittel zu einem integralen Bestandteil der DevSecOps-Strategie, die Automatisierung, Sicherheit und Compliance gleichermaßen berücksichtigt.

4.3.2 Deploymentstrategien (Rolling Updates, Feature Flags)

Das Ziel von modernen Deployments in Kubernetes ist, Aktualisierungen ohne Unterbrechungen des Betriebs (*Zero-Downtime*) bereitzustellen und gleichzeitig die Risiken durch neue Versionen zu reduzieren [The25z]. Zwei grundlegende Strategien sind dabei das Rolling-Update-Verfahren und der Einsatz von Feature Flags [The25z, Fow17].

Rolling Updates in Kubernetes

Das Rolling-Update-Verfahren ist die Standardstrategie von Deployments in Kubernetes und ersetzt alte `Pod`-Instanzen schrittweise durch neue [The25z]. Während des Rollouts laufen alte und neue Versionen parallel, bis alle alten `Pod`s aktualisiert wurden [The25z]. Über Parameter wie `maxUnavailable` und `maxSurge` lässt sich das Verhältnis von stabiler Verfügbarkeit und Rollout-Geschwindigkeit konfigurieren [The25z].

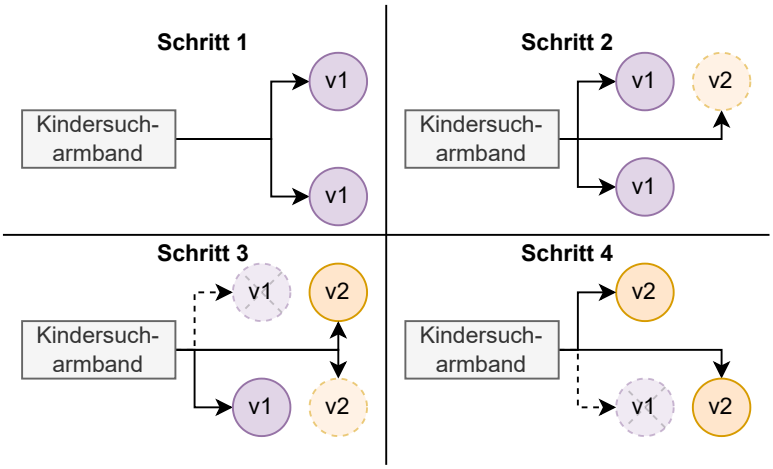

Abbildung 4.14: Visualisierte Darstellung eines Rolling-Update-Verfahrens. Die Pods *(bunte Kreise)* werden vom Deployment *„Kindersucharmband“ (graue Rechtecke)* schrittweise von *v1* auf *v2* aktualisiert [The25z].

Feature Flags in Laravel

Wie bereits im Zielprozess (siehe Abschnitt 4.2) beschrieben, ersetzt das Feature-Flag-Konzept die Testumgebung. Neue Funktionen können direkt in der Produktivumgebung ausgeliefert, aber zunächst nur intern aktiviert werden (*Dark Launch*) [Fow17]. Das bedeutet, dass Features in der Produktivumgebung von der Fachdomäne intensiv getestet werden können, ohne dass es Auswirkungen auf den Endnutzer hat. Sollten die Tests erfolgreich sein, kann die Fachdomäne jederzeit das Feature öffentlich aktivieren. Sollte es nach der Aktivierung zu Fehlern kommen, kann das Feature auch jederzeit wieder reversibel deaktiviert werden [Fow17].

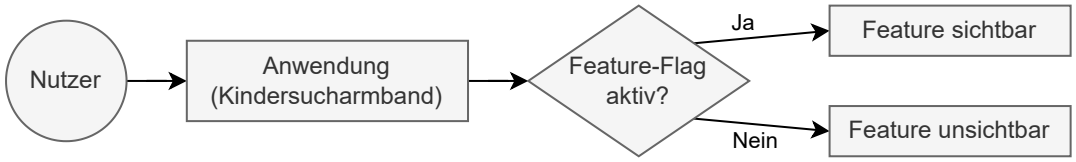

Abbildung 4.15: Dark Launch über Feature Flags

Dies reduziert die Komplexität von mehreren parallel laufenden Umgebungen (Nur Produktiv- statt Test- **und** Produktivumgebung), kann Feedbackzyklen verkürzen und

die Zusammenarbeit zwischen Entwicklung und Fachdomäne stärken [Fow17, HF10, FHK18].

Im Deploymentkontext dienen Feature Flags als ergänzender Mechanismus zu Rolling Updates. Während Kubernetes den technischen Austausch von Containerversionen steuert, ermöglichen Feature Flags eine kontrollierte Aktivierung neuer Funktionen innerhalb derselben Anwendungsversion. Auf diese Weise werden *Deployments entkoppelt von Releases*, was ein zentraler Baustein moderner DevOps-Praktiken ist [FHK18, CNC25].

Exemplarische Implementierung eines Feature Flags: Im Projekt *„Kindersucharmband“* kommt *Laravel Pennant* als leichtgewichtiges Feature-Flag-System zum Einsatz [Docd]. Ein praktisches Beispiel ist das Feature `SecureSearch`, das den Zugriff auf die Suchmaske für Armbandnummern mit einem zusätzlichen globalen Passwort absichert.

Der aktuelle Bedarf ergibt sich daraus, dass sich noch ältere Armband-Chargen im Umlauf befinden, deren Nummern sequentiell vergeben und somit leicht erratbar sind. Ohne zusätzliche Zugangsbeschränkung wäre es möglich, durch einfaches Hochzählen der Nummern auf fremde Datensätze zuzugreifen. Das Feature `SecureSearch` erzwingt daher zunächst die Eingabe eines globalen Passworts, bevor eine Suchanfrage ausgeführt werden kann. Damit ist sichergestellt, dass nur berechtigte Einsatzkräfte Zugriff auf die Suchfunktion erhalten.

```
namespace App\Features;

use App\Abstracts\Feature;
use InvalidArgumentException;

class SecureSearch extends Feature
{
    public function resolve(string $scope): bool
    {
        if ($scope === 'public') {
            return true;
        } else if ($scope === 'internal') {
            return true;
        } else {
            throw new InvalidArgumentException("Unknown scope: $scope");
        }
    }

    public function title(): string
    {
        return 'Globales Passwort vor Armbandsuche';
    }
```

```
    public function description(): string
    {
        return 'Schaltet das globale Passwort ein.
                Dieses muss zuerst eingegeben werden,
                bevor eine Kindersucharmbandnummer
                in die Suche eingegeben werden kann.';
    }
}
```

Listing 4.3: Feature-Flag Definition für `SecureSearch` in Laravel Pennant. In Ergänzung dazu wurden die Methoden `title()` und `description()` implementiert, um dem Domänenexperten einen erweiterten Kontext zu bieten. Die Visualisierung beider Werte erfolgt im Dashboard.

Die Verwaltung der Feature-Zustände erfolgt über ein eigenes Dashboard, das speziell für dieses Projekt entwickelt wurde.

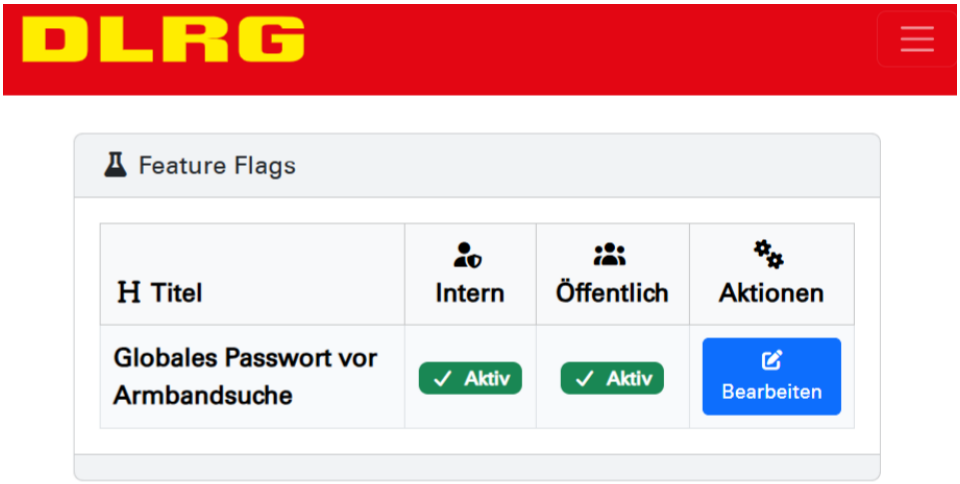

Abbildung 4.16: Anhand des beigefügten Screenshots wird das Dashboard veranschaulicht. Im vorliegenden Kontext wird das Feature *SecureSearch* präsentiert. Zusätzlich sind die Zustände für die öffentliche und interne Sichtbarkeit festgelegt. Domänexperten haben jederzeit die Möglichkeit, über das Dashboard das Feature reversibel zu aktivieren oder zu deaktivieren.

Damit können Features direkt in der Produktionsumgebung von den Stakeholdern aktiviert oder deaktiviert werden, ohne dass ein Entwickler eingreifen muss. Die Zustände werden in der Datenbank gespeichert und von Pennant zur Laufzeit ausgewertet [Docd].

So entsteht eine klare Trennung zwischen *technischem Deployment* (über CI/CD und GitOps) und *fachlicher Bereitstellung* (über das Dashboard).

Das Feature ist derzeit aktiv und schützt die Kindersuche während der Übergangsphase, in der noch alte, sequenzielle Armbandnummern im Umlauf sind. Sobald diese Bestände aufgebraucht und durch robuste Nummern ersetzt wurde, kann das Feature deaktiviert und aus dem Code entfernt werden. Damit wird ein temporäres Sicherheitsproblem elegant über Feature Flags abgefangen, ohne zusätzliche Deployments durchführen zu müssen.

Die Implementierung des Dashboards, kann dem Anhang im Abschnitt 7.4.4 entnommen werden.

Kritische Betrachtung: Die Integration von *Laravel Pennant* hat sich als leichtgewichtig und framework-nah erwiesen. Für kleine bis mittlere Projekte ist der Ansatz sehr praktikabel, da Feature Flags einfach im Code definiert und ohne externe Abhängigkeiten ausgewertet werden können. Langfristig erhöht jedoch die notwendige Eigenentwicklung von Verwaltungsoberflächen die Komplexität. Für größere Organisationen oder mehrere Anwendungen wäre eine zentralisierte Lösung, beispielsweise auf Basis von *OpenFeature* in Kombination mit *Unleash* oder *Flagd* sinnvoll, um einheitliche Governance, Observability und Richtlinienintegration zu gewährleisten [CNC25]. Im Kontext des Projekts stellt Pennant jedoch eine pragmatische und für den Umfang angemessene Lösung dar.

Ausblick – Grundlage für Traffic-Mirroring: Das eingeführte Konzept von Dark Launches legt zugleich den Grundstein für fortgeschrittene Deployment-Strategien wie *Traffic-Mirroring* [Kub25c]. Dabei werden eingehende Requests parallel an eine interne Schatteninstanz gesendet. Nur die Antwort der Primärinstanz geht an die Nutzer. Neue Versionen lassen sich so unter Realbedingungen beobachten, ohne die Benutzer zu beeinflussen [Ist]. Für eine belastbare Bewertung sind Telemetrie und aussagekräftige SLO-Metriken (z. B. Latenz, Fehlerraten, Ressourcenverbrauch) erforderlich. Zugleich muss die Schatteninstanz so konfiguriert sein, dass keine Seiteneffekte auftreten (z. B. keine externen *Schreiboperationen*). Progressive-Delivery-Werkzeuge im Kubernetes-Kontext unterstützen diesen Ansatz, etwa Argo Rollouts (Canary/Analysen) und Flagger (metrikenbasierte Freigaben) [Arg25, Flu25]. Die Kombination aus Rolling Updates, Feature Flags und künftigem Traffic-Mirroring bildet damit die Grundlage für eine vollständig *progressive Delivery Pipeline*, bei der technische und fachliche Änderungen kontrolliert, messbar und risikominimiert ausgerollt werden.

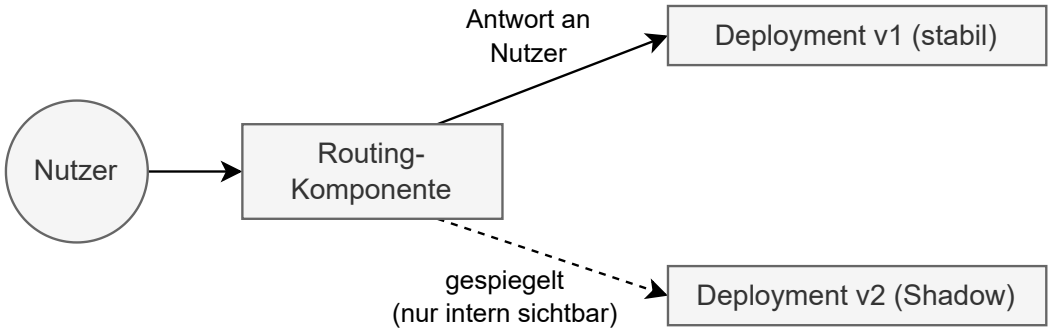

Abbildung 4.17: Traffic-Mirroring als zukünftige Erweiterung: parallele Verarbeitung ohne Nutzereinfluss

4.3.3 Geheimnisverwaltung mit HashiCorp Vault

Der Umgang mit vertraulichen Daten ist einer der sicherheitskritischsten Punkte im neuen System. Dazu gehören vor allem Datenbankzugangsdaten, Application Programming Interface (API)-Schlüssel oder der Laravel-App-Key. Würden solche Werte im Git-Repository landen oder in ConfigMaps beziehungsweise Container-Images eingebaut werden, wäre das ein erhebliches Risiko. Sie könnten leicht abgegriffen werden, und eine Rotation wäre nur schwer möglich.

Um dies zu verhindern, wird HashiCorp Vault genutzt. Vault dient hier als zentrale Plattform für die Verwaltung von Geheimnissen [OWA25]. Grundsätzlich unterscheidet Vault zwischen *statischen* und *dynamischen* Secrets [Has25]. Statische Werte, wie der Laravel-App-Key, werden verschlüsselt im Key-Value-Store abgelegt und bei Bedarf an die Anwendung übergeben [Hasc]. Dynamische Secrets dagegen entstehen erst, wenn sie angefordert werden [Hasa]. Sie laufen nach einer festgelegten Zeit automatisch ab [Hasd]. Praktisch bedeutet das: Selbst wenn ein Passwort kompromittiert würde, könnte es nur kurz genutzt werden.

Im Projekt *„Kindersucharmband“* wurde das bestehende Helm-Chart erweitert, um Vault einzubinden. Alle relevanten Workloads, also App, Worker, Scheduler, Migrator und Seeder nutzen `PodAnnotations`, mit denen der Vault-Agent aktiviert wird [Hase]. Dieser erzeugt beim Start eine `.env`-Datei und legt sie in den Container unter dem Pfad `/vault/secrets` ab. Auf diese Weise erhält jeder `Pod` genau die Konfiguration, die er braucht und zwar erst zur Laufzeit, nicht bereits beim Erstellen des Images. Eine beispielhafte `PodAnnotation`-Konfiguration ist in Listing 4.6 dokumentiert.

Besonders interessant ist die Kombination aus *Vault Agent Init* und *Vault Agent*. Zuerst authentifiziert sich der Init-Container gegenüber Vault, holt die ersten Werte und legt

sie in einem temporären `tmpfs`-Volume ab. So startet die Anwendung sofort mit allen nötigen Parametern. Danach bleibt der Sidecar-Container aktiv. Dieser erneuert Secrets regelmäßig und überschreibt die Dateien, sobald Werte ablaufen oder neu generiert werden. Gerade bei dynamischen Datenbankzugängen ist das unverzichtbar.

Authentifizierung: Für die Authentifizierung nutzt das System die *Kubernetes-Auth-Method* von Vault: Ein `ServiceAccount`-JSON Web Token (JWT) wird gegen die Kubernetes-API validiert [Hasb, The25t]. Eine Vault-Role bindet zulässige `ServiceAccount`s und `Namespace`s und verknüpft sie mit einer minimalen Policy [Hasb]. Das ausgehändigte Client-Token besitzt eine begrenzte Time-to-Live (TTL) und wird durch den Vault-Agent automatisch erneuert [Hasb]. Damit bleibt die Vertrauensbasis rotierbar und auf Kubernetes beschränkt. Die zugehörige Policy ist in Listing 4.4 dargestellt, die Role in Listing 4.5.

```
path "kv/data/kindersucharmband-laravel" { capabilities = ["read"] }
path "database/creds/kindersucharmband-laravel" { capabilities = ["read"] }
```

Listing 4.4: Vault-Policy für Kindersucharmband

```
vault write auth/kubernetes/role/kindersucharmband-laravel \
  bound_service_account_names="kindersucharmband-laravel" \
  bound_service_account_namespaces="kindersucharmband" \
  policies="kindersucharmband-laravel" ttl="24h"
```

Listing 4.5: Vault-Kubernetes-Role mit SA-Bindung

Helm-Integration (Agent-Inject): Die Einbindung erfolgt über `PodAnnotations` des Vault-Agents. Die `.env` wird zur Laufzeit aus dem `kv`-Pfad gerendert und im RAM abgelegt (`emptyDir.medium: Memory`) [The25w]. Beispielhaft zeigt Listing 4.6 einen Values-Ausschnitt.

```
podAnnotations:
  vault.hashicorp.com/agent-inject: "true"
  vault.hashicorp.com/role: "kindersucharmband-laravel"
  vault.hashicorp.com/agent-inject-secret-.env: >
    "kv/data/kindersucharmband-laravel"
  vault.hashicorp.com/agent-inject-template-.env: |
    {{- with secret "kv/data/kindersucharmband-laravel" -}}
    APP_KEY={{ .Data.data.APP_KEY }}
    DB_DATABASE={{ .Data.data.DB_DATABASE }}
    GLOBAL_PASSWORD_HASH={{ .Data.data.GLOBAL_PASSWORD_HASH }}
    SMS77_API_KEY={{ .Data.data.SMS77_API_KEY }}
    {{- end }}
```

Listing 4.6: Helm Values: Vault-Agent-Inject (Ausschnitt)

Nicht alle Werte müssen jedoch über Vault kommen. Parameter wie `LOG_LEVEL` oder `APP_URL` gelten nicht als vertraulich. Sie bleiben im Kubernetes-State, sind dort versioniert und werden über das GitOps-Repository ins Cluster gebracht. Dadurch lassen sich Änderungen im Nachhinein prüfen, und gleichzeitig ist sichergestellt, dass die `Pod`s genau mit den im Repository definierten Parametern laufen.

Ein weiterer Vorteil zeigt sich bei der Art der Speicherung: Anstatt im `etcd`-Cluster von Kubernetes abgelegt zu werden, schreibt der Vault-Agent die Secrets direkt in ein temporäres `tmpfs`-Volume [The25w]. Dadurch liegen die Daten ausschließlich im Arbeitsspeicher des `Pod`s und es erfolgt keine persistente Ablage im Cluster-Storage oder in Backups. Startet der `Pod` neu, werden frische Werte geladen. Das Risiko einer unbeabsichtigten Offenlegung sensibler Werte wird dadurch deutlich reduziert.

Integration in Laravel

Eine Besonderheit ergab sich in Kombination mit Laravel. Der *Vault Agent* legt standardmäßig Verzeichnisse wie `/vault/secrets` an, Laravel hingegen erwartet eine einzelne `.env`-Datei im Projektverzeichnis (z. B. `/var/www/html`) [Docc]. Deshalb wurde der Bootstrap-Prozess so erweitert, dass beim Start geprüft wird, ob im Vault-Verzeichnis eine `.env` vorhanden ist. Falls ja, wird sie geladen. Für Entwickler, die lokal mit *Laravel Sail* (siehe Abschnitt 4.2.2) arbeiten, ändert sich nichts. In der Produktion greift die Anwendung dagegen automatisch auf Vault zurück.

```
if (file_exists('/vault/secrets/.env')) {
    // Lies die Env-Datei direkt aus /vault/secrets
    Dotenv::createImmutable('/vault/secrets')->load();
}
```

Listing 4.7: Auszug zur Anpassung des Bootstrap-Prozesses um die abgelegte Vault-`.env` in Laravel zu lesen. Der gesamte Programmcode ist im Listing 7.4.3 dokumentiert.

Dynamische Datenbank-Secrets

Einen besonderen Stellenwert haben die dynamischen Datenbankzugänge. Vault legt dafür temporäre Benutzer an, die nur eingeschränkte Rechte besitzen. Diese Zugänge sind zeitlich befristet und werden nach Ablauf automatisch gesperrt. Im Projekt wurde hierfür eine begrenzte Lebensdauer der Anmeldeinformationen (TTL) konfiguriert. Nach

Ablauf dieser Zeitspanne, beispielsweise 30 Minuten bis 1 Stunde, werden die Zugangsdaten automatisch widerrufen und bei Bedarf neu generiert. Damit wird das Prinzip *Least Privilege* umgesetzt. Selbst wenn ein Angreifer an die Daten käme, wären sie nur sehr kurz nutzbar.

Im Projekt *„Kindersucharmband“* teilen sich derzeit alle Workloads denselben dynamischen Benutzer. Das hält die Komplexität niedrig und vereinfacht den Betrieb. Möglich wäre aber auch eine feinere Trennung. Zum Beispiel könnte der Migrator mehr Rechte bekommen, während die App-Pods nur Lese- und Schreibrechte hätten. Technisch ist das problemlos machbar, doch im Projekt wurde darauf verzichtet weil der zusätzliche Sicherheitsgewinn nicht im Verhältnis zum Mehraufwand stand.

```
vault write database/roles/kindersucharmband-laravel \
  db_name="kindersucharmband-laravel" \
  creation_statements="CREATE USER '{{name}}'@'%' IDENTIFIED BY '{{password}}';
    GRANT SELECT, INSERT, UPDATE,
    ALTER, CREATE, DELETE, DROP, REFERENCES ON *.* TO '{{name}}'@'%';" \
  default_ttl="1h" max_ttl="24h"
```

Listing 4.8: Vault: MySQL-Role mit eingeschränkten Rechten

Damit erfüllt die Lösung sowohl die Anforderungen an *Security by Design* als auch die Erwartungen moderner DevOps- und GitOps-Prozesse.

Einordnung und Alternativen

Die Entscheidung für HashiCorp Vault fiel nach einem Vergleich mehrerer Alternativen, die für den sicheren Umgang mit vertraulichen Daten in Kubernetes in Frage kamen.

Native Kubernetes-Secrets: Kubernetes selbst bringt bereits ein einfaches System für Secrets mit [The25g]. Die Secrets werden Base64-kodiert und im etcd-Cluster abgelegt und sind damit für Administratoren mit ausreichenden Rechten einsehbar. Außerdem gibt es keine eingebaute Möglichkeit, Zugangsdaten automatisch zu erneuern oder zeitlich zu begrenzen. Für Umgebungen mit hohen Sicherheitsanforderungen, etwa nach dem Zero-Trust-Prinzip, ist das deshalb keine ideale Lösung.

Sealed Secrets und SOPS: Etwas anders arbeiten Tools wie Bitnami Sealed Secrets oder Mozilla Secrets Operations (SOPS) [Bit25, Moz25]. Hier werden die Secrets schon im Git-Repository verschlüsselt gespeichert und erst im Cluster wieder entschlüsselt. Dadurch bleibt die gesamte Konfiguration nachvollziehbar und versioniert, was besonders im GitOps-Kontext hilfreich ist. Die Nachteile liegen vor allem in der fehlenden

Dynamik: Werte müssen manuell erneuert werden, es gibt keine automatische Rotation, und die Sicherheit hängt stark von den verwendeten Verschlüsselungsmechanismen ab. Eine weitere Herausforderung ist es, dass obwohl die Secrets verschlüsselt im GitOps-Repository gespeichert werden, entsteht beim Entschlüsseln im Cluster erneut ein natives Secret. Dieses wird im etcd-Cluster abgelegt und kann dort (trotz Encryption-at-rest) ohne Lease/Rotation typischerweise von Cluster-Administratoren im Klartext ausgelesen werden.

Cloud-basierte Secret-Manager: Auch viele Cloud-Anbieter stellen eigene Systeme bereit, zum Beispiel den AWS Secrets Manager, den Google Secret Manager oder den Azure Key Vault [Ama25, Goo25, Mic25]. Sie bieten eine gute Integration in die jeweilige Cloud-Umgebung und unterstützen teils auch automatische Schlüsselrotation. Für On-Premise- oder hybride Szenarien wie im Projekt *„Kindersucharmband“* sind sie allerdings nur bedingt geeignet. Sie benötigen in der Regel eine ständige Verbindung zur Cloud und geben einen Teil der Kontrolle über die Daten an den Anbieter ab, was nicht immer gewünscht ist.

Bewertung: HashiCorp Vault vereint viele Vorteile dieser Ansätze, ohne ihre Schwächen zu übernehmen. Es kann dynamische Secrets erzeugen, nutzt ein fein abgestuftes Rollen- und Rechtemodell und legt vertrauliche Daten ausschließlich im Arbeitsspeicher der Pods ab. Ein weiterer Vorteil ist die Integration über Init- und Sidecar-Container: Anwendungen müssen nicht verändert werden, um Vault zu nutzen. So entsteht ein System, das sicher, automatisierbar und zugleich GitOps-kompatibel ist und damit gut zu Umgebungen passt, in denen sowohl Laufzeitsicherheit als auch Nachvollziehbarkeit wichtig sind.

4.3.4 Observability und kontinuierliche Laufzeitüberwachung

Ein wesentliches Ziel der neuen Kubernetes-Architektur war es, die Anwendung nicht nur skalierbar, sondern auch quantifizierbar und beobachtbar zu machen. Observability beschreibt dabei die Möglichkeit, den Gesundheitszustand eines Systems aus Telemetriedaten (z.B. Metriken oder Logs) abzuleiten [Clo, Goob].

Im Sinne des DevOps-Prinzips des **Continuous Feedback** soll der Betrieb der Anwendung permanent Rückmeldungen über Stabilität, Performance und Ressourcenauslastung liefern [FHK18]. Diese Rückmeldungen fließen wiederum in die Weiterentwicklung ein und schließen damit den Kreislauf [FHK18].

Umsetzung und Integration in Laravel

Da Laravel keine native Unterstützung für Prometheus bietet, wurde für das *„Kindersucharmband“* das Open-Source-Paket `spatie/laravel-prometheus` hinzugefügt [Spa25]. Dieses erweitert Laravel um einen dedizierten `PrometheusServiceProvider`, über den anwendungsspezifische Metriken als sogenannte *Gauges, Counters* oder *Histograms* definiert werden können [Spa25]. Abhängig von der Konfiguration stehen die genannten Metriken unter `/metrics` oder `/prometheus` zur Verfügung [Spa25]. Das Helm-Chart (siehe Abschnitt 4.2.5) wurde so erweitert, dass über die Parameter in der `values.yaml` gesteuert werden kann, ob die Custom Resource Definition (CRD) `ServiceMonitor` erzeugt wird (siehe Listing 7.3.11) [The25i, Theb]. Wenn dieser aktiviert ist, erkennt Prometheus die Anwendung automatisch und beginnt in einem vorher festgelegten Intervall (z.B. 30 Sekunden), die bereitgestellten Metriken abzurufen. Damit lässt sich die Observability-Integration deklarativ aktivieren oder deaktivieren und ist somit ein Ansatz der sich konsistent in das bestehende GitOps-Deployment einfügt und reproduzierbare Konfigurationen gewährleistet.

Die exponierten Metriken umfassen sowohl systemische als auch anwendungsspezifische Kennzahlen. Die folgenden sind von besonderer Relevanz:

- die Anzahl der Jobs in der Queue und deren Zustand,
- die Erreichbarkeit und das verfügbare Guthaben der SMS-Schnittstelle,
- die Speicherauslastung des PHP-Prozesses,
- sowie allgemeine Systemkennzahlen wie die Load Average.

Darüber hinaus wird eine Metrik über Build-Informationen bereitgestellt, die PHP- und Laravel-Version sowie die Umgebung enthält. Dies ermöglicht eine eindeutige Zuordnung von Betriebszuständen zu konkreten Softwareständen. Dies ist ein bedeutender Aspekt für eine potenzielle Fehleranalyse.

Ein vollständiger Auszug der erzeugten Metriken ist dem Anhang im Listing 7.6.1 zu entnehmen.

Auf Grundlage dieser Metriken wurden Prometheus Regeln definiert, die typische Betriebszustände und Fehlerbilder automatisch erkennen. Dazu zählen beispielsweise:

- eine **hohe Queue-Auslastung** (Hinweis auf unzureichende Worker-Skalierung),
- **hängende Jobs**, die über einen längeren Zeitraum nicht abgearbeitetn werden,
- **nicht erreichbarer SMS-Gateway**,
- sowie **hohe CPU- oder Speicherauslastung** einzelner Pods.

Diese Regeln sind als `PrometheusRule` (CRD) im Kubernetes-Cluster hinterlegt und bilden die operativen SLO des Systems (siehe Listing 7.6.2) [The25i, Thea].

Die zugehörigen Alerts werden über den *Alertmanager* verarbeitet und in *Grafana* visualisiert [Graa]. Auf diese Weise wird eine automatisierte, zentralisierte Überwachung in Echtzeit ohne manuellen Eingriff ermöglicht.

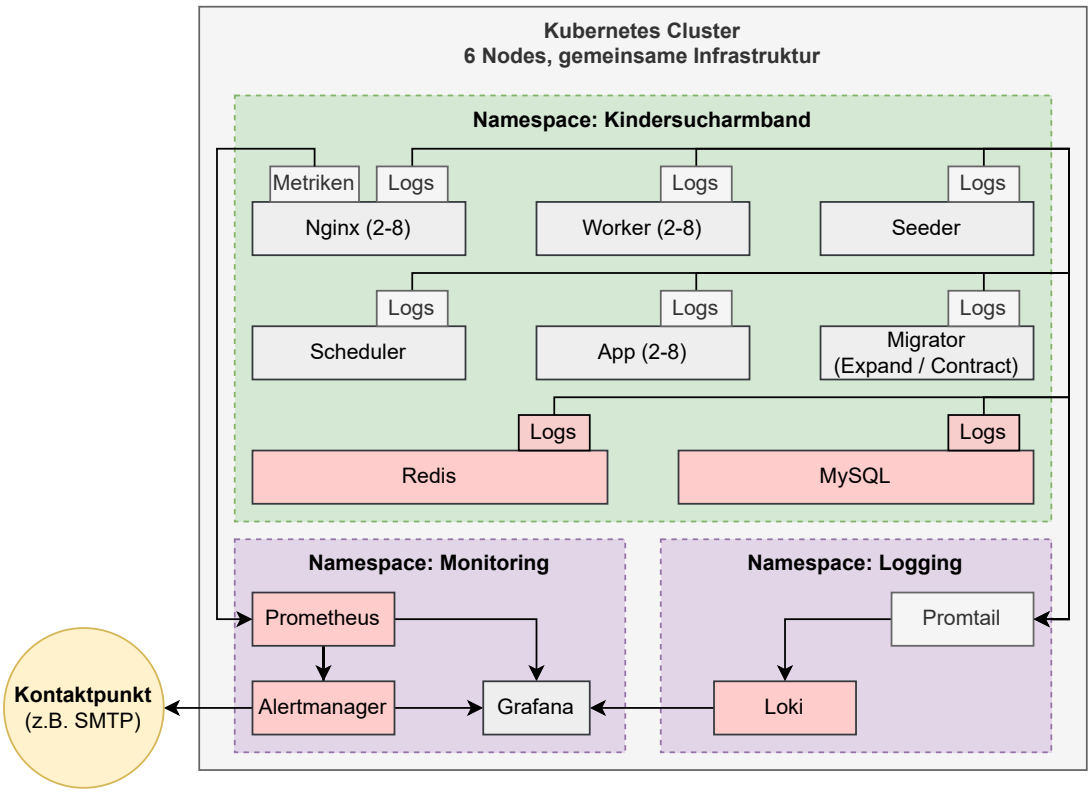

Abbildung 4.18: Darstellung der Interaktion um Telemetriedaten zentralisiert in Grafana zu visualisieren. Die Logs der einzelnen Pods im `Namespace` *„Kindersucharmband“* werden von *Promtail* gesammelt und in *Loki* aggregiert und gespeichert [Grad, Grab]. Metrik-Endpunkte werden von *Prometheus* aufgerufen und an *Alertmanager* sowie Grafana weitergeleitet. *Grafana* hat somit *Prometheus*, *Alertmanager* und *Loki* als Datenquelle [Graa, Grac].

Im Vergleich zur Docker-Architektur, bei der Systemzustände nur manuell geprüft werden konnten, wie zum Beispiel Log-Analysen, Jobs in der Queue stauen oder API-Erreichbarkeit, stellt dies einen deutlichen Fortschritt dar.

Perspektive und zukünftige Erweiterung

Zum aktuellen Zeitpunkt werden primär anwendungsspezifische Metriken erfasst. Eine Messung der tatsächlichen Anwendungslatenz (z.B. *p95* oder *p99*-Quantil) ist nur mit zusätzlichen Tools möglich. Diese Erweiterung ist für einen späteren Ausbauschritt vorgesehen. Das hätte den Vorteil, dass nicht nur Infrastrukturzustände, sondern auch Nutzererfahrungen quantitativ eruiert werden.

Mit dieser Architektur wurde der Grundstein für ein vollwertiges Continuous-Feedback-System gelegt [FHK18]. Betriebsdaten werden kontinuierlich gesammelt, bewertet und fließen zurück in Entscheidungen um das System langfristig zu optimieren.

4.3.5 Laufzeitüberwachung und Compliance-Prüfung

Während die zuvor beschriebene Observability-Ebene technische und betriebliche Zustände sichtbar macht, adressiert die Laufzeitüberwachung in diesem Kontext die Integrität der ausgeführten Container-Artefakte und die Einhaltung sicherheitsrelevanter Richtlinien [Kub23]. Unter Compliance-Prüfung wird im Folgenden die automatisierte Validierung kryptografischer Vertrauenskriterien verstanden, die sicherstellt, dass ausschließlich überprüfte, signierte und attestierte Software-Artefakte im Produktionssystem ausgeführt werden [Kub23]. Dieser Ansatz ist ein zentraler Bestandteil moderner Software-Supply-Chain-Security-Konzepte und wird unter anderem im NIST Secure Software Development Framework (SSDF) sowie der Cloud Native Computing Foundation (CNCF) Security Landscape eines DevSecOps-getriebenen Entwicklungsprozesses beschrieben [Nat23, MLK+23].

Das Ziel dieser Architektur ist es, die Vertrauenskette, die durch die Signierung und Attestierung im Build-Prozess entsteht, bis in die Laufzeitumgebung zu verlängern. Dadurch prüft das System Container-Images automatisiert auf Integrität.

Zur praktischen Umsetzung wird der *Sigstore Policy Controller* als Admission Controller im Kubernetes-Cluster eingesetzt [Sig25a, Kub23]. Dieser Controller überprüft bei jeder Bereitstellung, ob die referenzierten Container-Images die definierten Sicherheitsanforderungen erfüllen [Sig25a]. Darunter fällt eine gültige Signatur, eine überprüfbare Herkunft über eine OpenID Connect (OIDC)-Identität (in diesem Fall URL vom GitHub-Action-Workflow) sowie das Vorhandensein von Attestierungen wie einer Software Bill of Materials (SBOM) [Sig25a]. Die entsprechenden Richtlinien werden als `ClusterImagePolicies` zentral hinterlegt und sorgen dafür, dass nur `Deployments` akzeptiert werden, die diese Anforderungen erfüllen (siehe Listing 7.3.13) [Sig25a]. Zusätzlich lässt dieses Vorgehen Sicherheitsrichtlinien deklarativ (Policy as Code) in der Infrastruktur beschreiben, wodurch Compliance-Vorgaben reproduzierbar und versionierbar sind.

Die Signierung der Container-Images erfolgt automatisiert in der CI/CD-Pipeline über die sogenannte *keyless signing*-Funktion von *Cosign* [Sig25b].

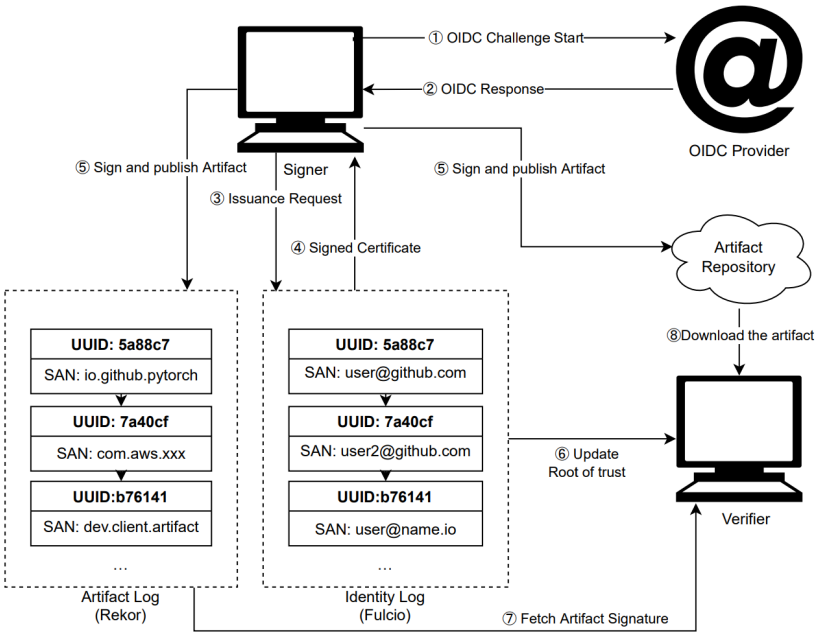

Abbildung 4.19: Ablauf der keyless Signierung und Zertifikatsausstellung mit Sigstore (nach [NMTA22]).

Dabei wird über die OIDC-Identität des Workflows bei *Fulcio* ein kurzlebiges Zertifikat bezogen, mit dem das Image signiert wird. Die Signatur und die zugehörige Attestierung werden anschließend in dem öffentlichen Transparenzlog *Rekor* gespeichert [NMTA22].

Hierdurch entsteht eine revisionssichere Nachweiskette, die eine eindeutige Zuordnung von Signatur, Repository, Workflow und Zeitpunkt gewährleistet.

Zur weiteren Absicherung werden sicherheitsrelevante Richtlinien zusätzlich auf `Namespace`-Ebene durchgesetzt. In der `Namespace`-Ressource der Anwendung *„Kindersucharmband“* ist der Modus `restricted` aktiviert [The25p]. Dadurch wird unter anderem der Betrieb privilegierter Container, die Nutzung von Host-Ressourcen sowie die Ausführung von Prozessen als Root-Nutzer unterbunden (siehe Listing 7.3.12) [The25p].

Diese Kombination aus kryptografischer Signaturprüfung und Laufzeit-Policy-Enforcement stellt sicher, dass ausschließlich verifizierte Container-Artefakte in der Produktionsumgebung ausgeführt werden. Damit wird die Vertrauenskette des Build-Prozesses konsequent bis in den Betrieb verlängert.

4.4 Zusammenfassung der Transformation und Vorher/Nachher-Überblick

Dieser Abschnitt fasst die vorgenommenen Struktur- und Prozessänderungen zusammen und stellt einen knappen Vorher/Nachher-Überblick bereit.

Strukturelle Änderungen

Tabelle 4.2: Änderungskarte – Design- und Prozessunterschiede

Aspekt	**Vor der Transformation**	**Nach der Transformation**
Artefakterzeugung	Manuelle/teilautom. Builds	Multistage-Builds, deterministische Artefakte (Digest-Pinning)
Auslieferung	Server-/Skript-basiert, manuelle Schritte	GitOps (HelmRelease & Flux), PR-getrieben
Nebenprozesse	Cron/Worker im App-Container vereint	Eigenständige Workloads (CronJob/Worker/Jobs)
DB-Änderungen	Manuelle Migration/Seeding nach Deploy	Orchestriert via Jobs & Hooks; Expand/Contract-Verfahren
Secrets	.env im Repo/Image/Server	Vault-Agent (Init/Sidecar), Laufzeit-Injection
Netz & Hardening	Keine NetPols, Root, RW-RootFS	Default-deny, Non-Root, RO-RootFS, Seccomp
Observability	Container-lokale Logs, kein zentrales Monitoring	Prometheus/Loki/Grafana, ServiceMonitor
Release-Steuerung	Feature = Deployment	Feature-Flags (entkoppelte Freischaltung)
Rollback	Ad-hoc/Manuell	Git-Revert/Version-Pinning im GitOps-Flow

Vorher/Nachher – kompakter Überblick

Tabelle 4.3: Vorher/Nachher-Überblick ausgewählter Kriterien

Kriterium	**Vorher (Beobachtung)**	**Nachher (Beobachtung)**
Bereitstellung	Container-Neustarts mit kurzer Nichtverfügbarkeit	RollingUpdate mit Readiness/Startup-Probes
Datenbankprozesse	Migration/Seeding manuell nach Deploy	Automatisierte Jobs (Hooks)
Queue/Scheduler	Supervisord & Cron im App-Container	Worker-Deployment & CronJob
Secrets-Umgang	Konfiguration zur Build-/Server-Zeit	Laufzeit-Injection (Vault)
Netz/Isolierung	Breite Ost–West-Kommunikation	Fein granulierte Network-Policies
Artefaktgröße	Runtime-Images sehr groß	Schlanke Runtime-Images (Multistage)
Transparenz	Logs in Containern, keine Metriken	Zentrales Logging/Metriken/Alerts
Releasefluss	Mehrere manuelle Schritte/-Serverzugriffe	PR→Merge→GitOps-Rollout (deklarativ)

5 Evaluation

5.1 Evaluationsprinzip: Anforderungen als Hypothesen

Die Anforderungen R1–R19 bilden die normativen Zielkriterien dieser Arbeit. Für die Evaluation werden sie als prüfbare Hypothesen H_{R1-19} operationalisiert. Jede Hypothese benennt Messgröße, Datengrundlage und eine Entscheidungsregel. Die Auswertung erfolgt in den jeweiligen Unterabschnitten 5.2.1–5.2.19.

5.2 Auswertung der Anforderungen

5.2.1 R1: Reproduzierbarer Build (H_{R1})

Hypothese: Builds aus identischem Commit erzeugen denselben `SHA256`-Digest des Runtime-Images.

Messung: Jeweils zehn Builds ($n = 10$) via CI/CD-Pipeline desselben Commits (`513167b3b49f`) für die Artefakte *App (PHP-FPM)* und *Nginx*. Neben dem Image-Digest wurde ein *Inhalts-Hash* des Zielverzeichnisses im Image gebildet (`find . -type f | sort | xargs sha256sum | sha256sum`), der ausschließlich Dateiinhalte berücksichtigt.

Ergebnis: Die Image-Digests unterscheiden sich in allen Läufen, aber die Inhalts-Hashes sind konstant:
App → `951c6592c890172e77e0a6e39a724204f4ca7e1352da17598e9d3112663324aa`,
Nginx → `5fa20fd160a62f0631d3e8be3356045aaa74baced764f9432a8f352745caa53a`.

Damit sind die Runtime-Inhalte beider Artefakte deterministisch. Die Digest-Differenzen sind auf Build-/COPY-Metadaten (z. B. `mtime/Owner/Mode`) zurückzuführen.

Bewertung: Teilweise erfüllt. Strikte Byte-Reproduzierbarkeit des Image-Digests wurde nicht stabil erreicht. Die funktionale Reproduzierbarkeit der Inhalte ist jedoch gegeben.

5.2.2 R2: Signatur & Provenance (H_{R2})

Hypothese: 100 % der ausgelieferten Container-Images sind kryptografisch signiert. Attestations (z. B. SBOM) werden erzeugt und sind verifizierbar.

Messung: Verifikation der Image-Signaturen und Attestations mit `cosign` gegen die erwartete OIDC-Identität des GitHub-Actions-Workflows (`--certificate-identity-regexp`, `--certificate-oidc-issuer`). Transparenzlog-Nachweis via `cosign triangulate`. Für SBOM: `cosign verify-attestation` mit `--type spdx`.

Nachweise (Auszug): *App (PHP-FPM):*

```
cosign triangulate \
  ghcr.io/dlrgev-bgst/kindersucharmband-app:513167b3b49f...23680059
# -> ghcr.io/.../kindersucharmband-app:sha256-929465...dd9393d.sig

cosign verify \
  ghcr.io/dlrgev-bgst/kindersucharmband-app:513167b3b49f...23680059 \
  --certificate-identity-regexp \
    "^https://github.com/DLRGeV-BGSt/(...)/ci-build-image.yaml@refs/.*$" \
  --certificate-oidc-issuer "https://token.actions.githubusercontent.com"

cosign verify-attestation \
  ghcr.io/dlrgev-bgst/kindersucharmband-app:513167b3b49f...23680059 \
  --certificate-identity-regexp \
    "^https://github.com/DLRGeV-BGSt/(...)/ci-build-image.yaml@refs/.*$" \
  --certificate-oidc-issuer "https://token.actions.githubusercontent.com" \
  --type spdx
```

Nginx:

```
cosign verify \
  ghcr.io/dlrgev-bgst/kindersucharmband-nginx:513167b3b49f...23680059 \
  --certificate-identity-regexp \
    "^https://github.com/DLRGeV-BGSt/(...)/ci-build-image.yaml@refs/.*$" \
  --certificate-oidc-issuer "https://token.actions.githubusercontent.com"

cosign triangulate \
  ghcr.io/dlrgev-bgst/kindersucharmband-nginx:513167b3b49f...23680059
# -> ghcr.io/.../kindersucharmband-nginx:sha256-e2d53f...1b659d.sig
```

Policy-Enforcement (Kurztest): Ein Test-`Pod` mit `busybox:1.36` im Namespace `kindersucharmband` wurde absichtlich ohne Signatur deployt und vom Admission Controller abgewiesen (*admission denied: image not trusted*). Ein `Pod` mit dem signierten

App-Image (`ghcr.io/dlrgev-bgst/kindersucharmband-app:513...059`) wurde akzeptiert. Damit ist belegt: Im Ziel-Namespace können ausschließlich signierte Images ausgeführt werden.

Ergebnis: Für *App* und *Nginx* wurden die Signaturen erfolgreich gegen die erwartete OIDC-Identität verifiziert. Die Existenz der Signatur-Einträge im Rekor-Transparenzlog ist belegt. Für das App-Image wurde eine SBOM-Attestation (SPDX) erfolgreich verifiziert. Der Negativtest im Cluster wurde durch die Admission-Policy blockiert.

Bewertung: Erfüllt. Sämtliche ausgelieferten Images sind kryptografisch signiert. Attestations (SBOM) sind verifizierbar. Die Bindung an den konkreten CI-Workflow (OIDC) und die Policy-Durchsetzung im Cluster sichern die Lieferkettenintegrität Ende-zu-Ende.

5.2.3 R3: Auditierbarkeit (H_{R3})

Hypothese: Jede Build-/Deploy-Aktion erzeugt eine durchgängige Auditkette (Who/What/When/Commit/Digest) vom Commit bis zum laufenden Pod.

Messung: Ende-zu-Ende-Korrelation eines Releases (v1.40.19) über *Git* (Commit, Tag), *CI* (Run-ID, Zeitpunkt), *Registry* (Image-Digest), *GitOps/Release-Artefakte* (SBOMs) sowie *Cluster-Runtime* (tatsächlich laufender Digest).

Evidenz (Auszug):

```
# Commit -> Autor/Zeit/Tag
git show 513167b3b49f4c6aeec82c14b4db965623680059
# commit ... (HEAD -> main, tag: v1.40.19, ...)
# Author: Superman9666 ...
# Date:   Tue Oct 14 08:36:07 2025 +0200

# Release -> veroeffentlicht, SBOM-Artefakte
gh release view v1.40.19
# ... released ... (2025-10-13)
# Assets:
#  sbom-app.spdx.json    sha256:6641eca1...
#  sbom-nginx.spdx.json  sha256:2e58957f...

# CI-Run (Who/When/Commit)
gh run list --workflow "[CI] Build & Sign Docker Image" \
   --json databaseId,createdAt,headSha --limit 1
# { "databaseId": 18487769733, "createdAt": "2025-10-14T06:36:21Z",
#   "headSha": "513167b3b49f4c6aeec82c14b4db965623680059" }

# Registry: Digest des App-Images zum Commit-Tag
```

```
skopeo inspect \
  docker://ghcr.io/dlrgev-bgst/kindersucharmband-app:513167b3b...23680059 \
  | jq -r '.Digest'
# sha256:929465298de5bfd417d1d81887c810e33d1b808316c54c06e5224ea13dd9393d

skopeo inspect \
  docker://ghcr.io/dlrgev-bgst/kindersucharmband-nginx:513167b3b...23680059 \
  | jq -r '.Digest'
# sha256:e2d53f152da0487a2555ffa60272b3d747503c3e501759f582a9ff575f1b659d

# Runtime: tatsaechlich laufende Digests im Cluster
kubectl -n kindersucharmband get pods -l app=kindersucharmband -o json \
  | jq -r '.items[].status.containerStatuses[].imageID'
# ghcr.io/.../kindersucharmband-app@sha256:929465298de...24ea13dd9393d
# ghcr.io/.../kindersucharmband-nginx@sha256:e2d53f152da0487...f575f1b659d
```

Ergebnis: Commit `513167b3...` → Release `v1.40.19` (mit SBOM-Artefakten) → CI-Run `#18487769733` (2025-10-14T06:36:21Z, `headSha=513167b3...`) → Registry-Digest `sha256:929465...` für das App-Image → laufende Pods mit `imageID=...@sha256:929465...` (App) bzw. `...@sha256:e2d53f...` (Nginx). Alle Referenzen sind konsistent.

Bewertung: Erfüllt. Die Auditkette ist lückenlos und revisionssicher: Commit/Tag, CI-Run, signiertes Image (Digest), Release-Artefakte (SBOM) und der im Cluster laufende Digest korrespondieren eindeutig. Verweise auf Signatur- und Attestationsprüfung siehe Abschnitt 5.2.3.

5.2.4 R4: GitOps-Deployment (H_{R4})

Hypothese: Deployments erfolgen ausschließlich deklarativ über versionierte Manifeste und einen GitOps-Controller. Änderungen laufen über Pull Requests (PR). Live-Abweichungen (*Drift*) werden automatisch erkannt und auf den Git-Stand zurückgesetzt (weiche Durchsetzung).

Messung: *Negativtest* mit manueller Live-Änderung (Scale) und Beobachtung der automatischen Drift-Korrektur auf den im Repository deklarierten Wert.

Evidenz (Auszug): *Negativtest (Drift-Korrektur, realer Lauf):*

```
# Startzustand:
kubectl -n kindersucharmband get deploy kindersucharmband-laravel-app \
  -o jsonpath='{.spec.replicas}{"\n"}'
# -> 2 (Replikate)

# Manuelle Live-Aenderung (Drift):
```

```
kubectl -n kindersucharmband scale deploy kindersucharmband-laravel-app \
  --replicas=3
# -> deployment.apps/kindersucharmband-laravel-app scaled

# Sofortiger Effekt:
kubectl -n kindersucharmband get deploy kindersucharmband-laravel-app \
  -o jsonpath='{.spec.replicas}{"\n"}'
# -> 3 (Replikate)

# Nach 20s: Flux reconciled -> Ruecksetzung auf Git-Stand
kubectl -n kindersucharmband get deploy kindersucharmband-laravel-app \
  -o jsonpath='{.spec.replicas}{"\n"}'
# -> 2 (Replikate)
```

Ergebnis: Die manuelle Skalierung auf `3` Replikas griff nur kurzfristig. Der GitOps-Controller erkannte den Drift und stellte den deklarierten Zustand `2` automatisch wieder her.

Bewertung: Erfüllt Git ist die *Single Source of Truth*. Der effektive Betriebszustand folgt dem Repository. Manuelle `kubectl`-Änderungen sind nicht persistent und werden automatisiert korrigiert.

5.2.5 R5: Zero-Downtime-Rollouts (H_{R5})

Hypothese: Rolling-Updates verursachen keine Nutzerunterbrechung. Das 95%-Quantil der Downtime pro Release beträgt 0 s.

Messung: Während geplanter Rollouts wird im *Sekundentakt* per `curl` der Ingress-Endpunkt (`/healthz`) abgefragt (1 Hz, Timeout 1 s). Eine Sekunde gilt als *down*, wenn in diesem Intervall kein erfolgreicher HTTP-Check ($2xx/3xx$) mit Gesamtzeit $\leq$ 1 s vorliegt. Auswertung: Downtime-Sekunden je Rollout sowie p95 der Downtime über 3 Rollouts.

Evidenz (Auszug):

```
# Beispiel-Rollout:
kubectl -n kindersucharmband rollout \
  restart deploy kindersucharmband-laravel-app

kubectl -n kindersucharmband port-forward \
  service/kindersucharmband-laravel-nginx 8080:80

# 1 Hz Hauptmessung (Sekundenaufloesung, Timeout 1s):
while true; do
```

```
  ts=$(date -Is)
  curl -sS --max-time 1 -o /dev/null \
       -w "$ts code=%{http_code} rt=%{time_total}\n" \
       http://localhost:8080/healthz
  sleep 1
done
```

Ergebnis: In allen betrachteten Rollouts wurden ausschließlich 200-Antworten beobachtet. Die Downtime-Sekunden lagen durchgängig bei 0. Die Erfolgsrate im Rollout-Fenster betrug $\approx 100\%$, die beobachtete Latenz blieb im Durchschnitt stabil. Über 3 Releases ergibt sich damit ein p95 der Downtime von 0 s.

Bewertung: Erfüllt. Die Rolling-Updates verliefen ohne messbare Unterbrechung. Readiness/Probes, `maxUnavailable: 0` und Pod Disruption Budgets (PDB) hielten Verkehr von nicht-bereiten Pods fern. Der Service antwortete durchgehend erfolgreich.

5.2.6 R6: Integriertes Secret Management (H_{R6})

Hypothese: Es befinden sich *keine Secrets* im Git-Repository (Arbeitsbaum *und* Historie) sowie in den finalen Runtime-Images. Secrets werden ausschließlich zur Laufzeit aus dem zentralen Secret-Management bereitgestellt.

Messung: (1) *Gitleaks* prüft Arbeitsbaum und komplette Git-Historie auf Secret-Muster (API-Keys, Tokens, private Schlüssel etc.) [Zac]. (2) *Trivy* prüft die finalen Runtime-Images ausschließlich mit dem Secret-Scanner (`--scanners secret`) [Tri].

Gate: Exit-Code = 0 und Findings = 0 in allen Prüfungen.

Evidenz (Auszug):

```
# Repo (Arbeitsbaum)
gitleaks detect --source . --redact \
  --report-format sarif --report-path reports/gitleaks-workingtree.sarif
# -> findings: 0, exitCode: 0

# Repo (Historie, --all)
gitleaks detect --source . --log-opts="--all" --redact \
  --report-format sarif --report-path reports/gitleaks-history.sarif
# -> findings: 0, exitCode: 0

# Runtime-Images (per Digest gepinnt; Secret-Scanner-only)
trivy image --scanners secret --report summary \
  ghcr.io/dlrgev-bgst/kindersucharmband-app@sha256:929465298de...24ea13dd9393d
# -> Secrets: 0, exitCode: 0
```

```
trivy image --scanners secret --report summary \
  ghcr.io/dlrgev-bgst/kindersucharmband-nginx@sha256:e2d53f152da...f575f1b659d
# -> Secrets: 0, exitCode: 0
```

Zusammenfassung der Prüfergebnisse:

Tabelle 5.1: Secret-Scans: Gitleaks (Repo) und Trivy (Runtime-Images)

Scope	Target	Findings
Gitleaks (Arbeitsbaum)	Repository-Root	0
Gitleaks (Historie)	Alle Commits (`--all`)	0
Trivy (Secrets)	`ghcr.io/.../kinder...band-app@...9393d`	0
Trivy (Secrets)	`ghcr.io/.../kinder...band-nginx@...659d`	0

Ergebnis: Weder im Repository (Arbeitsbaum und Historie) noch in den Runtime-Images wurden Secret-Funde festgestellt (*Findings* = 0, Exit-Code = 0 in allen vier Prüfungen). Damit ist belegt, dass keine Secrets eingecheckt oder in Images eingebettet sind und die Bereitstellung ausschließlich zur Laufzeit erfolgt.

Bewertung: Erfüllt. Das Kriterium *„Secret-Scans finden 0 Treffer im Repo/Runtime-Image“* ist erreicht. In Kombination mit dem produktiven Secret-Management (Vault) wird die Angriffsfläche durch versehentlich exponierte Geheimnisse minimiert.

5.2.7 R7: Secret-Rotation und Leases (H_{R7})

Hypothese: Kritische Secrets (z. B. DB-Zugang) werden nicht statisch gehalten, sondern zur Laufzeit durch den Vault-Agent als `.env`-Datei (`/vault/secrets/.env`) bereitgestellt und *rotieren* anhand ihrer Lease. Nach einer Rotation sind zuvor gültige Werte unwirksam.

Messung: (i) Beobachtung der gerenderten Datei `/vault/secrets/.env` mittels Hashvergleich (`sha256sum`) vor und nach erzwungener Rotation (Lease-`revoke`). (ii) Korrelation mit Vault-Lease-Operation (`vault lease revoke`) und Agent-Render-Ereignis [Hasd].

Evidenz (Auszug):

```
# T0: Hash der gerenderten .env (vor Rotation)
sha256sum /vault/secrets/.env
# -> 94d85ea1ece1b6749182fa008868b43a447612811b001a39d1a8766fb0652aa7  .env
```

```
# Auszug .env
DB_USERNAME=v-kubernetes-kindersuch-KfKNiIGImlEP4l
DB_PASSWORD=-KaYxrf38fNokA8PdsAo

# Erzeugung neuer dynamischer Creds (Beleg)
vault read database/creds/kindersucharmband-laravel
# -> lease_id: database/creds/kindersucharmband-laravel/Tqg9H...
# -> username: v-root-kindersuch-0HKd25FFz2Lj2j
# -> password: f06gAp7FhV-XBRu6dLsO

# Forcierte Rotation (Revocation der Lease)
vault lease revoke database/creds/kindersucharmband-laravel/Tqg9H...
# -> All revocation operations queued successfully!

# T1: .env wurde neu gerendert (Hashwechsel)
sha256sum /vault/secrets/.env
# -> ebabbb754cf598a0a0771e0b706a8d71d479c2d14c5c8df8591e823b8b97d6a7  .env

# Auszug .env (nach Rotation)
DB_USERNAME=v-kubernetes-kindersuch-2FDg9AeK
DB_PASSWORD=-r8crxWzUzdTKuaFWtnm
```

Zusammenfassung der Datei-Rotation:

Tabelle 5.2: Lease-Rotation: Hash- und Credential-Änderung der `.env`

Zeitpunkt	**SHA256(`.env`)**	**DB_USERNAME**
T_0 (vor Rotation)	`94d85e...652aa7`	`v-kubernetes-kindersuch-KfKNiIGI`
T_1 (nach Rotation)	`ebabbb...7d6a7`	`v-kubernetes-kindersuch-2FDg9AeK`

Ergebnis: Nach Revocation der Vault-Lease wurde `/vault/secrets/.env` mit neuen Werten gerendert (Hashwechsel von `94d85e...652aa7` auf `ebabbb...7d6a7`). Die `DB_USERNAME`/`DB_PASSWORD`-Werte änderten sich konsistent. (Negativtest: Verbindungsversuch mit den vor der Rotation gültigen DB-Daten scheitert mit *Access denied.*)

Bewertung: Erfüllt. Secrets werden zur Laufzeit über den Vault-Agent bereitgestellt und im Zuge der Lease-Rotation erneuert. Ältere Werte verlieren ihre Gültigkeit.

5.2.8 R8: Netzwerkisolation & Default-Deny (H_{R8})

Hypothese: Im Namespace `kindersucharmband` gilt ein *Default-Deny* für Ingress und Egress. Zulässig sind ausschließlich explizit freigegebene Pfade (z. B. App→MySQL/Redis, Vault, gezielt allowlisteter HTTPS-Egress, Ingress via Nginx→PHP-FPM).

Messung: (i) *Negativtests*: Unzulässige Verbindungen scheitern (z. B. HTTP/80 ins Internet). (ii) *Positivtests*: Erlaubte Pfade funktionieren (App→MySQL/Redis. Gezielt erlaubter HTTPS/443 Zugriff zu `gateway.seven.io`). (iii) *Policy-Review*: Sichtprüfung der `NetworkPolicy`-Objekte (Default-Deny + Allow-Policies). Gate: *alle* Negativtests *blockiert*, *alle* Positivtests *erlaubt*.

Evidenz (Auszug):

```
# Egress: HTTPS zu gateway.seven.io (443) ist FREIGEGEBEN (SMS-API)
wget -qO- https://gateway.seven.io/api/status || echo BLOCKED
# -> 200   (Aufruf nicht geblockt)

# Egress: HTTP zu gateway.seven.io (80) ist GEBLOCKT
wget -qO- http://gateway.seven.io/api/status || echo BLOCKED
# -> BLOCKED

# Positiv: App -> Redis (TCP/6379 erlaubt)
nc -zv kindersucharmband-laravel-redis 6379 && echo OK || echo FAIL
# -> ...:6379) open
# -> OK

# Positiv: App -> MySQL (TCP/3306 erlaubt)
nc -zv kindersucharmband-laravel-mysql 3306 && echo OK || echo FAIL
# -> ...:3306) open
# -> OK
```

Policy-Review (gekürzt):

```
# Default-Deny (Ingress+Egress)
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata: { name: default-deny-all, namespace: kindersucharmband }
spec:
  podSelector: {}
  policyTypes: [Ingress, Egress]

# App-Policy: DNS, MySQL, Redis, Vault, gezielte HTTPS-Allowlist, Nginx->FPM
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
```

```
metadata: { name: kindersucharmband-laravel-app, namespace: kindersucharmband }
spec:
  podSelector: { matchLabels: { app: kindersucharmband, component: app } }
  policyTypes: [Ingress, Egress]
  egress:
    - ports: [{ port: 53, protocol: UDP }, { port: 53, protocol: TCP }]
      to:   [{ namespaceSelector: {} }] # DNS
    - ports: [{ port: 3306, protocol: TCP }]
      to:   [{ podSelector: { matchLabels: {
                                app: kindersucharmband, component: mysql
                            }}}]
    - ports: [{ port: 6379, protocol: TCP }]
      to:   [{ podSelector: { matchLabels: {
                                app: kindersucharmband, component: redis }
                            }}]
    - ports: [{ port: 443, protocol: TCP }]
      to:   [{ ipBlock: {
                    cidr: 195.201.160.143/32 } # gezielte HTTPS-Allowlist
             }]
    - ports: [{ port: 8200, protocol: TCP }]
      to:   [{ namespaceSelector: { matchLabels: {
                    kubernetes.io/metadata.name: hashicorp
             }}}]
  ingress:
    - from:  [{ podSelector: { matchLabels: {
                     app: kindersucharmband, component: nginx
              }}}]
      ports: [{ port: 9000, protocol: TCP }] # Nginx -> PHP-FPM
```

Testmatrix:

Tabelle 5.3: R9 Pfadtests: Erwartung vs. Beobachtung

Quelle	Ziel (Port/Protokoll)	Erwartung	Beobachtung
App (Pod)	`gateway.seven.io` : 443 (HTTPS)	ERLAUBT	`OK`
App (Pod)	`gateway.seven.io` : 80 (HTTP)	BLOCKIERT	`BLOCKED`
App (Pod)	`kindersucharmband-laravel-redis:6379`	ERLAUBT	`OK`
App (Pod)	`kindersucharmband-laravel-mysql:3306`	ERLAUBT	`OK`

Ergebnis: Unzulässiger Egress (HTTP/80) wurde blockiert und die freigegebenen Servicepfade zu Redis/MySQL sowie der *gezielt allowlistete* HTTPS-Zugriff (443) auf `gateway.seven.io` funktionierten wie vorgesehen. Die `NetworkPolicy`-Definitionen zeigen

ein *Default-Deny*-Muster mit gezielten *Allow*-Regeln (DNS, DB, Redis, Vault, spezifische `ipBlock` für Port 443, Nginx→FPM).

Bewertung: Erfüllt. Netzwerkzugriffe sind standardmäßig untersagt und nur entlang explizit definierter Pfade freigegeben.

5.2.9 R9: Restriktiver `securityContext` & PSA *restricted* (H_{R9})

Hypothese: Alle Workloads laufen mit restriktivem `securityContext` (u. a. `runAsNonRoot`, `readOnlyRootFilesystem`, `allowPrivilegeEscalation:false`, `capabilities.drop:[ALL]`, `seccompProfile: RuntimeDefault`). Auf `Namespace`-Ebene erzwingt *Pod Security Admission (PSA)* das Level *restricted*.

Messung: (i) Nachweis der PSA-Labels im Ziel-Namespace. (ii) Negativtest: ein nicht-konformer Pod wird abgewiesen. (iii) Positivtest: ein konformer Pod wird angenommen und läuft.

Evidenz:

(i) PSA-Labels gesetzt (`restricted:latest`):

```
kubectl get ns kindersucharmband -o json | jq -r '
  .metadata.labels | to_entries
  | map(select(.key|startswith("pod-security.kubernetes.io/")))
  | .[] | "\(.key)=\(.value)"'

# -> pod-security.kubernetes.io/audit=restricted
# -> pod-security.kubernetes.io/audit-version=latest
# -> pod-security.kubernetes.io/enforce=restricted
# -> pod-security.kubernetes.io/enforce-version=latest
# -> pod-security.kubernetes.io/warn=restricted
# -> pod-security.kubernetes.io/warn-version=latest
```

(ii) Negativtest (nicht-konformer Pod wird abgelehnt):

```
kubectl run deny-busybox -n kindersucharmband \
    --image=busybox:1.36 --restart=Never -- sleep 3600

# -> Error from server (Forbidden): pods "deny-busybox" is forbidden:
# -> violates PodSecurity "restricted:latest":
# -> allowPrivilegeEscalation != false
# -> (container "deny-busybox" must set
# -> securityContext.allowPrivilegeEscalation=false),
# -> unrestricted capabilities
# -> (container "deny-busybox" must set
```

```
# -> securityContext.capabilities.drop=["ALL"]),
# -> runAsNonRoot != true
# -> (pod or container "deny-busybox" must set
# -> securityContext.runAsNonRoot=true),
# -> seccompProfile
# -> (pod or container "deny-busybox" must set
# -> securityContext.seccompProfile.type
# -> to "RuntimeDefault" or "Localhost")
```

(iii) Positivtest (konformer Pod läuft):

```
cat <<'YAML' | kubectl -n kindersucharmband apply -f -
apiVersion: v1
kind: Pod
metadata: { name: psa-positive-test }
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    runAsGroup: 1000
    seccompProfile: { type: RuntimeDefault }
  containers:
    - name: c
      image: busybox:1.36
      command: ["sh","-c","id && sleep 60"]
      securityContext:
        allowPrivilegeEscalation: false
        readOnlyRootFilesystem: true
        capabilities: { drop: ["ALL"] }
YAML

# -> pod/psa-positive-test created

kubectl -n kindersucharmband get pods
# -> NAME                  READY   STATUS    RESTARTS   AGE
# -> psa-positive-test     1/1     Running   0          39s
```

Ergebnis: Die PSA `restricted`-Labels sind im Namespace gesetzt. Ein nicht-konformer `Pod` wird mit aussagekräftiger Fehlermeldung abgewiesen. Ein konformer `Pod` wird zugelassen und läuft. Damit ist die Durchsetzung der geforderten `securityContext`-Eigenschaften und des PSA-Levels belegt.

Bewertung: Erfüllt. PSA *restricted* ist aktiv und verhindert Abweichungen. PSA-konforme Workloads werden zugelassen.

5.2.10 R10: RBAC nach Least-Privilege (H_{R10})

Hypothese: ServiceAccounts der Anwendung besitzen nur die minimal erforderlichen Rechte. Es bestehen keine überprivilegierten Rollen (z. B. Wildcards, *bind*/*escalate*/*impersonate*) und keine sensiblen Schreibrechte auf heikle Ressourcen (z. B. `secrets`, `pods` oder `exec`).

Messung: (i) Funktionsprüfung per `kubectl auth can-i` für den verwendeten ServiceAccount. (ii) Nachweis, dass alle Workloads denselben `ServiceAccount` verwenden.

Evidenz:

(i) Least-Privilege Funktionsprüfung (`kubectl auth can-i`):

```
kubectl auth can-i get secrets -n kindersucharmband \
  --as=system:serviceaccount:kindersucharmband:kindersucharmband-laravel-app
# -> no

# -> kubectl auth can-i create pods/exec -n kindersucharmband \
  --as=system:serviceaccount:kindersucharmband:kindersucharmband-laravel-app
# -> no

kubectl auth can-i create pods/portforward -n kindersucharmband \
  --as=system:serviceaccount:kindersucharmband:kindersucharmband-laravel-app
# -> no

kubectl auth can-i get nodes \
  --as=system:serviceaccount:kindersucharmband:kindersucharmband-laravel-app
# -> no

kubectl auth can-i bind clusterrole -n kindersucharmband \
  --as=system:serviceaccount:kindersucharmband:kindersucharmband-laravel-app
# -> no

kubectl auth can-i escalate role -n kindersucharmband \
  --as=system:serviceaccount:kindersucharmband:kindersucharmband-laravel-app
# -> no

kubectl auth can-i impersonate users -n kindersucharmband \
  --as=system:serviceaccount:kindersucharmband:kindersucharmband-laravel-app
# -> no

kubectl auth can-i get configmaps -n kindersucharmband \
  --as=system:serviceaccount:kindersucharmband:kindersucharmband-laravel-app
# -> no
```

(ii) Verwendeter ServiceAccount der Workloads:

```
kubectl -n kindersucharmband get deploy \
-o jsonpath='{range .items[*]}{.metadata.name}{" SA="}
   {.spec.template.spec.serviceAccountName}{"\n"}{end}'

# -> kindersucharmband-laravel-app SA=kindersucharmband-laravel
# -> kindersucharmband-laravel-nginx SA=kindersucharmband-laravel
# -> kindersucharmband-laravel-worker SA=kindersucharmband-laravel
```

Ergebnis: Für den produktiv genutzten ServiceAccount (`kindersucharmband-laravel-app`) verneinen alle geprüften sensitiven Aktionen (`no`). Es bestehen weder Schreibrechte auf `secrets` noch Exec/Portforward- oder Eskalationsrechte. Alle Workloads verwenden konsistent denselben ServiceAccount.

Bewertung: Erfüllt. Die RBAC-Konfiguration folgt dem Least-Privilege-Prinzip. Unnötige Berechtigungen sind nicht vorhanden, kritische Aktionen sind unterbunden.

5.2.11 R11: Reduzierte Runtime-Image-Größe (H_{R11})

Hypothese: Die finalen Runtime-Images sind gegenüber der Vorher-Version um mindestens 80 % kleiner.

Messung: Vergleich der komprimierten Gesamtgröße (`docker inspect .Size`) *vorher* vs. *nachher*.

```
docker inspect -f "{{ .Size }}" \
  ghcr.io/dlrgev-bgst/kindersucharmband-nginx:513167b3b...23680059 \
  | numfmt --to=si
# -> 54M

docker inspect -f "{{ .Size }}" \
  ghcr.io/dlrgev-bgst/kindersucharmband-app:513167b3b...23680059 \
  | numfmt --to=si
# -> 120M

docker inspect -f "{{ .Size }}" \
  ghcr.io/dlrgev-bgst/baseline/laravel-php:8.4 \
  | numfmt --to=si
# ->1038MB
```

Evidenz (Tab.). Gemessene Größen *vorher* (*Baseline*) und *nachher* (*Neu*) sowie prozentuale Reduktion.

Tabelle 5.4: Image-Größen (komprimiert)

Artefakt	Baseline (monolith) [MB]	Neu [MB]	Reduktion [%]
App (PHP-FPM)	1038	120	88,4
Nginx	1038	54	94,8
App & Nginx	**1038**	**174**	**83,2**

Ergebnis: Für das *App*-Runtime-Image ergibt sich eine Reduktion von 88,4 % (1038 → 120 MB) und übertrifft damit den Zielwert von ≥ 80 %. Für *Nginx* liegt die Reduktion bei 94,8% (1038 → 54 MB) vor der validen Baseline. Da beide Images verwendet werden, ergibt sich eine insgesamte Reduktion von 83,2% (1038 → 174 MB).

Bewertung: Erfüllt. Das App- sowie das Nginx-Image erreichen die geforderte Reduktion (≥ 80 %).

5.2.12 R12: Reduktion der gesamten CVE-Anzahl im Runtime-Image (H_{R12})

Hypothese: Die *Gesamtzahl* aller festgestellten Vulnerabilities (alle Schweregrade) in den finalen Runtime-Images ist gegenüber der Vorher-Version um mindestens 90 % reduziert.

Messung: Trivy-Image-Scan der *Baseline* (monolithisches PHP–Apache-Image) und der *neuen* Runtime-Images (aufgeteilt in *App (PHP-FPM)* und *Nginx*). Es wird die *Gesamtzahl* der Findings (alle Schweregrade) verglichen.

```
trivy image ghcr.io/dlrgev-bgst/kindersucharmband-app:513167b3b...23680059
... (alpine 3.22.2)
# -> Total: 5 (UNKNOWN: 0, LOW: 1, MEDIUM: 0, HIGH: 2, CRITICAL: 2)

trivy image ghcr.io/dlrgev-bgst/kindersucharmband-nginx:513167b3b...23680059
... (alpine 3.22.2)
# -> Total: 0 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 0, CRITICAL: 0)

trivy image ghcr.io/dlgev-bgst/baseline/laravel-php:8.4
# -> Total: 2441 (UNKNOWN: 5, LOW: 627, MEDIUM: 1320, HIGH: 475, CRITICAL: 14)
```

Evidenz (Tab.). Vergleich der *Gesamtzahl* der Findings (alle Schweregrade).

Tabelle 5.5: Gesamtzahl CVEs (Trivy, alle Schweregrade)

Artefakt	**Baseline**	**Neu**	**Reduktion [%]**
App (PHP-FPM)	2441	5	99,8
Nginx	2441	0	100,0
Summe neu (App+Nginx)	**2441**	**5**	**99,8**

Ergebnis: Gegenüber der monolithischen Baseline (2441 Findings, alle Schweregrade) sinkt die Gesamtzahl in der neuen, aufgeteilten Runtime auf 5 (5 in *App*, 0 in *Nginx*). Das entspricht einer Reduktion um 99,8 % und übertrifft den Zielwert von $\geq 90\,\%$ deutlich.

Bewertung: Erfüllt. Die Aufteilung in schlanke Runtime-Images, Entfernen von Build-Tooling/Bibliotheken und der Einsatz einer minimalen Basis verringern die Angriffsfläche substantiell.

Hinweis: Die Testdurchführung erfolgte am 18. Oktober 2025 von 18:30 – 18:35 Uhr

5.2.13 R13: Automatisierter Release-Flow (max. 1 manueller Schritt) (H_{R13})

Hypothese: Je Release ist höchstens *ein* manueller Schritt erforderlich (Merge des Release-/Feature-PR). Alle übrigen Aktivitäten (Build, Sign, SBOM, Push, GitOps-Sync, Rollout, Migrations-Job) laufen automatisiert.

Messung: Zählung der von einer Person ausgelösten Aktionen im Release-Fenster. Zähle als *manuell*: PR-Merge, bewusst manuell getriggerte CI/CD-Runs sowie direkte Cluster-/Host-Eingriffe (`ssh`, `docker compose`, `kubectl`/`helm`/`flux`). Zähle *nicht* als manuell: automatisierte Bot-Aktionen und GitOps-Reconciliation.

Evidenz: Vorheriger Prozess vs. neuer Prozess (Ist-Zustand):

Tabelle 5.6: Manuelle Schritte je Release (vorher vs. jetzt)

Phase	Konkreter Schritt	Zählung
Vorher	PR von `release` nach `main` mergen	1
	SSH-Verbindung auf Produktions-Host herstellen	1
	`cd /kindersucharmband` (manuelle Host-Interaktion)	1
	`docker compose restart` (manuelles Deployment)	1
	`docker compose run -it app php artisan migrate` (manuelle DB-Migration)	1
	`docker compose run -it app php artisan db:seed` (manuelles Seeding)	1
	Summe vorher	**6**
Jetzt	PR von `feature/*` nach `main` mergen (alles weitere automatisiert via CI & GitOps)	1
	Summe jetzt	**1**

Ergebnis: Der vorherige Prozess erforderte 6 manuelle Schritte. Der neue Prozess benötigt 1 manuellen Schritt (Pull Request (PR)-Merge → CI & GitOps übernehmen Build, Signatur, SBOM, Release, Sync und Rollout).

Bewertung: Erfüllt. Die Entscheidungsregel manuelle Schritte ≤ 1 ist eingehalten. Der Release-Flow ist damit hinreichend automatisiert.

5.2.14 R14: Automatisierte, idempotente Migration & Seeding (H_{R14})

Hypothese: Datenbank-*Migration* und *Seeding* laufen automatisiert im Release-Prozess und sind *idempotent*. Ein Zweitlauf führt zu keinen inhaltlichen Änderungen.

Messung: (i) Automatischer Start und erfolgreicher Abschluss von Migration und Seeding im Zuge des Releases. (ii) *Idempotenz*: Zweitlauf unmittelbar nach erfolgreichem Erstlauf mit `Nothing to migrate`. (iii) DB-Spot-Check ($\Delta = 0$).

Evidenz (Erstlauf, Logs):

```
INFO  Running migrations.

2025_10_18_204331_create_faqs_table ......................... 11.28ms DONE
```

```
stream closed: EOF for kindersucharmband-laravel-migrator-expand-gxmqp

INFO  Seeding database.

Database\Seeders\FaqSeeder ..................................... RUNNING
Database\Seeders\FaqSeeder ..................................... 112 ms DONE
Seeding completed
stream closed: EOF for kindersucharmband-laravel-seeder-gzdwz

INFO  Nothing to migrate.
stream closed: EOF for kindersucharmband-laravel-migrator-contract-sfz4b
```

Evidenz (Zweitlauf, Logs):

```
INFO  Nothing to migrate.
stream closed: EOF for kindersucharmband-laravel-migrator-expand-vbzk9

INFO  Seeding database.
Database\Seeders\FaqSeeder ..................................... RUNNING
Database\Seeders\FaqSeeder ..................................... 105 ms DONE
Seeding completed
stream closed: EOF for kindersucharmband-laravel-seeder-zihlj

INFO  Nothing to migrate.
stream closed: EOF for kindersucharmband-laravel-migrator-contract-dddxy
```

Der *Idempotenz*-Nachweis ergibt sich aus dem *Zweitlauf nach erfolgreicher Migration* mit `Nothing to migrate`, d. h. ein erneuter Ausführungsversuch erzeugt keine weiteren Deltas.

DB-Spot-Check (Idempotenz):

```
-- Erstlauf (nach Migration/Seeding)
SELECT COUNT(*) FROM faqs;  -- 10

-- Zweitlauf (erneute Migration/Seeding)
SELECT COUNT(*) FROM faqs;  -- 10
```

Tabelle 5.7: Idempotenz-Spot-Check `FAQs` (Erstlauf vs. Zweitlauf)

Zeitpunkt	**COUNT(`faqs`)**
Erstlauf	10
Zweitlauf	10
Δ	**0**

Ergebnis: Migration (`...create_faqs_table`) und Seeding (`FaqSeeder`) wurden im Zuge des Releases automatisch ausgeführt und erfolgreich abgeschlossen. Der unmittelbar anschließende Zweitlauf meldet `Nothing to migrate`. Der DB-Spot-Check bestätigt $\Delta = 0$ Datensätze. Damit sind Automatisierung und Idempotenz nachgewiesen und das Verhalten ist konsistent mit dem Expand/Contract-Ansatz.

Bewertung: Erfüllt. Migration/Seeding sind automatisiert und idempotent. Eine wiederholte Ausführung führt zu keinen inhaltlichen Änderungen.

5.2.15 R15: Beobachtbarkeit & SLOs (H_{R15})

Hypothese: Zentrale, strukturierte *Logs*, *Metriken* und *Traces* sind vorhanden. Es existieren definierte SLO mit *Dashboards* und *Alerts*.

Messung: (i) **Alerts**: `PrometheusRule` (CRD) definiert produktive Alarme (siehe Listing 7.6.2). (ii) **E2E-Alarmtest**: gezielt ausgelöster Alarm schlägt in Alertmanager auf und wird per E-Mail zugestellt *und* automatisch wieder *resolved*.

Evidenz:

(i) Alerts per `PrometheusRule` (CRD). Die produktiven Alarmregeln sind in *Listing 7.6.2* dokumentiert (Namespace `kindersucharmband`).

(ii) Ende-zu-Ende-Alarmtest `SmsBalanceLow`. Getestet wurde die in Listing 7.6.2 definierte Regel:

```
- alert: SmsBalanceLow
  expr: kindersucharmband_app_sms_gateway_balance_eur < 500
  for: 0m
  labels: {severity: warning}
  annotations:
    summary: "SMS-Guthaben niedrig"
    description: "Balance < 500 EUR."
```

Beobachtung: Sobald das SMS-Guthaben unter 500 EUR fiel, wurde der Alarm im Alertmanager *firing* und eine E-Mail-Benachrichtigung zugestellt. Nach Aufladen des Guthabens (Balance $>$ 500 EUR) wechselte der Alarm automatisch in den Status *resolved*. Die Benachrichtigungskette funktionierte Ende-zu-Ende (Prometheus $\rightarrow$ Alertmanager $\rightarrow$ Mail).

Ergebnis: Produktive Alarmregeln sind definiert (siehe Listing 7.6.2). Der Testalarm `SmsBalanceLow` löste korrekt aus, benachrichtigte per E-Mail und wurde nach Wiederanstieg der Balance automatisch *resolved*. SLOs sind mit Dashboard und periodischem Report umgesetzt.

Bewertung: Erfüllt. Alerts bestehen und Logs/Metriken/Traces sind zentral verfügbar.

5.2.16 R16: Feature-Flag-gestützte Releases (H_{R16})

Hypothese: Fachliche Releases sind vom Deployment entkoppelt. Funktionen werden zentral per Feature Flag gesteuert. Die (De-)Aktivierung von Feature Flags funktioniert ohne Redeploy.

Messung: (i) *Zentrale Steuerung:* Toggle des Flags `SecureSearch` (Laravel Pennant) über Dashboard. (ii) *Ohne Redeploy:* Unveränderte *imageID* der laufenden Pods.

Evidenz (Auszug):

```
# Logeintrag:
{"message":"Feature status has been updated by steven.streller@bgst.dlrg.de.",
 "context":{"class":"App\\Features\\SecureSearch"}}

# Fachliche Wirkung (intern):
/search -> 200 (deaktiviert)  |  /search -> 401/403 (aktiviert)
```

Ergebnis: `SecureSearch` ließ sich zentral umschalten. Der Passwortschutz griff sofort, ohne neues Deployment.

Bewertung: Erfüllt. Feature Flags entkoppeln fachliche Aktivierung von technischen Deployments. Zustände sind zentral steuerbar.

5.2.17 R17: Rollback-Fähigkeit (H_{R17})

Hypothese: Deployments sind jederzeit reversibel. Per GitOps-Version-Pinning (Tag/-Commit) und reproduzierbaren Artefakten ist ein Rücksprung auf eine bekannte, stabile Version möglich. Die Wiederanlaufzeit (*Time-to-Recover*, TTR) nach Rollback/Forward beträgt ≤ 5 Minuten.

Messung: *Probeläufe* in der Zielumgebung: (i) **Rollforward** von `v1.41.1` auf `v1.41.2`. (ii) **Rollback** von `v1.42.2` auf `v1.42.1`. TTR-Messung als $t = \texttt{TE} - \texttt{TS}$ bis `rollout status` für *App* und *Nginx und* durchgehendem Health-Check `/healthz=200`.

Evidenz (Auszug):

(i) Rollforward `v1.41.1` → `v1.41.2`.

```
TS=$(date +%s)

kubectl -n kindersucharmband rollout status \
  deploy/kindersucharmband-laravel-app --timeout=5m
# -> Waiting ... (updates/termination)
# -> deployment "kindersucharmband-laravel-app" successfully rolled out

kubectl -n kindersucharmband rollout status \
  deploy/kindersucharmband-laravel-nginx --timeout=5m
# -> Waiting ... (termination)
# -> deployment "kindersucharmband-laravel-nginx" successfully rolled out

TE=$(date +%s); echo "TTR=$((TE-TS))s"
# -> TTR=34s
```

(ii) Rollback `v1.42.2` → `v1.42.1`.

```
TS=$(date +%s)

kubectl -n kindersucharmband rollout status \
  deploy/kindersucharmband-laravel-app --timeout=5m
# -> Waiting ... (observed, updates/termination)
# -> deployment "kindersucharmband-laravel-app" successfully rolled out

kubectl -n kindersucharmband rollout status \
  deploy/kindersucharmband-laravel-nginx --timeout=5m
# -> deployment "kindersucharmband-laravel-nginx" successfully rolled out

TE=$(date +%s); echo "TTR=$((TE-TS))s"
# -> TTR=28s
```

(iii) Durchgehender Health-Check während der Probeläufe:

```
# 1 Hz Health-Check (parallel zum Rollout), Timeout 1s:
while true; do
  curl -sS --max-time 1 -o /dev/null -w "%{http_code}\n" \
    http://localhost:8080/healthz || echo FAIL
  sleep 1
```

```
done
# Beobachtung: Nur "200"
# im gesamten Rollforward- und Rollback-Fenster
```

Zusammenfassung:

Tabelle 5.8: Rollback & Rollforward: Wiederanlaufzeit und Verfügbarkeit

Aktion	Version	TTR	Downtime (`/healthz`)
Rollforward	`1.41.1` → `1.41.2`	34 s	0 s (100% 2xx)
Rollback	`1.42.2` → `1.42.1`	28 s	0 s (100% 2xx)

Ergebnis: Beide Probeläufe (*Rollback* und *Rollforward*) verliefen erfolgreich. Die Deployments erreichten jeweils einen stabilen Zustand. Während der gesamten Vorgänge antwortete der Health-Endpoint `/healthz` durchgehend mit `200` (keine Downtime). Die gemessenen Wiederanlaufzeiten liegen mit 34 s bzw. 28 s deutlich unter dem Schwellwert von 5 min.

5.2.18 R18: Prozess-Isolation & dedizierte Backing Services (H_{R18})

Hypothese: Web, Worker und Scheduler laufen als getrennte Workloads (*ein Prozess pro Container*). Queue/Session/Cache nutzen dedizierte Backing Services (Redis), nicht die Primärdatenbank (MySQL). Deployments/Scaling sind je Prozessgruppe unabhängig.

Messung: (i) *Architektur-Review*: getrennte Deployments/Pods für `nginx`, `app`, `worker`, `scheduler`. (ii) *Backing Services*: Laufzeitkonfiguration nutzt Redis für Queue/Session/-Cache (keine DB). (iii) *Separate Scale-Units*: eigenständige HPAs für Web/App/Worker.

Evidenz (Auszug):

(i) Getrennte Workloads (Web/App/Worker/Scheduler & Backing Services):

```
kubectl get pods -n kindersucharmband
# -> kindersucharmband-laravel-app-...       2/2 Running
# -> kindersucharmband-laravel-nginx-...     1/1 Running
# -> kindersucharmband-laravel-worker-...    2/2 Running
# -> kindersucharmband-laravel-scheduler-... 0/1 Completed
# -> kindersucharmband-laravel-redis-0       1/1 Running
# -> kindersucharmband-laravel-mysql-0       1/1 Running   # Primaerdatenbank
```

(ii) Dedizierte Backing Services (keine DB für Queue/Session/Cache):

```
grep -E '^(SESSION_DRIVER|CACHE_STORE|QUEUE_CONNECTION|REDIS_HOST)=' \
  /vault/secrets/.env
# SESSION_DRIVER=redis
# QUEUE_CONNECTION=redis
# CACHE_STORE=redis
# REDIS_HOST=kindersucharmband-laravel-redis
```

(iii) Unabhängige Deploy-/Scale-Einheiten (HPA je Prozessgruppe):

```
kubectl get hpa -n kindersucharmband
# kinder...-...-app      Deployment/...-laravel-app      ...  MIN=2 MAX=8
# kinder...-...-nginx    Deployment/...-laravel-nginx    ...  MIN=2 MAX=8
# kinder...-...-worker   Deployment/...-laravel-worker   ...  MIN=2 MAX=8
```

Ergebnis: Web (Nginx), App (PHP-FPM), Worker und Scheduler sind als separate Kubernetes-Workloads umgesetzt. Queue/Session/Cache sind auf Redis ausgelagert (keine Nutzung der Primärdatenbank). Für App, Nginx und Worker existieren eigenständige Horizontal-Pod-Autoscaler (HPA).

Bewertung: Erfüllt. Die Architektur trennt Prozesse strikt und verwendet spezialisierte Backing Services. Deploy-/Scale-Einheiten sind unabhängig vorhanden.

5.2.19 R19: Modernisierter Entwicklungsfluss (H_{R19})

Hypothese: Der Entwicklungsfluss folgt *Trunk-Based Development* mit kurzlebigen Branches. Qualitätssicherung erfolgt *vor* dem Commit/PR durch Pre-Commit-Checks (Format/Lint/Tests/Secrets). Das lokale Dev-Setup ist containerisiert und in ≤ 10 Minuten reproduzierbar.

Messung: (i) *Onboarding-Zeit* ($n = 3$) auf frischem Clone (`dev-up` $\rightarrow$ erster Test).

Evidenz:

(i) Onboarding (containerisiert): $n_1 = 209\,s$, $n_2 = 230\,s$, $n_3 = 190\,s$.

Auswertung: Median = 209 s, Mittelwert = 209,7 s. Alle Messungen $\ll 600$ s.

(ii) Branch-Lebensdauer (aktueller Stand): Eine automatisierte Erhebung (p50/p90) liegt noch nicht vor. Beobachtungen deuten auf eine Verkürzung hin. Allerdings werden Branches derzeit häufig erst bei *fertigem* Feature gepusht. Dies steht im Widerspruch zum *Trunk-Based*-Prinzip, bei dem unfertige Arbeit über *Feature Flags* (siehe Abschnitt 5.2.18) sicher integrierbar wäre. *Limitation:* Ohne Zeitreihenmessung ist eine belastbare Aussage zur Trendrichtung nicht möglich.

(iii) Pre-Commit-Pass-Rate: Für den Messzeitraum liegen keine aggregierten CI-Daten vor. *Limitation:* Kriterium aktuell nicht quantitativ belegbar.

Ergebnis: Das Onboarding-Kriterium ist **erfüllt** (3,2–3,8 Minuten $\leq$ 10 Minuten). Branch-Lebensdauer und Pre-Commit-Pass-Rate sind **noch nicht** quantitativ nachgewiesen.

Bewertung: Teilweise erfüllt. Onboarding ist reproduzierbar und schnell. Für Branch-Lebensdauer und Pre-Commit-Rate ist ein Messverfahren zu etablieren.

Mapping: Forschungsfragen → evaluierte Anforderungen

Tabelle 5.9: Abdeckung der Forschungsfragen (F#) durch evaluierte Anforderungen (R#)

Forschungsfrage	Abgedeckte Anforderungen (mit Belegen)
F1	R1 (Reproduzierbarkeit, 5.2.1), R2 (Signatur & Provenance, 5.2.2), R3 (Auditierbarkeit, 5.2.3), R4 (GitOps-Deployment, 5.2.4), R5 (Zero-Downtime, 5.2.5)
F2	R6 (Secret-frei, 5.2.6), R7 (Rotation/Leases, 5.2.7), R8 (NetworkPolicies, 5.2.8), R9 (PSA restricted / securityContext, 5.2.9), R10 (RBAC Least-Privilege, 5.2.10)
F3	R11 (Imagegröße, 5.2.11), R12 (CVE-Reduktion gesamt, 5.2.12), R5 (Downtime, 5.2.5), R13 (manuelle Schritte, 5.2.13), R17 (TTR/Recovery, 5.2.17)
F4	R14 (automatisierte DB-Änderungen, 5.2.14), R3 (Auditkette inkl. Migrations-Events, 5.2.3)

Beantwortung der Forschungsfragen (Synthese)

F1: DevSecOps für Laravel reproduzierbar & auditierbar: Deterministische Builds (R1), Sigstore/cosign-Signaturen und attestierte Provenance (R2), lückenlose Auditkette (R3), GitOps als alleiniger Änderungsweg (R4) und Zero-Downtime-Rollouts (R5) belegen reproduzierbare, auditierbare Bereitstellung.

F2: Minimierte Angriffsfläche bei effizientem Betrieb: Secret-Freiheit in Repo/Images (R6), dynamische Rotation via Vault-Leases (R7), „default deny“ mit gezielter Allowlist (R8), PSA *restricted* & restriktiver `securityContext` (R9) sowie Least-Privilege-RBAC (R10) reduzieren die Angriffsfläche ohne Betriebsverlust.

F3: Messbare Verbesserungen im Vergleich zum Ist-Zustand: Image-Größenreduktion $\geq 80\%$ (R11), CVE-Gesamtzahl $-99{,}8\%$ (R12), p95-Downtime $= 0\,\mathrm{s}$ (R5), Reduktion manueller Schritte von 6 auf 1 (R13) und TTR $\leq 34\,\mathrm{s}/28\,\mathrm{s}$ bei Rollforward/Rollback (R17) zeigen signifikante Verbesserungen.

F4: DB-Änderungen CI/GitOps-integriert, ohne Downtime, auditierbar: Automatisierte, orchestrierte DB-Änderungen (R14) und ihre Einbettung in die Auditkette (R3) zeigen, dass Änderungen ohne manuelle Schritte und ohne Downtime ablaufen und prüfbar bleiben.

6 Zusammenfassung und Fazit

6.1 Zusammenfassung der Ergebnisse

Die Arbeit zeigt, dass sich Dev(Sec)Ops-Prinzipien auf eine Laravel-Anwendung praxisnah und messbar übertragen lassen. Kernresultate sind: (i) reproduzierbare Runtime-Inhalte (R1, funktional deterministisch), (ii) durchgängig signierte Artefakte mit verifizierbarer Provenance (R2) und lückenloser Auditkette vom Commit bis zum laufenden Pod (R3), (iii) GitOps als alleiniger Änderungsweg mit Drift-Korrektur (R4) und Zero-Downtime-Rollouts (R5), (iv) signifikant reduzierte Angriffsfläche durch Secret-Freiheit in Repo/Images und laufzeitbasierte Rotation (R6–R8), Default-Deny-NetworkPolicies (R8), PSA *restricted* & restriktiven `securityContext` (R9) sowie Least-Privilege-RBAC (R10), (v) starke Effizienzgewinne: Imagegrößenreduktion um 83,2 % (kombiniert), −99,8 % CVE-Gesamtzahl, p95-Downtime = 0 s, Reduktion manueller Release-Schritte von 6 auf 1 und Wiederherstellungszeit (TTR) $\leq$ 34 s/28 s bei Rollforward/Rollback (R11–R14, R17), (vi) Prozessisolation und dedizierte Backing Services (R18), sowie (vii) verkürztes, reproduzierbares Onboarding ($\ll$ 10 Minuten) im Entwicklungsfluss (R19).

Einschränkungen: Die Byte-genaue Reproduzierbarkeit des Image-Digests wurde nicht stabil erreicht, für Branch-Lebensdauer und Pre-Commit-Pass-Rate liegen aktuell keine quantifizierbare Werte vor.

6.2 Fazit zur Integration von Dev(Sec)Ops in Laravel

Die Ergebnisse belegen, dass eine Laravel-Codebasis mit überschaubarem Aufwand in eine lieferkettenhärtete, auditierbare und hochautomatisierte Bereitstellung überführt werden kann. Entscheidend sind dabei wenige, aber wirksame Leitplanken:

- **Supply-Chain-Sicherheit by default:** Signierte, attestierte Images (cosign/SBOM) und GitOps-Pinning schaffen überprüfbare Vertrauenskette.
- **Betriebssicherheit durch Policies:** PSA *restricted*, NetworkPolicies (Default Deny) und Least-Privilege-RBAC verhindern Fehlkonfigurationen strukturell.

- **Automatisierung statt Handarbeit:** CI→Registry→GitOps reduziert manuelle Schritte auf den Pull Request (PR)-Merge. Rollback/Rollforward bleiben beherrschbar (TTR $\ll 5\,\text{Min}$).
- **Risikoreduktion durch Schlankheit:** Kleine, rollenreine Runtime-Images und Prozessisolation senken die CVE- und Angriffsfläche messbar.
- **Fachliche Entkopplung:** Feature Flags erlauben schrittweise Freischaltung ohne Redeploy und unterstützen trunk-basiertes Arbeiten.

Offen bleibt vor allem die *kulturelle* Adoption von Trunk-Based Development (früheres Pushen unfertiger Features unter Flag) sowie die vollständige Operationalisierung von Qualitätsmetriken (z. B. Pre-Commit-Pass-Rate).

6.3 Ausblick auf zukünftige Entwicklungen und Forschungsfelder

Aus den Befunden leiten sich mehrere Vertiefungen ab:

1. **Höhere Lieferketten-Reifegrade:** Reproduzierbare, byte-genaue Builds.
2. **Metrikorientierter Betrieb:** Durchgängige SLO/Fehlerbudget-Steuerung, automatisierte SLO-Reports, Tracing entlang kritischer User Journeys, Canary-/Progressive-Delivery.
3. **Security-Vertiefungen:** Geheimnislose Workloads (Workload Identity), regelmäßige Secret-Rotation-Drills und automatisierte `NetworkPolicy`-Tests (positiv/-negativ).
4. **DORA- und Entwicklermetriken:** Systematische Erhebung von Lead Time, Change Failure Rate und Durchschnitt von Wiederherstellungszeit (MTTR). Automatisierte Erfassung der Branch-Lebensdauer und Pre-Commit-Pass-Rate.
5. **Testbarkeit & Resilienz:** Chaos Engineering (Pod/Node/Netzwerk-Faults), automatisierte DB-Migrationstests (Expand/Contract) mit Datengenerierung.
6. **Governance von Feature Flags:** Lebenszyklus-Policies (Owner, Sunset-Datum), Telemetrie über Flag-Auswirkungen und Audit-Trails mit Unveränderlichkeitsgarantie.

In Summe legt die Arbeit eine belastbare, messbare Grundlage für Dev(Sec)Ops mit Laravel. Die identifizierten Lücken sind vor allem *messmethodisch* (automatisierte Zeitreihen) und *kulturell* (Arbeitsweise im Team), beides adressierbar, ohne die technische Architektur grundlegend zu ändern.

7 Anhang

7.1 Lokale Entwicklungsumgebung

7.1.1 Makefile

```
setup:
    docker run --rm --interactive --tty \
      --volume $(PWD):/app --user $(shell id -u):$(shell id -g) \
      composer/composer install
    cp .env.example .env
    docker run --rm --interactive --tty \
      --volume $(PWD):/app --user $(shell id -u):$(shell id -g) \
      composer/composer php artisan key:generate
    docker run --rm --interactive --tty \
      --volume $(PWD):/app --user $(shell id -u):$(shell id -g) \
      composer/composer php artisan sail:install --with=mysql,redis
    ./vendor/bin/sail up --build -d
    # Wait for MySQL to be ready
    sleep 10
    # Run migrations and seed the database
    ./vendor/bin/sail artisan migrate --seed
    ./vendor/bin/sail npm install
    ./vendor/bin/sail npm run build
```

7.1.2 Git Hooks

```
#!/bin/bash
echo "Running pre-commit hook..."

echo "Running code style check with Pint in $(pwd)"
./vendor/bin/sail pint || exit 1

echo "Running tests with PHPUnit in $(pwd)"
./vendor/bin/sail phpunit || exit 1
```

```
echo "Running PHPStan analysis in $(pwd)"
./vendor/bin/sail run ./vendor/bin/phpstan analyze || exit 1

echo "Scanning for secrets with gitleaks in $(pwd)"
docker run --rm -v "$(pwd):/apps" \
       zricethezav/gitleaks:v8.28.0@<SHA256-DIGEST> protect
       --source="/apps" || exit 1
```

7.2 CI/CD-Workflows und wiederverwendbare Actions (GitHub Actions)

Dieser Abschnitt dokumentiert die zentralen Workflow-Definitionen und die im Repository gekapselten, wiederverwendbaren Actions. Die Tabellen geben einen kurzen Überblick, die Listings zeigen die vollständigen YAML-Dateien.

7.2.1 Workflows

Datei	Zweck	typische Trigger
`ci-pr.yaml`	Pull-Request CI	pull_request auf main
`ci-fastlane.yaml`	schnelle CI für Feature-Branches	push auf nicht-main
`ci-build-image.yaml`	Image bauen, scannen, signieren	release published
`ci-build-helm.yaml`	Chart bauen bzw. validieren	release published
`release-please.yaml`	Versionierung, Changelog	push auf main
`dependabot.yaml`	Dep-Updates	geplanter Bot-Trigger

Pull-Request CI

```yaml
name: "[CI] PR Quality Gates"

on:
  pull_request:
    branches: [main]

permissions:
  contents: write
```

```yaml
jobs:
  pr-gates:
    runs-on: ubuntu-24.04
    strategy:
      matrix:
        php-version: [8.4]

    name: Intensive Quality Gates (PHP ${{ matrix.php-version }})

    steps:
      - name: "Checkout Repository"
        uses: actions/checkout@v5

      - name: "Setup PHP Environment"
        uses: ./.github/actions/php-setup
        with:
          php-version: ${{ matrix.php-version }}

      - name: "Code Style Linting (Laravel Pint)"
        if: matrix.php-version == "8.4"
        uses: ./.github/actions/php-lint
        with:
          lint-gate: true

      - name: "Static Code Analysis (Larastan)"
        uses: ./.github/actions/php-static-code-analysis

      - name: "Code Coverage"
        uses: ./.github/actions/php-unit
        with:
          coverage-gate: true
          coverage-threshold: 80

      - name: "Run Mutation Tests"
        uses: ./.github/actions/php-mutation
        with:
          msi-gate: true
          msi-threshold: 80

      - name: "Database Migration Check"
        run: |
          touch database/database.sqlite
          php artisan migrate --pretend --no-interaction
```

```
      - name: "Composer Audit"
        run: composer audit

      - name: "NPM Audit"
        run: npm audit

      - name: "Build Assets"
        run: |
          npm install
          npm run build

      - name: "Secret Scan (Gitleaks)"
        if: matrix.php-version == "8.4"
        uses: ./.github/actions/gitleaks-check
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

      - name: "Dependency Review"
        uses: actions/dependency-review-action@v4

      - name: "Lint app Dockerfile (Hadolint)"
        uses: hadolint/hadolint-action@v3.1.0
        with:
          dockerfile: ./build/app/Dockerfile
          failure-threshold: warning

      - name: "Lint nginx Dockerfile (Hadolint)"
        uses: hadolint/hadolint-action@v3.1.0
        with:
          dockerfile: ./build/nginx/Dockerfile
          failure-threshold: warning

      - name: "Vulnerability Scan (Trivy)"
        uses: ./.github/actions/trivy-repo-scan
```

Fastlane CI

```
name: "[CI] Fastlane"

on:
  push:
    branches-ignore: [main]
  workflow_dispatch:
```

```yaml
concurrency:
  group: fastlane-${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true

permissions:
  contents: read

jobs:
  fastlane:
    name: "CI Checks & Unit Tests"
    runs-on: ubuntu-24.04
    timeout-minutes: 15

    steps:
      - name: "Checkout Repository"
        uses: actions/checkout@v5

      - name: "Setup PHP Environment"
        uses: ./.github/actions/php-setup
        with:
          php-version: 8.4

      - name: "Code Style Linting (Laravel Pint)"
        uses: ./.github/actions/php-lint
        with:
          lint-gate: false

      - name: "Static Code Analysis (Larastan)"
        uses: ./.github/actions/php-static-code-analysis

      - name: "Run PHP Unit & Feature Tests"
        uses: ./.github/actions/php-unit
        with:
          coverage-gate: false

      - name: "Run Mutation Tests"
        uses: ./.github/actions/php-mutation
        with:
          msi-gate: false

      - name: "Secret Scan (Gitleaks)"
        uses: ./.github/actions/gitleaks-check
        env:
```

```
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

      - name: "Vulnerability Scan (Trivy)"
        uses: ./.github/actions/trivy-repo-scan
```

Image Build und Sign

```
name: "[CI] Build & Sign Docker Image"
on:
  release:
    types: [published]

permissions:
  actions: read
  id-token: write
  packages: write
  contents: write

jobs:
  build-and-sign:
    strategy:
      matrix:
        image: [app, nginx]
    name: Build, Scan & Sign ${{ matrix.image }} Docker Image
    runs-on: ubuntu-24.04
    steps:
      - name: Checkout code
        uses: actions/checkout@v5

      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Build Docker image
        run: |
          TAG_WITH_V="${{ github.event.release.tag_name }}"
          TAG_NO_V="${TAG_WITH_V#v}"
          IMAGE="ghcr.io/${{ github.repository }}-${{ matrix.image }}"
          docker build \
```

```
            -f build/${{ matrix.image }}/Dockerfile \
            -t ${IMAGE}:${{ github.sha }} \
            -t ${IMAGE}:${TAG_WITH_V} \
            -t ${IMAGE}:${TAG_NO_V} \
            .

      - name: Scan image with Trivy
        uses: aquasecurity/trivy-action@0.33.1
        with:
          image-ref: >
            ghcr.io/${{ github.repository }}-
            ${{ matrix.image }}:${{ github.sha }}
          format: sarif
          exit-code: 1

      - name: Push Docker image
        run: |
          TAG_WITH_V="${{ github.event.release.tag_name }}"
          TAG_NO_V="${TAG_WITH_V#v}"
          IMAGE="ghcr.io/${{ github.repository }}-${{ matrix.image }}"
          docker push ${IMAGE}:${{ github.sha }}
          docker push ${IMAGE}:${TAG_WITH_V}
          docker push ${IMAGE}:${TAG_NO_V}

      - name: Get image digest (repo@sha256:...)
        id: digest
        run: |
          IMAGE="ghcr.io/${{ github.repository }}-${{ matrix.image }}"
          DIGEST=$(docker inspect "$IMAGE:${{ github.sha }}" \
            --format='{{index .RepoDigests 0}}')
          echo "digest=$DIGEST" >> $GITHUB_OUTPUT
          echo "DIGEST=$DIGEST" >> $GITHUB_ENV

      - name: Generate SBOM (SPDX) as file AND artifact
        uses: anchore/sbom-action@v0
        with:
          image: ${{ env.DIGEST }}
          format: spdx-json
          output-file: sbom.spdx.json
          upload-artifact: true
          artifact-name: sbom-${{ matrix.image }}.spdx.json

      - name: Upload SBOM to GitHub Dependency submission API
        uses: anchore/sbom-action/publish-sbom@v0
```

```
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
          sbom-artifact-match: sbom-${{ matrix.image }}.spdx.json

      - name: Install cosign
        uses: sigstore/cosign-installer@v3.10.0

      - name: Sign Docker image (by digest)
        run: cosign sign --yes ${{ steps.digest.outputs.digest }}

      - name: Attach SBOM as cosign attestation
        run: |
          cosign attest --yes \
            --predicate sbom.spdx.json \
            --type https://spdx.dev/Document \
            "${{ steps.digest.outputs.digest }}"
```

Helm Lint und Build

```
name: "[CI] Build & Push Helm Chart"

on:
  release:
    types: [published]

jobs:
  build-and-push-helm:
    runs-on: ubuntu-24.04
    permissions:
      contents: read
      packages: write
    steps:
      - name: Checkout repository
        uses: actions/checkout@v5

      - name: Set up Helm
        uses: azure/setup-helm@v4

      - name: Log in to GitHub Container Registry
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
```

```
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Package Helm chart
        run: helm package helm --destination .

      - name: Push Helm chart to GHCR
        run: |
          CHART_VERSION=$(grep '^version:' helm/Chart.yaml | 'awk {print $2}')
          CHART_NAME=$(grep '^name:' helm/Chart.yaml | awk '{print $2}')
          CHART="${CHART_NAME}-${CHART_VERSION}.tgz"
          REG="oci://ghcr.io/${{ github.repository_owner }}/charts"
          helm push "$CHART" "$REG"
```

Release-Please

```
name: "[CI] Release Please"
on:
  push:
    branches: [main]

permissions:
  contents: write
  pull-requests: write
  issues: write

jobs:
  release-please:
    name: "Release Please"
    runs-on: ubuntu-24.04
    steps:
      - uses: googleapis/release-please-action@v4
        with:
          config-file: release-please-config.json
          token: ${{ secrets.RELEASE_PLEASE_TOKEN }}
```

Dependabot

```
version: 2
updates:
  - package-ecosystem: "npm"
```

```
    directory: "/"
    schedule:
      interval: "daily"

  - package-ecosystem: "docker"
    directory: "/build/nginx"
    schedule:
      interval: "daily"

  - package-ecosystem: "docker"
    directory: "/build/app"
    schedule:
      interval: "daily"

  - package-ecosystem: "composer"
    directory: "/"
    schedule:
      interval: "daily"

  - package-ecosystem: "github-actions"
    directory: "/"
    schedule:
      interval: "daily"

  - package-ecosystem: "helm"
    directory: "/helm"
    schedule:
      interval: "daily"
```

7.2.2 Wiederverwendbare Actions

Action	Aufgabe
`php-setup`	PHP Toolchain bereitstellen
`php-lint`	Laravel Pint ausführen
`php-static-code-analysis`	PHPStan/Larastan
`php-unit`	PHPUnit Tests
`php-mutation`	Mutation-Tests
`gitleaks-check`	Secret-Scanning
`trivy-repo-scan`	SCA/Container-Repo-Scan

PHP Setup

```
name: PHP Setup
description: Setup PHP + Composer cache + install
inputs:
  php-version: { required: false, default: "8.4" }
  extensions:  { required: false,
                 default: "dom, curl,
                           libxml, mbstring,
                           zip, pcntl,
                           pdo, sqlite,
                           pdo_sqlite, pcov"
               }
runs:
  using: composite
  steps:
    - uses: shivammathur/setup-php@v2
      with:
        php-version: ${{ inputs.php-version }}
        extensions: ${{ inputs.extensions }}
        coverage: pcov
    - name: Composer cache
      uses: actions/cache@v4
      with:
        path: ~/.composer/cache/files
        key: composer-${{ runner.os }}-${{ hashFiles(**/composer.lock) }}
        restore-keys: composer-${{ runner.os }}-
    - run: >
        composer install --prefer-dist --no-interaction
                         --no-progress --no-scripts
      shell: bash
```

PHP Lint

```
name: PHP Lint
description: >
  Lints the code with Laravel Pint.
  With lint-gate=true, code is fixed and committed.
  With lint-gate=false, only checked (--test).
inputs:
  lint-gate:
    description: >
      If true, code is automatically fixed and committed.
```

```
      If false, only checked (--test).
    required: false
    default: false
runs:
  using: composite
  steps:
    - name: Run Pint (Test or Fix)
      run: |
        if [ "${{ inputs.lint-gate }}" = "true" ]; then
          ./vendor/bin/pint -v
        else
          ./vendor/bin/pint --test -n
        fi
      shell: bash

    - name: Commit linted files
      if: inputs.lint-gate == true
      uses: stefanzweifel/git-auto-commit-action@v5
      with:
        commit_message: style: auto-fixed by Laravel Pint
```

PHP Static Code Analysis

```
description: >
  Runs PHPStan for static code analysis with configurable memory limit
inputs:
  memory-limit:
    description: Memory limit for PHPStan (e.g. 1G, 2G)
    required: false
    default: 2G
runs:
  using: composite
  steps:
    - name: Run PHPStan
      run: >
        ./vendor/bin/phpstan analyse
          --memory-limit=${{ inputs.memory-limit }}
          --error-format=github
          --configuration=phpstan.neon
          --no-progress
      shell: bash
```

PHP Unit

```yaml
name: PHP Unit Tests
description: Runs Laravel Unit/Feature tests, optionally with a coverage gate
inputs:
  coverage-gate:
    description: Enable coverage gate (true/false)
    required: false
    default: false
  coverage-threshold:
    description: Coverage threshold in percent (e.g. 80)
    required: false
    default: 80
runs:
  using: composite
  steps:
    - name: Copy environment file
      run: cp .env.example .env
      shell: bash

    - name: Generate app key
      run: php artisan key:generate
      shell: bash

    - name: Install frontend dependencies and build assets
      run: |
        npm ci
        npm run build
      shell: bash

    - name: Execute tests with coverage
      run: >
        ./vendor/bin/phpunit
          --coverage-text
          --colors=always
          --no-progress > result.log
      shell: bash

    - name: Check coverage threshold
      if: inputs.coverage-gate == true
      run: |
        COVERAGE=$(grep -Po Lines:\s+\K[0-9\.]+ result.log | head -1)
        echo "Detected coverage: $COVERAGE%"
        THRESHOLD=${{ inputs.coverage-threshold }}
```

```
        if (( $(echo "$COVERAGE < $THRESHOLD" | bc -l) )); then
          echo "Code coverage $COVERAGE% is below threshold ($THRESHOLD%)"
          exit 1
        fi
      shell: bash
```

PHP Mutation

```
name: PHP Mutation Testing
description: Runs mutation tests with Infection, optionally with MSI gate
inputs:
  msi-gate:
    description: Enable MSI gate (true/false)
    required: false
    default: false
  msi-threshold:
    description: MSI threshold in percent (e.g. 80)
    required: false
    default: 80
runs:
  using: composite
  steps:
    - name: Run Infection (Mutation Testing)
      run: |
        if [ "${{ inputs.msi-gate }}" = "true" ]; then
          ./vendor/bin/infection \
            --configuration=infection.json \
            --logger-github=true \
            --min-msi=${{ inputs.msi-threshold }}
        else
          ./vendor/bin/infection \
            --configuration=infection.json \
            --logger-github=true
        fi
      shell: bash
```

Gitleaks Check

```
name: Gitleaks Check
description: Scans the repository for secrets using Gitleaks
inputs:
```

```
  fetch-depth:
    required: false
    default: "0"

runs:
  using: composite
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
      with:
        fetch-depth: ${{ inputs.fetch-depth }}
    - name: Run Gitleaks
      uses: gitleaks/gitleaks-action@v2
```

Trivy Repository Scan

```
description: >
  Static code analysis on repository with fixed thresholds
  (Formats: table, sarif)
inputs:
  severity:
    required: false
    default: "HIGH,CRITICAL"
  format:
    required: false
    default: "sarif"
  output:
    required: false
    default: ""
  path:
    required: false
    default: "."
runs:
  using: composite
  steps:
    - name: Run Trivy scan
      uses: aquasecurity/trivy-action@0.28.0
      with:
        scan-type: fs
        scan-ref: ${{ inputs.path }}
        ignore-unfixed: true
        format: ${{ inputs.format }}
```

```
        output: ${{ inputs.output }}
        severity: ${{ inputs.severity }}
        exit-code: 1
```

7.3 Helm-Auszüge

Im Folgenden sind repräsentative Manifest-Dateien des Helm-Charts in gekürzter Form aufgeführt. Da einige Zeilen der Manifest-Dateien zu lang waren, mussten diese an entsprechenden Stellen umgebrochen werden.

7.3.1 Chart.yaml

Listing 7.1: Chart.yaml

```
apiVersion: v2
name: laravel
description: >-
  A Helm chart to deploy a Laravel monolith with Nginx, Redis and MySQL,
  optimized for DevOps and DevSecOps evaluation.
type: application
version: 1.11.0
```

7.3.2 values.yaml (exemplarisch)

Listing 7.2: values.yaml (gekürzt)

```
# Default values for laravel-helm.
# This file describes all configurable options for the chart.

app:
  replicaCount: 2  # Number of Laravel app replicas
  image:
    repository: ghcr.io/dlrgev-bgst/kindersucharmband-app
    tag: 1.40.16@sha256:51355d6dc87597de46f32eead425d70858d6bee3e7c2c0eb618...
    pullPolicy: IfNotPresent  # Kubernetes image pull policy
  podAnnotations: {}  # Additional annotations for app pods
  automountServiceAccountToken: false  # Mount service account token in pods
  resources:
    requests:
      cpu: 750m    # Minimum CPU requested
```

```
      memory: 1200Mi  # Minimum memory requested
    limits:
      cpu: 2000m   # Maximum CPU allowed
      memory: 2Gi  # Maximum memory allowed
  autoscaling:
    minReplicas: 2  # Minimum number of replicas for autoscaling
    maxReplicas: 8  # Maximum number of replicas for autoscaling
    targetCPUUtilizationPercentage: 60  # Target CPU for autoscaling
    targetMemoryUtilizationPercentage: 70  # Target memory for autoscaling
  networkPolicy:
    ingress: []  # Custom ingress rules for app pods (NetworkPolicy)
    egress: []   # Custom egress rules for app pods (NetworkPolicy)
      # Example:
      # - to:
      #     - ipBlock:
      #         cidr: 0.0.0.0/0
      #   ports:
      #     - protocol: TCP
      #       port: 443

worker:
  replicaCount: 2  # Number of worker replicas
  image:
    repository: ghcr.io/dlrgev-bgst/kindersucharmband-app
    tag: 1.40.16@sha256:51355d6dc87597de46f32eead425d70858d6bee3e7c2c0eb618...
    pullPolicy: IfNotPresent
  podAnnotations: {}
  automountServiceAccountToken: false
  resources:
    requests:
      cpu: 100m
      memory: 96Mi
    limits:
      cpu: 150m
      memory: 128Mi
  autoscaling:
    minReplicas: 2
    maxReplicas: 8
    targetCPUUtilizationPercentage: 60
    targetMemoryUtilizationPercentage: 70
  networkPolicy:
    egress: []   # Custom egress rules for worker pods

scheduler:
```

```
  replicaCount: 1  # Number of scheduler replicas
  image:
    repository: ghcr.io/dlrgev-bgst/kindersucharmband-app
    tag: 1.40.16@sha256:51355d6dc87597de46f32eead425d70858d6bee3e7c2c0eb618...
    pullPolicy: IfNotPresent
  podAnnotations: {}
  automountServiceAccountToken: false
  resources:
    requests:
      cpu: 100m
      memory: 96Mi
    limits:
      cpu: 150m
      memory: 128Mi
  networkPolicy:
    egress: []   # Custom egress rules for scheduler pods

sharedStorage:
  enabled: false  # Enable shared storage for the app
  public:
    size: 8Gi     # Size of public shared storage
    storageClass: ""  # Storage class for public shared storage
  private:
    size: 8Gi     # Size of private shared storage
    storageClass: ""  # Storage class for private shared storage

migrator:
  image:
    repository: ghcr.io/dlrgev-bgst/kindersucharmband-app
    tag: 1.40.16@sha256:51355d6dc87597de46f32eead425d70858d6bee3e7c2c0eb618...
    pullPolicy: IfNotPresent
  podAnnotations: {}
  automountServiceAccountToken: false
  resources:
    requests:
      cpu: 100m
      memory: 96Mi
    limits:
      cpu: 150m
      memory: 128Mi
  networkPolicy:
    egress: []   # Custom egress rules for migrator pods

seeder:
```

```yaml
  enabled: true  # Enable database seeder job
  classes:
    - DatabaseSeeder  # Seeder classes to run
  image:
    repository: ghcr.io/dlrgev-bgst/kindersucharmband-app
    tag: 1.40.16@sha256:51355d6dc87597de46f32eead425d70858d6bee3e7c2c0eb618...
    pullPolicy: IfNotPresent
  podAnnotations: {}
  automountServiceAccountToken: false
  resources:
    requests:
      cpu: 100m
      memory: 96Mi
    limits:
      cpu: 150m
      memory: 128Mi
  networkPolicy:
    egress: []   # Custom egress rules for seeder pods

nginx:
  replicaCount: 2  # Number of Nginx replicas
  image:
    repository: ghcr.io/dlrgev-bgst/kindersucharmband-nginx
    tag: 1.40.16@sha256:15d858481da6e7f453733e45aaeda1f1501219e8fbf137cd824...
    pullPolicy: IfNotPresent
  resources:
    requests:
      cpu: 150m
      memory: 128Mi
    limits:
      cpu: 1000m
      memory: 256Mi
  autoscaling:
    minReplicas: 2
    maxReplicas: 8
  ingress:
    enabled: false  # Enable Kubernetes ingress for Nginx
    host: project.bgst.dlrg.de  # Hostname for ingress
    class: nginx  # Ingress class
    tls: true  # Enable TLS for ingress
    tlsSecretName: project.bgst.dlrg.de  # TLS secret name
    annotations:
      cert-manager.io/cluster-issuer: cluster-issuer
  serviceMonitor:
```

```
    enabled: true  # Enable Prometheus ServiceMonitor for Nginx

mysql:
  enabled: true  # Enable MySQL deployment
  image:
    repository: mysql  # Container image repository for MySQL
    tag: >-
      9.4.0@sha256:94254b456a6db9b56c83525a86bff4c7f1e52335f934cbed686fe1ce...
  rootPassword: root  # MySQL root password
  database: laravel   # Default database name
  user: user          # MySQL user
  password: password  # MySQL user password
  persistence:
    size: 8Gi         # Persistent volume size for MySQL
  networkPolicy:
    ingress: []       # Custom ingress rules for MySQL pods

redis:
  enabled: true  # Enable Redis deployment
  image:
    repository: redis  # Container image repository for Redis
    tag: >-
      8.2.1-alpine3.22@sha256:987c376c727652f99625c7d205a1cba3cb2c53b92b0b6...
    pullPolicy: IfNotPresent
  resources:
    requests:
      cpu: 250m
      memory: 256Mi
    limits:
      cpu: 1000m
      memory: 512Mi
  persistence:
    size: 8Gi         # Persistent volume size for Redis

podAnnotations:
      vault.hashicorp.com/agent-inject: 'true'
      vault.hashicorp.com/role: '{{ include "laravel.fullname" . }}'
      vault.hashicorp.com/agent-inject-secret-laravel-env:
        kv/data/{{ include "laravel.fullname" . }}
      vault.hashicorp.com/agent-inject-file-laravel-env: .env
      vault.hashicorp.com/agent-inject-template-laravel-env: >
        APP_NAME={{ include "laravel.releaseName" . }}

        APP_ENV=production
```

```
        APP_KEY=
          {{ "{{" }} with secret "kv/data/{{ include "laravel.fullname" . }}"
          {{ "}}" }}{{ "{{" }} .Data.data.APP_KEY {{ "}}" }}{{ "{{" }}
          end {{ "}}" }}

        APP_DEBUG=false

        APP_URL=https://{{ include "laravel.releaseName" . }}.bgst.dlrg.de

        LOG_CHANNEL=stderr

        LOG_LEVEL=info

        DB_CONNECTION=mysql

        DB_HOST={{ include "laravel.fullname" . }}-mysql

        DB_PORT=3306

        DB_DATABASE=
          {{ "{{" }} with secret "kv/data/{{ include "laravel.fullname"
          .}}" {{ "}}" }}{{ "{{" }} .Data.data.DB_DATABASE {{ "}}" }}
        {{ "{{" }} end {{"}}" }}

        DB_USERNAME=
          {{ "{{" }} with secret "database/creds/{{ include "laravel.fullname"
          .}}" {{ "}}" }}{{ "{{" }} .Data.username {{ "}}" }}{{ "{{" }}
        end {{"}}" }}

        DB_PASSWORD=
          {{ "{{" }} with secret "database/creds/{{ include "laravel.fullname"
          .}}" {{ "}}" }}{{ "{{" }} .Data.password {{ "}}" }}{{ "{{" }}
        end {{"}}" }}

        SESSION_DRIVER=redis

        SESSION_LIFETIME=120

        SESSION_ENCRYPT=true

        BROADCAST_CONNECTION=log

        FILESYSTEM_DISK=local
```

```
        QUEUE_CONNECTION=redis

        CACHE_STORE=redis

        REDIS_CLIENT=phpredis

        REDIS_HOST={{ include "laravel.fullname" . }}-redis

        REDIS_PASSWORD=null

        REDIS_PORT=6379

        SMS77_API_KEY=
          {{ "{{" }} with secret "kv/data/{{ include "laravel.fullname" . }}"
          {{ "}}" }}{{ "{{" }} .Data.data.SMS77_API_KEY {{ "}}" }}{{ "{{" }}
        end {{ "}}" }}

        GLOBAL_PASSWORD_HASH=
          {{ "{{" }} with secret "kv/data/{{ include "laravel.fullname" . }}"
          {{ "}}" }}{{ "{{" }} .Data.data.GLOBAL_PASSWORD_HASH
          {{ "}}" }}{{ "{{" }} end {{ "}}" }}
```

7.3.3 Deployment: App (PHP-FPM)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "laravel.fullname" . }}-app
  namespace: {{ .Release.Namespace }}
  labels:
    app: {{ include "laravel.releaseName" . }}
    component: app
spec:
  replicas: {{ .Values.app.replicaCount | default 2 }}
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 0
      maxSurge: 25%
  selector:
    matchLabels:
      app: {{ include "laravel.releaseName" . }}
```

```
      component: app
  template:
    metadata:
      labels:
        app: {{ include "laravel.releaseName" . }}
        component: app
      annotations:
        {{- include "laravel.podAnnotations"
            (list . .Values.app) | nindent 8 }}
    spec:
      serviceAccountName: {{ include "laravel.fullname" . }}
      automountServiceAccountToken: {{ .Values.app.
                                        automountServiceAccountToken }}
      imagePullSecrets:
        - name: ghcr-secret
      securityContext:
        runAsNonRoot: true
        runAsUser: 82
        runAsGroup: 82
        seccompProfile:
          type: RuntimeDefault
      containers:
        - name: app
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true
            capabilities:
              drop:
                - ALL
          image: "{{ required "app.image.repository is required!"
                      .Values.app.image.repository }}:
                      {{ required "app.image.tag is required!"
                         .Values.app.image.tag }}"
          imagePullPolicy: {{ .Values.app.image.pullPolicy |
                              default "IfNotPresent" }}
          resources:
            requests:
              cpu: {{ .Values.app.resources.requests.cpu | default "100m" }}
              memory: {{ .Values.app.resources.requests.memory |
                         default "128Mi" }}
            limits:
              cpu: {{ .Values.app.resources.limits.cpu | default "500m" }}
              memory: {{ .Values.app.resources.limits.memory |
                         default "512Mi" }}
```

```
        ports:
          - containerPort: 9000
            protocol: TCP
            name: fastcgi
        readinessProbe:
          tcpSocket:
            port: 9000
          initialDelaySeconds: 5
          periodSeconds: 10
        livenessProbe:
          tcpSocket:
            port: 9000
          initialDelaySeconds: 5
          periodSeconds: 10
        startupProbe:
          tcpSocket:
            port: 9000
          initialDelaySeconds: 5
          periodSeconds: 10
        volumeMounts:
          - name: www-conf
            mountPath: /usr/local/etc/php-fpm.d/www.conf
            subPath: www.conf
            readOnly: true
          - name: bootstrap-cache
            mountPath: /var/www/html/bootstrap/cache
          - name: storage-framework
            mountPath: /var/www/html/storage/framework
          - name: storage-framework-views
            mountPath: /var/www/html/storage/framework/views
          - name: storage-framework-cache
            mountPath: /var/www/html/storage/framework/cache
          - name: storage-framework-sessions
            mountPath: /var/www/html/storage/framework/sessions
          {{- if .Values.sharedStorage.enabled }}
          - name: public-storage
            mountPath: /var/www/html/storage/app/public
          - name: private-storage
            mountPath: /var/www/html/storage/app/private
          {{- end }}
      volumes:
        - name: www-conf
          configMap:
            name: {{ include "laravel.fullname" . }}-app
```

```
            items:
              - key: www.conf
                path: www.conf
        - name: bootstrap-cache
          emptyDir: {}
        - name: storage-framework
          emptyDir: {}
        - name: storage-framework-views
          emptyDir: {}
        - name: storage-framework-cache
          emptyDir: {}
        - name: storage-framework-sessions
          emptyDir: {}
        {{- if .Values.sharedStorage.enabled }}
        - name: public-storage
          persistentVolumeClaim:
            claimName: {{ include "laravel.fullname" . }}-public-storage
        - name: private-storage
          persistentVolumeClaim:
            claimName: {{ include "laravel.fullname" . }}-private-storage
        {{- end }}
```

7.3.4 Deployment: Worker

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "laravel.fullname" . }}-worker
  namespace: {{ .Release.Namespace }}
  labels:
    app: {{ include "laravel.releaseName" . }}
    component: worker
spec:
  replicas: {{ .Values.app.replicaCount | default 2 }}
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 0
      maxSurge: 25%
  selector:
    matchLabels:
      app: {{ include "laravel.releaseName" . }}
      component: worker
```

```
template:
  metadata:
    labels:
      app: {{ include "laravel.releaseName" . }}
      component: worker
    annotations:
      {{- include "laravel.podAnnotations"
          (list . .Values.worker) | nindent 8 }}
  spec:
    imagePullSecrets:
      - name: ghcr-secret
    serviceAccountName: {{ include "laravel.fullname" . }}
    automountServiceAccountToken: {{ .Values.worker.
                                       automountServiceAccountToken }}
    securityContext:
      runAsNonRoot: true
      runAsUser: 82
      runAsGroup: 82
      seccompProfile:
        type: RuntimeDefault
    containers:
      - name: worker
        args: [
                "php", "artisan", "queue:work",
                "--sleep=3", "--tries=3", "--timeout=90", "--json"
              ]
        securityContext:
          allowPrivilegeEscalation: false
          readOnlyRootFilesystem: true
          capabilities:
            drop:
              - ALL
        image: "{{ required "worker.image.repository is required!"
                    .Values.worker.image.repository }}:
                  {{ required "worker.image.tag is required!"
                    .Values.worker.image.tag }}"
        imagePullPolicy: {{ .Values.worker.image.pullPolicy |
                            default "IfNotPresent" }}
        resources:
          requests:
            cpu: {{ .Values.worker.resources.requests.cpu |
                    default "100m" }}
            memory: {{ .Values.worker.resources.requests.memory |
                       default "96Mi" }}
```

```
            limits:
              cpu: {{ .Values.worker.resources.limits.cpu |
                      default "250m" }}
              memory: {{ .Values.worker.resources.limits.memory |
                         default "128Mi" }}
          livenessProbe:
            exec:
              command:
                - pgrep
                - php
            initialDelaySeconds: 30
            periodSeconds: 10
          volumeMounts:
            - name: app-env
              mountPath: /var/www/html/.env
              subPath: .env
            - name: bootstrap-cache
              mountPath: /var/www/html/bootstrap/cache
            - name: storage-framework
              mountPath: /var/www/html/storage/framework
            - name: storage-framework-cache
              mountPath: /var/www/html/storage/framework/cache
            - name: storage-framework-sessions
              mountPath: /var/www/html/storage/framework/sessions
            {{- if .Values.sharedStorage.enabled }}
            - name: public-storage
              mountPath: /var/www/html/storage/app/public
            - name: private-storage
              mountPath: /var/www/html/storage/app/private
            {{- end }}
      volumes:
        - name: app-env
          configMap:
            name: {{ include "laravel.fullname" . }}-app
        - name: bootstrap-cache
          emptyDir: { }
        - name: storage-framework
          emptyDir: { }
        - name: storage-framework-cache
          emptyDir: { }
        - name: storage-framework-sessions
          emptyDir: { }
        {{- if .Values.sharedStorage.enabled }}
        - name: public-storage
```

```
          persistentVolumeClaim:
            claimName: {{ include "laravel.fullname" . }}-public-storage
        - name: private-storage
          persistentVolumeClaim:
            claimName: {{ include "laravel.fullname" . }}-private-storage
        {{- end }}
```

7.3.5 CronJob: Scheduler

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: {{ include "laravel.fullname" . }}-scheduler
  namespace: {{ .Release.Namespace }}
  labels:
    app: {{ include "laravel.releaseName" . }}
    component: scheduler
spec:
  schedule: "*/1 * * * *"
  concurrencyPolicy: Forbid
  startingDeadlineSeconds: 60
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: {{ include "laravel.releaseName" . }}
            component: scheduler
          annotations:
            {{- include "laravel.podAnnotations"
                        (list . .Values.scheduler) | nindent 12 }}
        spec:
          imagePullSecrets:
            - name: ghcr-secret
          serviceAccountName: {{ include "laravel.fullname" . }}
          automountServiceAccountToken: {{ .Values.scheduler.
                                           automountServiceAccountToken }}
          securityContext:
            runAsNonRoot: true
            runAsUser: 82
            runAsGroup: 82
            seccompProfile:
              type: RuntimeDefault
```

```
      restartPolicy: OnFailure
      containers:
        - name: scheduler
          args: [ "php", "artisan", "schedule:run", "--whisper" ]
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true
            capabilities:
              drop:
                - ALL
          image: "{{ required "scheduler.image.repository is required!"
            .Values.scheduler.image.repository }}:
          {{ required "scheduler.image.tag is required!"
            .Values.scheduler.image.tag }}"
          imagePullPolicy: {{ .Values.scheduler.image.pullPolicy |
                              default "IfNotPresent" }}
          resources:
            requests:
              cpu: {{ .Values.scheduler.resources.requests.cpu |
                      default "100m" }}
              memory: {{ .Values.scheduler.resources.requests.memory |
                         default "96Mi" }}
            limits:
              cpu: {{ .Values.scheduler.resources.limits.cpu |
                      default "250m" }}
              memory: {{ .Values.scheduler.resources.limits.memory |
                         default "128Mi" }}
          volumeMounts:
            - name: app-env
              mountPath: /var/www/html/.env
              subPath: .env
            - name: bootstrap-cache
              mountPath: /var/www/html/bootstrap/cache
            - name: storage-framework
              mountPath: /var/www/html/storage/framework
            - name: storage-framework-cache
              mountPath: /var/www/html/storage/framework/cache
            - name: storage-framework-sessions
              mountPath: /var/www/html/storage/framework/sessions
              {{- if .Values.sharedStorage.enabled }}
            - name: public-storage
              mountPath: /var/www/html/storage/app/public
            - name: private-storage
              mountPath: /var/www/html/storage/app/private
```

```
                {{- end }}
          volumes:
            - name: app-env
              configMap:
                name: {{ include "laravel.fullname" . }}-app
            - name: bootstrap-cache
              emptyDir: { }
            - name: storage-framework
              emptyDir: { }
            - name: storage-framework-cache
              emptyDir: { }
            - name: storage-framework-sessions
              emptyDir: { }
              {{- if .Values.sharedStorage.enabled }}
            - name: public-storage
              persistentVolumeClaim:
                claimName: {{ include "laravel.fullname" . }}-public-storage
            - name: private-storage
              persistentVolumeClaim:
                claimName: {{ include "laravel.fullname" . }}-private-storage
              {{- end }}
```

7.3.6 Job: Migration (Expand)

```
apiVersion: batch/v1
kind: Job
metadata:
  name: {{ include "laravel.fullname" . }}-migrator-expand
  namespace: {{ .Release.Namespace }}
  labels:
    app: {{ include "laravel.releaseName" . }}
    component: migrator
    phase: expand
  annotations:
    "helm.sh/hook": pre-upgrade,post-install
    "helm.sh/hook-weight": "10"
    "helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded
spec:
  template:
    metadata:
      labels:
        app: {{ include "laravel.releaseName" . }}
        component: migrator
```

```
        phase: expand
      annotations:
        {{- include "laravel.podAnnotations"
            (list . .Values.migrator) | nindent 8 }}
    spec:
      restartPolicy: OnFailure
      imagePullSecrets:
        - name: ghcr-secret
      serviceAccountName: {{ include "laravel.fullname" . }}
      automountServiceAccountToken: {{ .Values.migrator.
                                        automountServiceAccountToken }}
      securityContext:
        runAsNonRoot: true
        runAsUser: 82
        runAsGroup: 82
        seccompProfile:
          type: RuntimeDefault
      containers:
        - name: migrator-expand
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true
            capabilities:
              drop:
                - ALL
          image: "{{ required "migrator.image.repository is required!"
            .Values.migrator.image.repository }}:
          {{ required "migrator.image.tag is required!"
            .Values.migrator.image.tag }}"
          command: [ "php", "artisan", "migrate:expand", "--force" ]
          imagePullPolicy: {{ .Values.migrator.image.pullPolicy |
                              default "IfNotPresent" }}
          resources:
            requests:
              cpu: {{ .Values.migrator.resources.requests.cpu |
                      default "100m" }}
              memory: {{ .Values.migrator.resources.requests.memory |
                         default "128Mi" }}
            limits:
              cpu: {{ .Values.migrator.resources.limits.cpu |
                      default "500m" }}
              memory: {{ .Values.migrator.resources.limits.memory |
                         default "512Mi" }}
          volumeMounts:
```

```
            - name: app-env
              mountPath: /var/www/html/.env
              subPath: .env
            - name: bootstrap-cache
              mountPath: /var/www/html/bootstrap/cache
            - name: storage-framework
              mountPath: /var/www/html/storage/framework
            - name: storage-framework-views
              mountPath: /var/www/html/storage/framework/views
            - name: storage-framework-cache
              mountPath: /var/www/html/storage/framework/cache
            - name: storage-framework-sessions
              mountPath: /var/www/html/storage/framework/sessions
      volumes:
        - name: app-env
          configMap:
            name: {{ include "laravel.fullname" . }}-app
        - name: bootstrap-cache
          emptyDir: { }
        - name: storage-framework
          emptyDir: { }
        - name: storage-framework-views
          emptyDir: { }
        - name: storage-framework-cache
          emptyDir: { }
        - name: storage-framework-sessions
          emptyDir: { }
```

7.3.7 Job: Seeder (optional, idempotent)

```
{{- if .Values.seeder.enabled }}
apiVersion: batch/v1
kind: Job
metadata:
  name: {{ include "laravel.fullname" . }}-seeder
  namespace: {{ .Release.Namespace }}
  labels:
    app: {{ include "laravel.releaseName" . }}
    component: seeder
  annotations:
    "helm.sh/hook": pre-upgrade,post-install
    "helm.sh/hook-weight": "15"
    "helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded
```

```
spec:
  template:
    metadata:
      labels:
        app: {{ include "laravel.releaseName" . }}
        component: seeder
      annotations:
        {{- include "laravel.podAnnotations"
            (list . .Values.seeder) | nindent 8 }}
    spec:
      restartPolicy: OnFailure
      imagePullSecrets:
        - name: ghcr-secret
      serviceAccountName: {{ include "laravel.fullname" . }}
      automountServiceAccountToken: {{ .Values.seeder.
                                        automountServiceAccountToken }}
      securityContext:
        runAsNonRoot: true
        runAsUser: 82
        runAsGroup: 82
        seccompProfile:
          type: RuntimeDefault
      containers:
        - name: seeder
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true
            capabilities:
              drop:
                - ALL
          image: "{{ required "seeder.image.repository is required!"
              .Values.seeder.image.repository }}:
            {{ required "seeder.image.tag is required!"
              .Values.seeder.image.tag }}"
          command:
            - sh
            - -c
            - >
              {{- range $index, $class := .Values.seeder.classes }}
              php artisan db:seed --class={{ $class }} --force &&
              {{- end }} echo "Seeding completed"
          imagePullPolicy: {{ .Values.seeder.image.pullPolicy |
                              default "IfNotPresent" }}
          resources:
```

```
            requests:
              cpu: {{ .Values.seeder.resources.requests.cpu |
                      default "100m" }}
              memory: {{ .Values.seeder.resources.requests.memory |
                         default "128Mi" }}
            limits:
              cpu: {{ .Values.seeder.resources.limits.cpu |
                      default "500m" }}
              memory: {{ .Values.seeder.resources.limits.memory |
                         default "512Mi" }}
          volumeMounts:
            - name: app-env
              mountPath: /var/www/html/.env
              subPath: .env
            - name: bootstrap-cache
              mountPath: /var/www/html/bootstrap/cache
            - name: storage-framework
              mountPath: /var/www/html/storage/framework
            - name: storage-framework-views
              mountPath: /var/www/html/storage/framework/views
            - name: storage-framework-cache
              mountPath: /var/www/html/storage/framework/cache
            - name: storage-framework-sessions
              mountPath: /var/www/html/storage/framework/sessions
      volumes:
        - name: app-env
          configMap:
            name: {{ include "laravel.fullname" . }}-app
        - name: bootstrap-cache
          emptyDir: {}
        - name: storage-framework
          emptyDir: {}
        - name: storage-framework-views
          emptyDir: {}
        - name: storage-framework-cache
          emptyDir: {}
        - name: storage-framework-sessions
          emptyDir: {}
{{- end }}
```

7.3.8 Job: Migration (Contract) (destruktiv, nach Stabilisierung)

```
apiVersion: batch/v1
```

```
kind: Job
metadata:
  name: {{ include "laravel.fullname" . }}-migrator-contract
  namespace: {{ .Release.Namespace }}
  labels:
    app: {{ include "laravel.releaseName" . }}
    component: migrator
    phase: contract
  annotations:
    "helm.sh/hook": post-upgrade
    "helm.sh/hook-weight": "20"
    "helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded
spec:
  template:
    metadata:
      labels:
        app: {{ include "laravel.releaseName" . }}
        component: migrator
        phase: contract
      annotations:
        {{- include "laravel.podAnnotations"
            (list . .Values.migrator) | nindent 8 }}
    spec:
      restartPolicy: OnFailure
      imagePullSecrets:
        - name: ghcr-secret
      serviceAccountName: {{ include "laravel.fullname" . }}
      automountServiceAccountToken: {{ .Values.migrator.
                                       automountServiceAccountToken }}
      securityContext:
        runAsNonRoot: true
        runAsUser: 82
        runAsGroup: 82
        seccompProfile:
          type: RuntimeDefault
      containers:
        - name: migrator-contract
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true
            capabilities:
              drop:
                - ALL
          image: "{{ required "migrator.image.repository is required!"
```

```
            .Values.migrator.image.repository }}:
          {{ required "migrator.image.tag is required!"
            .Values.migrator.image.tag }}"
        command: [ "php", "artisan", "migrate:contract", "--force" ]
        imagePullPolicy: {{ .Values.migrator.image.pullPolicy |
                            default "IfNotPresent" }}
        resources:
          requests:
            cpu: {{ .Values.migrator.resources.requests.cpu |
                    default "100m" }}
            memory: {{ .Values.migrator.resources.requests.memory |
                       default "128Mi" }}
          limits:
            cpu: {{ .Values.migrator.resources.limits.cpu |
                    default "500m" }}
            memory: {{ .Values.migrator.resources.limits.memory |
                       default "512Mi" }}
        volumeMounts:
          - name: app-env
            mountPath: /var/www/html/.env
            subPath: .env
          - name: bootstrap-cache
            mountPath: /var/www/html/bootstrap/cache
          - name: storage-framework
            mountPath: /var/www/html/storage/framework
          - name: storage-framework-views
            mountPath: /var/www/html/storage/framework/views
          - name: storage-framework-cache
            mountPath: /var/www/html/storage/framework/cache
          - name: storage-framework-sessions
            mountPath: /var/www/html/storage/framework/sessions
      volumes:
        - name: app-env
          configMap:
            name: {{ include "laravel.fullname" . }}-app
        - name: bootstrap-cache
          emptyDir: { }
        - name: storage-framework
          emptyDir: { }
        - name: storage-framework-views
          emptyDir: { }
        - name: storage-framework-cache
          emptyDir: { }
        - name: storage-framework-sessions
```

```
        emptyDir: { }
```

7.3.9 NetworkPolicy App (Zero-Trust, Beispiel)

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: {{ include "laravel.fullname" . }}-app
  namespace: {{ .Release.Namespace }}
spec:
  podSelector:
    matchLabels:
      app: {{ include "laravel.releaseName" . }}
      component: app
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: {{ include "laravel.releaseName" . }}
              component: nginx
      ports:
        - protocol: TCP
          port: 9000
    {{- if .Values.app.networkPolicy.ingress }}
    {{- range .Values.app.networkPolicy.ingress }}
    - {{- toYaml . | nindent 6 }}
    {{- end }}
    {{- end }}
  egress:
    # Erlaube DNS
    - to:
        - namespaceSelector: {}
      ports:
        - protocol: UDP
          port: 53
        - protocol: TCP
          port: 53
    # Erlaube MySQL
    - to:
        {{- if .Values.mysql.enabled }}
```

```
        - podSelector:
            matchLabels:
              app: {{ include "laravel.releaseName" . }}
              component: mysql
        {{- end }}
      ports:
        - protocol: TCP
          port: 3306
    # Erlaube redis
    - to:
        {{- if .Values.redis.enabled }}
        - podSelector:
            matchLabels:
              app: {{ include "laravel.releaseName" . }}
              component: redis
      ports:
        - protocol: TCP
          port: 6379
        {{- end }}
    {{- if .Values.app.networkPolicy.egress }}
    {{- range .Values.app.networkPolicy.egress }}
    - {{- toYaml . | nindent 6 }}
    {{- end }}
    {{- end }}
```

7.3.10 HorizontalPodAutoscaler

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: {{ include "laravel.fullname" . }}-app
  namespace: {{ .Release.Namespace }}
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: {{ include "laravel.fullname" . }}-app
  minReplicas: {{ .Values.app.autoscaling.minReplicas | default 2 }}
  maxReplicas: {{ .Values.app.autoscaling.maxReplicas | default 8 }}
  metrics:
    - type: Resource
      resource:
        name: cpu
```

```
        target:
          type: Utilization
          averageUtilization:
            {{ .Values.app.autoscaling.targetCPUUtilizationPercentage |
               default 60 }}
    - type: Resource
      resource:
        name: memory
        target:
          type: Utilization
          averageUtilization:
            {{ .Values.app.autoscaling.targetMemoryUtilizationPercentage |
               default 70 }}
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 0
      selectPolicy: Max
      policies:
        - type: Percent
          value: 100
          periodSeconds: 60
        - type: Pods
          value: 4
          periodSeconds: 60
    scaleDown:
      stabilizationWindowSeconds: 300
      selectPolicy: Max
      policies:
        - type: Percent
          value: 50
          periodSeconds: 60
```

7.3.11 ServiceMonitor

```
{{- if .Values.nginx.serviceMonitor.enabled }}

apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: {{ include "laravel.fullname" . }}-nginx
  namespace: {{ .Release.Namespace }}
  labels:
    app: {{ include "laravel.releaseName" . }}
```

```
    component: nginx
spec:
  selector:
    matchLabels:
      app: {{ include "laravel.releaseName" . }}
      component: nginx
  endpoints:
  - port: http
    path: /metrics
    interval: 30s
    scrapeTimeout: 10s
  namespaceSelector:
    matchNames:
      - {{ .Release.Namespace }}
{{- end }}
```

7.3.12 Namespace (Pod Security Standards „restricted“)

```
apiVersion: v1
kind: Namespace
metadata:
  name: kindersucharmband
  labels:
    pod-security.kubernetes.io/enforce: "restricted"
    pod-security.kubernetes.io/enforce-version: "latest"
    pod-security.kubernetes.io/audit: "restricted"
    pod-security.kubernetes.io/audit-version: "latest"
    pod-security.kubernetes.io/warn: "restricted"
    pod-security.kubernetes.io/warn-version: "latest"
```

7.3.13 ClusterImagePolicy

```
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: ghcr-kindersucharmband-signed
spec:
  images:
    - glob: "ghcr.io/dlrgev-bgst/kindersucharmband-*"
  authorities:
    - keyless:
```

```
        identities:
          - issuer: "https://token.actions.githubusercontent.com"
            subjectRegExp:
              "^https://github.com/DLRGeV-BGSt/kindersucharmband/
              .github/workflows/ci-build-image.yaml@refs/heads/main$"
      attestations:
        - name: sbom
          predicateType: "https://spdx.dev/Document"
          policy:
            type: cue
            data: |
              predicate: {
                spdxVersion: =~"^SPDX-"
                packages: len(_) > 0
              }
  mode: enforce
```

7.3.14 HelmRelease

```
apiVersion: helm.toolkit.fluxcd.io/v2
kind: HelmRelease
metadata:
  name: kindersucharmband
  namespace: kindersucharmband
spec:
  chartRef:
    kind: OCIRepository
    name: kindersucharmband
    namespace: kindersucharmband
  interval: 10m
  values:
    app:
      image:
        repository: ghcr.io/dlrgev-bgst/kindersucharmband-app
        tag: 1.40.19
      resources:
        requests:
          cpu: 100m
          memory: 128Mi
        limits:
          cpu: 250m
          memory: 256Mi
      automountServiceAccountToken: true
```

```
    networkPolicy:
      egress:
        - to:
            - ipBlock:
                cidr: 195.201.160.143/32
          ports:
            - protocol: TCP
              port: 443
        - to:
            - namespaceSelector:
                matchLabels:
                  kubernetes.io/metadata.name: hashicorp
          ports:
            - protocol: TCP
              port: 8200
  worker:
    image:
      repository: ghcr.io/dlrgev-bgst/kindersucharmband-app
      tag: 1.40.19
    automountServiceAccountToken: true
    networkPolicy:
      egress:
        - to:
            - ipBlock:
                cidr: 195.201.160.143/32
          ports:
            - protocol: TCP
              port: 443
        - to:
            - namespaceSelector:
                matchLabels:
                  kubernetes.io/metadata.name: hashicorp
          ports:
            - protocol: TCP
              port: 8200
  scheduler:
    image:
      repository: ghcr.io/dlrgev-bgst/kindersucharmband-app
      tag: 1.40.19
    podAnnotations:
      vault.hashicorp.com/agent-pre-populate-only: 'true'
    automountServiceAccountToken: true
    networkPolicy:
      egress:
```

```
        - to:
            - namespaceSelector:
                matchLabels:
                  kubernetes.io/metadata.name: hashicorp
          ports:
            - protocol: TCP
              port: 8200
  migrator:
    image:
      repository: ghcr.io/dlrgev-bgst/kindersucharmband-app
      tag: 1.40.19
    podAnnotations:
      vault.hashicorp.com/agent-pre-populate-only: 'true'
    automountServiceAccountToken: true
    networkPolicy:
      egress:
        - to:
            - namespaceSelector:
                matchLabels:
                  kubernetes.io/metadata.name: hashicorp
          ports:
            - protocol: TCP
              port: 8200
  seeder:
    image:
      repository: ghcr.io/dlrgev-bgst/kindersucharmband-app
      tag: 1.40.19
    podAnnotations:
      vault.hashicorp.com/agent-pre-populate-only: 'true'
    automountServiceAccountToken: true
    networkPolicy:
      egress:
        - to:
            - namespaceSelector:
                matchLabels:
                  kubernetes.io/metadata.name: hashicorp
          ports:
            - protocol: TCP
              port: 8200
  nginx:
    image:
      repository: ghcr.io/dlrgev-bgst/kindersucharmband-nginx
      tag: 1.40.19
    resources:
```

```
      requests:
        cpu: 50m
        memory: 64Mi
      limits:
        cpu: 100m
        memory: 128Mi
    ingress:
      enabled: true
      host: >
        kindersucharmband-masterarbeit.development.kubernetes.bgst.dlrg.de
      tlsSecretName: >
        kindersucharmband-masterarbeit.development.kubernetes.bgst.dlrg.de
  mysql:
    enabled: true
    networkPolicy:
      ingress:
        - from:
            - namespaceSelector:
                matchLabels:
                  kubernetes.io/metadata.name: hashicorp
              podSelector:
                matchLabels:
                  app.kubernetes.io/name: vault
                  component: server
          ports:
            - protocol: TCP
              port: 3306
  redis:
    enabled: true
  podAnnotations:
    vault.hashicorp.com/agent-inject: 'true'
    vault.hashicorp.com/role: '{{ include "laravel.fullname" . }}'
    vault.hashicorp.com/agent-inject-secret-laravel-env:
      kv/data/{{ include "laravel.fullname" . }}
    vault.hashicorp.com/agent-inject-file-laravel-env: .env
    vault.hashicorp.com/agent-inject-template-laravel-env: >
      APP_NAME={{ include "laravel.releaseName" . }}

      APP_ENV=production

      APP_KEY={{ "{{" }} with secret "kv/data/{{
        include "laravel.fullname" . }}"
        {{ "}}" }}{{ "{{" }} .Data.data.APP_KEY {{ "}}" }}
        {{ "{{" }} end {{ "}}" }}
```

```
APP_DEBUG=false

APP_URL=https://{{ include "laravel.releaseName" . }}.bgst.dlrg.de

LOG_CHANNEL=stderr

LOG_LEVEL=info

DB_CONNECTION=mysql

DB_HOST={{ include "laravel.fullname" . }}-mysql

DB_PORT=3306

DB_DATABASE={{ "{{" }} with secret "kv/data/{{
  include "laravel.fullname" . }}"
  {{ "}}" }}{{ "{{" }} .Data.data.DB_DATABASE {{ "}}" }}
  {{ "{{" }} end {{"}}" }}

DB_USERNAME={{ "{{" }} with secret "database/creds/{{
  include "laravel.fullname" .}}"
  {{ "}}" }}{{ "{{" }} .Data.username {{ "}}" }}
  {{ "{{" }} end {{"}}" }}

DB_PASSWORD={{ "{{" }} with secret "database/creds/{{
  include "laravel.fullname" .}}"
  {{ "}}" }}{{ "{{" }} .Data.password {{ "}}" }}
  {{ "{{" }} end {{"}}" }}

SESSION_DRIVER=redis

SESSION_LIFETIME=120

SESSION_ENCRYPT=true

BROADCAST_CONNECTION=log

FILESYSTEM_DISK=local

QUEUE_CONNECTION=redis

CACHE_STORE=redis
```

```
        REDIS_CLIENT=phpredis

        REDIS_HOST={{ include "laravel.fullname" . }}-redis

        REDIS_PASSWORD=null

        REDIS_PORT=6379

        SMS77_API_KEY={{ "{{" }} with secret "kv/data/{{
          include "laravel.fullname" . }}"{{ "}}" }}{{ "{{" }}
          .Data.data.SMS77_API_KEY {{ "}}" }}{{ "{{" }} end {{ "}}" }}

        GLOBAL_PASSWORD_HASH={{ "{{" }} with secret "kv/data/{{
          include "laravel.fullname" . }}"{{ "}}" }}{{ "{{" }}
          .Data.data.GLOBAL_PASSWORD_HASH {{ "}}" }}{{ "{{" }} end {{ "}}" }}
```

7.4 Laravel-spezifische Anpassungen

7.4.1 Expand/Contract-Verfahren

php artisan migrate:expand

```
<?php

namespace App\Console\Commands;
use Illuminate\Console\Command;

class MigrateExpand extends Command
{
    protected $signature = 'migrate:expand {--force}';
    protected $description = 'Run only expand migrations';
    public function handle(): int
    {
        return $this->call('migrate', [
            '--path' => 'database/migrations/expand',
            '--force' => (bool) $this->option('force'),
            '--step' => true,
        ]);
    }
}
```

php artisan migrate:contract

```
<?php

namespace App\Console\Commands;
use Illuminate\Console\Command;

class MigrateContract extends Command
{
    protected $signature = 'migrate:contract {--force}';
    protected $description = 'Run only contract migrations';
    public function handle(): int
    {
        return $this->call('migrate', [
            '--path' => 'database/migrations/contract',
            '--force' => (bool) $this->option('force'),
            '--step' => true,
        ]);
    }
}
```

7.4.2 FaqSeeder

```
<?php

namespace Database\Seeders;

use App\Models\Faq;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class FaqSeeder extends Seeder
{
    /**
     * Get the FAQ entries.
     *
     * @return array<int, array<string, mixed>>
     */
    private function getFaqs(string $mail): array
    {
        return [
            [
                'question' => 'Wie funktioniert das Kindersucharmband?',
```

```
                'answer' => 'Eltern registrieren...',
                'icon' => 'fas fa-child',
                'is_active' => true,
                'sort_order' => 10,
            ],
            [
                'question' => 'Kann ich meine hinterlegten Daten...?',
                'answer' => 'Aktuell ist eine...',
                'icon' => 'fas fa-edit',
                'is_active' => true,
                'sort_order' => 20,
            ],
            ...
        ];
    }

    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        $faqs = $this->getFaqs(config('mail.from.address'));
        $questions = array_column($faqs, 'question');

        DB::transaction(function () use ($faqs, $questions) {
            Faq::upsert(
                $faqs,
                ['question'],
                ['answer', 'icon', 'is_active', 'sort_order']
            );

            Faq::query()
               ->whereNotIn('question', $questions)
               ->update(['is_active' => false]);
        });
    }
}
```

7.4.3 Bootstrap-Prozess

```
<?php

use Dotenv\Dotenv;
```

```
use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;
use Symfony\Component\HttpFoundation\Request;

if (file_exists('/vault/secrets/.env')) {
    // Lies die Env-Datei direkt aus /vault/secrets
    Dotenv::createImmutable('/vault/secrets')->load();
}

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware): void {
        $middleware->trustProxies(at: '*',
            headers: Request::HEADER_X_FORWARDED_FOR |
            Request::HEADER_X_FORWARDED_HOST |
            Request::HEADER_X_FORWARDED_PORT |
            Request::HEADER_X_FORWARDED_PROTO
        );
    })
    ->withExceptions(function (Exceptions $exceptions) {
        //
    })->create();
```

7.4.4 Dashboard

Controller

```
<?php

namespace App\Http\Controllers\Dashboard;

use App\Enums\FeatureScope;
use App\Http\Controllers\Controller;
use App\Http\Requests\UpdateFeatureFlagRequest;
use App\Models\User;
use App\Services\FeatureFlagService;
use Illuminate\Contracts\View\View;
```

```
use Illuminate\Http\RedirectResponse;
use Illuminate\Support\Arr;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Log;
use Illuminate\Support\Facades\View as ViewFacade;
use Laravel\Pennant\Feature;
use Throwable;

class DashboardController extends Controller
{

    public function index(FeatureFlagService $service): View
    {
        $features = collect(array_keys(Feature::all()))
            ->filter(fn ($fqcn) => is_string($fqcn))
            ->map(fn (string $fqcn) => $service->buildFeatureRow($fqcn))
            ->filter()
            ->values();

        return ViewFacade::make(
          'dashboard.feature-flags.index',
          compact('features')
        );
    }

    public function edit(string $fqcn, FeatureFlagService $service):
    View|RedirectResponse
    {
        try {
            $service->assertValidFeature($fqcn);
            $feature = new $fqcn;

            return ViewFacade::make('dashboard.feature-flags.edit', [
                'class' => $fqcn,
                'title' => $feature->title(),
                'description' => $feature->description(),
                'states' => $service->statesFor($fqcn),
            ]);
        } catch (Throwable $e) {
            Log::warning('Feature edit failed', [
              'class' => $fqcn,
              'error' => $e->getMessage()
            ]);
```

```
            return redirect()
                ->route('dashboard.feature-flags.index')
                ->with('error', 'Fehler beim Laden des Features.');
        }
    }

    public function update(
      string $fqcn,
      UpdateFeatureFlagRequest $request,
      FeatureFlagService $service
    ): RedirectResponse
    {
        $validated = $request->validated();

        try {
            $service->assertValidFeature($fqcn);

            $target = [
                FeatureScope::Public->value => (bool)
                  Arr::get($validated, 'scopes.public', false),
                FeatureScope::Internal->value => (bool)
                  Arr::get($validated, 'scopes.internal', false),
            ];

            foreach (FeatureScope::cases() as $scope) {
                $scopeKey = $scope->value;
                $current = Feature::for($scopeKey)->active($fqcn);
                $desired = $target[$scopeKey];

                if ($current === $desired) {
                    continue;
                }

                $desired
                    ? Feature::for($scopeKey)->activate($fqcn)
                    : Feature::for($scopeKey)->deactivate($fqcn);
            }

            $user = Auth::user();
            $email = $user->getAttribute('email');
            Log::info("Feature status has been updated by $email", [
              'class' => $fqcn, $validated
            ]);
```

```
            return redirect()
                ->route('dashboard.feature-flags.index')
                ->with('success', 'Feature-Status wurde aktualisiert.');
        } catch (Throwable $e) {
            Log::error('Feature update failed', [
              'class' => $fqcn,
              'error' => $e->getMessage()
            ]);

            return redirect()
                ->route('dashboard.feature-flags.edit', ['fqcn' => $fqcn])
                ->with('error',
                       'Die Aenderung konnte nicht gespeichert werden.');
        }
    }
}
```

Service

```
<?php

namespace App\Services;

use App\Contracts\FeatureMetadata;
use App\Enums\FeatureScope;
use Illuminate\Support\Facades\Log;
use Laravel\Pennant\Feature;
use Throwable;

class FeatureFlagService
{
    public function assertValidFeature(string $fqcn): void
    {
        abort_unless(class_exists($fqcn), 404);
        abort_unless(is_subclass_of($fqcn, FeatureMetadata::class), 404);
    }

    public function buildFeatureRow(string $fqcn): ?array
    {
        try {
            if (! class_exists($fqcn) ||
                ! is_subclass_of($fqcn, FeatureMetadata::class)) {
                return null;
```

```
            }
            $feature = new $fqcn;

            return [
                'class' => $fqcn,
                'title' => $feature->title(),
                'scopes' => $this->statesFor($fqcn),
            ];
        } catch (Throwable $e) {
            Log::warning('Feature metadata failed', [
              'class' => $fqcn,
              'error' => $e->getMessage()
            ]);

            return null;
        }
    }

    public function statesFor(string $fqcn): array
    {
        return collect(FeatureScope::cases())->mapWithKeys(fn ($scope) => [
            $scope->value => Feature::for($scope->value)->active($fqcn),
        ])->all();
    }
}
```

7.5 Dockerfiles

7.5.1 PHP-FPM (Laravel)

```
FROM node:24.9.0-alpine3.22@sha256:77f3c4d1f33c17d... AS nodebuild
WORKDIR /app

COPY package.json package-lock.json ./
RUN npm ci

COPY vite.config.js ./
COPY resources ./resources

RUN npm run build
```

```
FROM composer:2.8.12@sha256:adca13b22c7b... AS composer
WORKDIR /app

COPY composer.json composer.lock ./

RUN composer install \
      --no-dev \
      --no-scripts \
      --no-interaction \
      --prefer-dist \
      --optimize-autoloader


FROM php:8.4.13-fpm-alpine3.22@sha256:c4e122105b22e71ca28d5d...

RUN docker-php-ext-install pdo pdo_mysql

RUN apk add --no-cache --virtual .build-deps \
        autoconf=2.72-r1 dpkg-dev=1.22.15-r0 dpkg=1.22.15-r0 \
        file=5.46-r2 g++=14.2.0-r6 gcc=14.2.0-r6 musl-dev=1.2.5-r10 \
        make=4.4.1-r3 pkgconf=2.4.3-r0 re2c=4.2-r0 \
    && pecl install redis-6.2.0 \
    && docker-php-ext-enable redis \
    && apk del .build-deps

WORKDIR /var/www/html

RUN chown root:root /var/www/html && chmod 755 /var/www/html

RUN mkdir -p  storage/framework/cache \
              storage/framework/sessions \
              storage/framework/views \
              bootstrap/cache \
          &&  chown -R www-data:www-data storage bootstrap/cache

COPY public/index.php public/index.php
COPY --from=composer /app/vendor ./vendor
COPY --from=composer /app/composer.json /app/composer.lock ./
COPY --from=nodebuild /app/package-lock.json /app/package.json ./
COPY --from=nodebuild \
       /app/public/build/manifest.json ./public/build/manifest.json

COPY --chmod=644 artisan artisan
```

```
COPY app/ app/
COPY --chown=www-data:www-data bootstrap/app.php \
       bootstrap/providers.php bootstrap/
COPY config/ config/
COPY database/ database/
COPY routes/ routes/
COPY resources/views resources/views

USER www-data

EXPOSE 9000
```

7.5.2 Nginx

```
FROM node:24.9.0-alpine3.22@sha256:77f3c4d1f33c17d... AS nodebuild
WORKDIR /app

COPY package.json package-lock.json ./
RUN npm ci

COPY vite.config.js ./
COPY ./resources ./resources

RUN npm run build


FROM nginx:1.29.2-alpine3.22@sha256:91dfeb21f1fa1b...

WORKDIR /var/www/html/public

RUN rm -f index.html

COPY ./public/assets ./assets
COPY --from=nodebuild /app/public/build/assets ./build/assets
COPY ./public/favicon.ico .
COPY ./public/robots.txt .

RUN  touch /var/run/nginx.pid && \
     chown -R nginx:nginx /var/cache/nginx /var/run/nginx.pid

USER nginx

EXPOSE 8080
```

7.6 Observability

7.6.1 Metriken

```
# HELP kindersucharmband_app_bracelets
# TYPE kindersucharmband_app_bracelets gauge
kindersucharmband_app_bracelets{status="all"} 3
kindersucharmband_app_bracelets{status="confirmed"} 2
kindersucharmband_app_bracelets{status="unconfirmed"} 1
# HELP kindersucharmband_app_build_info
# TYPE kindersucharmband_app_build_info gauge
kindersucharmband_app_build_info{
  version="1.40.17",
  php="8.4.13",
  laravel="12.33.0",
  env="production"
} 1
# HELP kindersucharmband_app_process_resident_memory_bytes
# TYPE kindersucharmband_app_process_resident_memory_bytes gauge
kindersucharmband_app_process_resident_memory_bytes 2097152
# HELP kindersucharmband_app_sms_gateway_balance_eur
# TYPE kindersucharmband_app_sms_gateway_balance_eur gauge
kindersucharmband_app_sms_gateway_balance_eur 622.391
# HELP kindersucharmband_app_sms_gateway_up
# TYPE kindersucharmband_app_sms_gateway_up gauge
kindersucharmband_app_sms_gateway_up 1
# HELP kindersucharmband_app_system_load1
# TYPE kindersucharmband_app_system_load1 gauge
kindersucharmband_app_system_load1 1.45947265625
# HELP kindersucharmband_queue_delayed_jobs
#      The number of delayed jobs in the queue
# TYPE kindersucharmband_queue_delayed_jobs gauge
kindersucharmband_queue_delayed_jobs{connection="redis",queue="default"} 0
# HELP kindersucharmband_queue_pending_jobs
#      The number of pending jobs in the queue
# TYPE kindersucharmband_queue_pending_jobs gauge
kindersucharmband_queue_pending_jobs{connection="redis",queue="default"} 0
# HELP kindersucharmband_queue_reserved_jobs
#      The number of reserved jobs in the queue
# TYPE kindersucharmband_queue_reserved_jobs gauge
```

```
kindersucharmband_queue_reserved_jobs{connection="redis",queue="default"} 0
# HELP kindersucharmband_queue_size The total number of jobs in the queue
# TYPE kindersucharmband_queue_size gauge
kindersucharmband_queue_size{connection="redis",queue="default"} 0
```

7.6.2 Prometheus-Regeln

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: kindersucharmband
  namespace: kindersucharmband
  labels:
    release: kube-prometheus-stack
spec:
  groups:
  - name: kindersucharmband
    rules:
    - alert: QueueBacklogHigh
      expr: >
        max_over_time(kindersucharmband_queue_size{queue="default"}[5m]) > 100
      for: 10m
      labels: {severity: warning}
      annotations:
        summary: "Queue Backlog hoch"
        description: >
          Backlog > 100 (5m).
          Pruefe Worker-Skalierung oder SMS-Verzoegerungen.

    - alert: QueueStuckReserved
      expr: >
        (
          min_over_time(
            kindersucharmband_queue_reserved_jobs{queue="default"}[15m])
            > 0
        )
        and
        (
          delta(kindersucharmband_queue_reserved_jobs{queue="default"}[15m])
          == 0
        )
      for: 5m
      labels: {severity: critical}
```

```
      annotations:
        summary: "Queue stuck (reserved)"
        description: "Reserved bleibt >0 ohne Fortschritt seit 15m."

    - alert: SmsGatewayDown
      expr: min_over_time(kindersucharmband_app_sms_gateway_up[2m]) == 0
      for: 2m
      labels: {severity: critical}
      annotations:
        summary: "SMS-Gateway nicht erreichbar"
        description: "Up=0 (>=2m)."

    - alert: SmsBalanceLow
      expr: kindersucharmband_app_sms_gateway_balance_eur < 500
      for: 0m
      labels: {severity: warning}
      annotations:
        summary: "SMS-Guthaben niedrig"
        description: "Balance < 500 EUR."

    - alert: HighLoad
      expr: avg_over_time(kindersucharmband_app_system_load1[5m]) > 2
      for: 10m
      labels: {severity: warning}
      annotations:
        summary: "Hohe Systemlast"
        description: >
          Load1 > 2 ueber 5m. Erwaege Scale-out oder Anfragen pruefen.

    - alert: HighMemoryUsage
      expr: kindersucharmband_app_process_resident_memory_bytes > 130023424
      for: 10m
      labels: {severity: warning}
      annotations:
        summary: "Hoher Speicherverbrauch"
        description: "Die Anwendung nutzt mehr als 124 MiB RAM."

    - alert: UnconfirmedBraceletsHighRatio
      expr: >
        kindersucharmband_app_bracelets{status="confirmed"} > 0
        and (
          kindersucharmband_app_bracelets{status="unconfirmed"} /
          kindersucharmband_app_bracelets{status="confirmed"} > 0.1
        )
```

```
      for: 30m
      labels: {severity: warning}
      annotations:
        summary: "Hoher Anteil unbestaetigter Armbander"
        description: >
          "Mehr als 10% der Armbander sind unbestaetigt."
```

Akronyme

API Application Programming Interface. 56, 57, 62

CD Continuous Delivery. 7, 12, 16, 17, 20–22, 33, 34, 38, 39, 55, 64, 67

CI Continuous Integration. 7, 11, 12, 15–17, 20–22, 26, 33, 34, 38, 39, 55, 64, 67, 69, 70, 83, 90, 93

CRD Custom Resource Definition. 61, 62, 85

CVE Common Vulnerabilities and Exposures. 7, 10–12, 22, 27, 35–37, 92, 93

Dev Developer. 15

DevOps Development and Operations. 6, 10, 12, 14–16, 31, 33, 53

DevSecOps Development - Security - Operations. 6, 10–12, 16, 17, 31, 51

DLRG Deutsche Lebens-Rettungs-Gesellschaft. 10, 11, 19, 21, 29, 49

FPM FastCGI Process Manager. 40

GHCR GitHub Container Registry. 7, 31, 37–40

HPA Horizontal Pod Autoscaling. 40, 47, 89

IaC Infrastructure as Code. 16

JSON JavaScript Object Notation. 46

JWT JSON Web Token. 57

MSI Mutation Score Indicator. 7, 36

OCI Open Container Initiative. 37

OIDC OpenID Connect. 63, 64, 68, 69

Ops Operations. 15

ORM Object-Relational Mapping. 14

PDB Pod Disruption Budget. 44, 47, 72

PR Pull Request. 7, 26, 27, 32, 39, 70, 83, 93

PSA Pod Security Admission. 12, 77, 78, 92

PVC Persistent Volume Claim. 44, 47

SBOM Software Bill of Materials. 11, 37, 63, 68–70, 82, 83, 92

SFTP Secure File Transfer Protocol. 6, 22, 24

SLO Service Level Objective. 27, 55, 62, 85, 93

SPoF Single Point of Failure. 25, 49

TLS Transport Layer Security. 40

TTL Time-to-Live. 57, 58

Literaturverzeichnis

[AKNN+12] Jafar M. Al-Kofahi, Hung Viet Nguyen, Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N. Nguyen. Detecting semantic changes in makefile build code. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 150–159, 2012.

[Ama25] Amazon Web Services, Inc. Was ist aws secrets manager? `https://docs.aws.amazon.com/de_de/secretsmanager/latest/userguide/intro.html`, 2025. Zugriff am 05. Oktober 2025.

[Arg25] Argo Rollouts. Canary deployment strategy. `https://argo-rollouts.readthedocs.io/en/stable/features/canary/`, 2025. [Online; Eingesehen am 16. Oktober 2025].

[BH22] Florian Beetz and Simon Harrer. Gitops: The evolution of devops? *IEEE Software*, 39(4):70–75, 2022.

[Bit25] Bitnami Labs. Bitnami labs: Sealed secrets. `https://github.com/bitnami-labs/sealed-secrets`, 2025. Zugriff am 05. Oktober 2025.

[Car22] Carmen Carrión. Kubernetes as a standard container orchestrator – a bibliometric analysis. *Journal of Grid Computing*, 20(4):42, December 2022. Zugriff am 20. Oktober 2025.

[Clo] Cloud Native Computing Foundation. Observability — cloud native glossary. `https://glossary.cncf.io/observability/`. Zugriff am 16. Oktober 2025.

[CNC25] CNCF OpenFeature. Openfeature specification. `https://openfeature.dev/specification/`, 2025. [Online; Eingesehen am 16. Oktober 2025].

[Dat23] Datadog, Inc. 10 insights on real-world container use. `https://www.datadoghq.com/container-report/`, November 2023. Zugriff am 20. Oktober 2025.

[dir25] direnv Docs. direnv. `https://direnv.net/`, 2025. Zugriff am 11. Oktober 2025.

[DLR] DLRG. Kindersucharmband – zwrd-k: Zentraler wasserrettungsdienst küste. https://www.dlrg.de/mitmachen/wasserrettungsdienst/zentraler-wasserrettungsdienst-kueste-zwrd-k/kindersucharmband/. Zugriff am 07. Oktober 2025.

[Doca] Laravel Docs. Blade Templates. https://laravel.com/docs/12.x/blade. [Online; Eingesehen am 16. Oktober 2025].

[Docb] Laravel Docs. Eloquent: Getting Started. https://laravel.com/docs/12.x/eloquent. [Online; Eingesehen am 16. Oktober 2025].

[Docc] Laravel Docs. Laravel Configuration. https://laravel.com/docs/12.x/configuration. [Online; Eingesehen am 16. Oktober 2025].

[Docd] Laravel Docs. Laravel Pennant. https://laravel.com/docs/12.x/pennant. [Online; Eingesehen am 10. Oktober 2025].

[Doce] Laravel Docs. Laravel Sail. https://laravel.com/docs/12.x/sail. [Online; Eingesehen am 23. Juni 2025].

[Docf] Laravel Docs. Routing. https://laravel.com/docs/12.x/routing. [Online; Eingesehen am 16. Oktober 2025].

[Doc25a] Docker Inc. Building best practices. https://docs.docker.com/build/building/best-practices/, 2025. Zugriff am 11.09.2025.

[Doc25b] Docker Inc. Buildkit: High-performance builds. https://docs.docker.com/build/buildkit/, 2025. Zugriff am 11.09.2025.

[Doc25c] Docker Inc. Docker overview. https://docs.docker.com/get-started/overview/, 2025. Zugriff am 11.09.2025.

[Doc25d] Docker Inc. Multi-stage builds. https://docs.docker.com/build/building/multi-stage/, 2025. Zugriff am 13.09.2025.

[Doc25e] Docker Inc. Run multiple processes in a container. https://docs.docker.com/engine/containers/multi-service_container/, 2025. Zugriff am 13.09.2025.

[Doc25f] Docker Inc. View container logs. https://docs.docker.com/engine/logging/, 2025. Zugriff am 11.09.2025.

[Fed] Federal Office for Information Security (BSI). Bsi tr-03183-2: Software bill of materials (sbom). https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03183/BSI-TR-03183-2_v2_1_0.pdf?__blob=publicationFile&v=5. Zugriff am 14.10.2025.

[FHK18] Nicole Forsgren, Jez Humble, and Gene Kim. *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations*. IT Revolution Press, 1st edition, 2018.

[Flu25] FluxCD / Flagger. Flagger — progressive delivery for kubernetes. `https://fluxcd.io/flagger/`, 2025. [Online; Eingesehen am 16. Oktober 2025].

[Fow17] Martin Fowler. Feature toggles (aka feature flags). `https://martinfowler.com/articles/feature-toggles.html`, 2017. Zugriff am 13. September 2025.

[Fow20] Martin Fowler. Feature branch. `https://martinfowler.com/bliki/FeatureBranch.html`, 2020. Zugriff am 13. September 2025.

[Git] Git SCM. Git hooks documentation. `https://git-scm.com/docs/githooks`. Zugriff am 13. September 2025.

[Gooa] Google APIs Authors. Release please documentation. `https://github.com/googleapis/release-please`. Zugriff am 14. Oktober 2025.

[Goob] Google SRE Book (Rob Ewaschuk; ed. Betsy Beyer). Monitoring distributed systems. `https://sre.google/sre-book/monitoring-distributed-systems/`. Zugriff am 16. Oktober 2025.

[Goo25] Google Cloud. Google cloud: Secret manager – Übersicht. `https://cloud.google.com/secret-manager/docs/overview?hl=de`, 2025. Zugriff am 05. Oktober 2025.

[Graa] Grafana Labs. Alertmanager data source — grafana documentation. `https://grafana.com/docs/grafana/latest/datasources/alertmanager/`. Zugriff am 16. Oktober 2025.

[Grab] Grafana Labs. Grafana loki — documentation hub. `https://grafana.com/docs/loki/latest/get-started/overview/`. Zugriff am 16. Oktober 2025.

[Grac] Grafana Labs. Prometheus data source — grafana documentation. `https://grafana.com/docs/grafana/latest/datasources/prometheus/`. Zugriff am 16. Oktober 2025.

[Grad] Grafana Labs. Promtail agent — send data to loki. `https://grafana.com/docs/loki/latest/send-data/promtail/`. Zugriff am 16. Oktober 2025.

[Ham] Hammant, Paul. Trunk based development. `https://trunkbaseddevelopment.com/`. Zugriff am 13. September 2025.

[Hasa] HashiCorp. Database secrets engine. `https://developer.hashicorp.com/vault/docs/secrets/databases`. Zugriff am 16. Oktober 2025.

[Hasb] HashiCorp. Kubernetes – auth methods. `https://developer.hashicorp.com/vault/docs/auth/kubernetes`. Zugriff am 16. Oktober 2025.

[Hasc] HashiCorp. Kv – secrets engines. `https://developer.hashicorp.com/vault/docs/secrets/kv`. Zugriff am 16. Oktober 2025.

[Hasd] HashiCorp. Mysql/mariadb database secrets engine. `https://developer.hashicorp.com/vault/docs/secrets/databases/mysql-maria`. Zugriff am 16. Oktober 2025.

[Hase] HashiCorp. Vault agent injector annotations. `https://developer.hashicorp.com/vault/docs/deploy/kubernetes/injector/annotations`. Zugriff am 16. Oktober 2025.

[Has25] HashiCorp. Understand static and dynamic secrets — getting started with vault. `https://developer.hashicorp.com/vault/tutorials/get-started/understand-static-dynamic-secrets`, 2025. Zugriff am 20. Oktober 2025.

[HF10] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010.

[IH14] Laura Inozemtseva and Reid Holmes. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 435–445, New York, NY, USA, 2014. Association for Computing Machinery.

[Ist] Istio Authors. Istio documentation: Traffic shadowing. `https://istio.io/latest/docs/tasks/traffic-management/mirroring/`. Zugriff am 13. September 2025.

[Jur54] Joseph M. Juran. *Managerial Breakthrough: A New Concept of the Manager's Job*. McGraw-Hill, New York, 1954.

[KHDW16] Gene Kim, Jez Humble, Patrick Debois, and John Willis. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[Kri22] Mikael Krief. *Learning DevOps, Second Edition*. Packt Publishing, 2022.

[Kub23] Kubernetes Authors. Verifying container image signatures within cri runtimes. `https://kubernetes.io/blog/2023/06/29/container-image-signature-verification/`, June 2023. Zugriff am 20. Oktober 2025.

[Kub25a] Kubernetes Authors. Kubernetes concepts: Logging architecture. `https://kubernetes.io/docs/concepts/cluster-administration/logging`, 2025. Zugriff am 11. September 2025.

[Kub25b] Kubernetes Authors. Kubernetes docs: Specifying a pod disruption budget for your application. `https://kubernetes.io/docs/tasks/run-application/configure-pdb/`, 2025. Zugriff am 11. September 2025.

[Kub25c] Kubernetes Ingress-NGINX. Annotations — traffic mirroring. `https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#mirror`, 2025. [Online; Eingesehen am 16. Oktober 2025].

[Lar25a] Laravel. Laravel documentation – artisan console. `https://laravel.com/docs/12.x/artisan#generating-commands`, 2025. Zugriff am 13.09.2025.

[Lar25b] Laravel. Laravel documentation – cache. `https://laravel.com/docs/12.x/cache#main-content`, 2025. Zugriff am 13.09.2025.

[Lar25c] Laravel. Laravel documentation – database: Migrations. `https://laravel.com/docs/12.x/migrations`, 2025. Zugriff am 13.09.2025.

[Lar25d] Laravel. Laravel documentation – database: Seeding. `https://laravel.com/docs/12.x/seeding`, 2025. Zugriff am 13.09.2025.

[Lar25e] Laravel. Laravel documentation – logging. `https://laravel.com/docs/12.x/logging`, 2025. Zugriff am 13.09.2025.

[Lar25f] Laravel. Laravel documentation – queues. `https://laravel.com/docs/12.x/queues`, 2025. Zugriff am 13.09.2025.

[Lar25g] Laravel. Laravel documentation – task scheduling. `https://laravel.com/docs/12.x/scheduling`, 2025. Zugriff am 13.09.2025.

[Mic25] Microsoft Corporation. About azure key vault. `https://learn.microsoft.com/en-us/azure/key-vault/general/overview`, 2025. Zugriff am 05. Oktober 2025.

[MIT25] MITRE Corporation. Common vulnerabilities and exposures (cve). `https://cve.mitre.org/`, 2025. Zugriff am 11.09.2025.

[MLK+23] Marina Moore, Michael Lieberman, John Kjell, James Carnegie, Ben Cotton, and Aditya Sirish A. Yelgundhalli. Software supply chain best practices v2. Technical report, Cloud Native Computing Foundation (CNCF) TAG Security Working Group, 2023. Zugriff am 11. Oktober 2025.

[MNDT09] Audris Mockus, Nachiappan Nagappan, and Trung Dinh-Trong. Test coverage and post-verification defects: A multiple case study. In *Test coverage and post-verification defects: A multiple case study*, pages 291–301, 10 2009.

[Moz25] Mozilla SOPS Maintainers. Mozilla sops: Secrets operations. `https://getsops.io/docs/`, 2025. Zugriff am 20. Oktober 2025.

[Nat23] National Institute of Standards and Technology (NIST). Implementation of devsecops for a microservices-based application using service mesh. Technical Report NIST Special Publication 800-204C, U.S. Department of Commerce, 2023. Zugriff am 11. Oktober 2025.

[NMTA22] Zachary Newman, John Speed Meyers, and Santiago Torres-Arias. Sigstore: Software signing for everybody. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2353–2367, New York, NY, USA, 2022. Association for Computing Machinery.

[OCI25] OCI. Open container initiative: Image specification. `https://github.com/opencontainers/image-spec`, 2025. Zugriff am 11.09.2025.

[OWA25] OWASP Foundation. Owasp cheat sheet series: Secrets management cheat sheet. `https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html`, 2025. Zugriff am 20. Oktober 2025.

[PP24] Luís Prates and Rúben Pereira. Devsecops practices and tools. *International Journal of Information Security*, 24(1):11, Nov 2024.

[PSW+24] Ziyue Pan, Wenbo Shen, Xingkai Wang, Yutian Yang, Rui Chang, Yao Liu, Chengwei Liu, Yang Liu, and Kui Ren. Ambush from all sides: Understanding security threats in open-source software ci/cd pipelines. *IEEE Transactions on Dependable and Secure Computing*, 21(1):403–418, January 2024.

[Red25] Red Hat. What is gitops? `https://www.redhat.com/en/topics/devops/what-is-gitops`, 2025. Zugriff am 11. September 2025.

[Ren] Renovate Authors. Renovate docs. `https://docs.renovatebot.com/`. Zugriff am 14. Oktober 2025.

[SBZ17] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *CoRR*, abs/1703.07019, 2017.

[Seh23] Vandana Verma Sehgal. *Implementing DevSecOps Practices: Understand application security testing and secure coding by integrating SAST and DAST*. Packt Publishing, 2023.

[SH25] Valerie Silverthorne and Stephen Hendrick. Cloud native 2024: Approaching a decade of code, cloud, and change. `https://www.cncf.io/wp-content/uploads/2025/04/cncf_annual_survey24_031225a.pdf`, March 2025. Cloud Native Computing Foundation / The Linux Foundation, Zugriff am 20. Oktober 2025.

[Sig25a] Sigstore Project. Kubernetes policy controller. `https://docs.sigstore.dev/policy-controller/overview/`, 2025. Zugriff am 20. Oktober 2025.

[Sig25b] Sigstore Project. Signing containers – sigstore. `https://docs.sigstore.dev/cosign/signing/signing_with_containers/`, 2025. Zugriff am 14.10.2025.

[SMS24] Sabbir Saleh, Nazim Madhavji, and John Steinbacher. A systematic literature review on continuous integration and deployment (ci/cd) for secure cloud computing. In *Proceedings of the 20th International Conference on Web Information Systems and Technologies*, page 331–341. SCITEPRESS - Science and Technology Publications, 2024.

[Spa25] Spatie BVBA. Spatie laravel prometheus documentation – introduction. `https://spatie.be/docs/laravel-prometheus/v1/introduction`, 2025. Zugriff am 10. Oktober 2025.

[Sta19] Matt Stauffer. *Laravel: Up and Running*. O'Reilly Media, 2019.

[Thea] The Prometheus Authors. Alertmanager — prometheus alerting. `https://prometheus.io/docs/alerting/latest/alertmanager/`. Zugriff am 16. Oktober 2025.

[Theb] The Prometheus Operator Authors. Prometheus operator — api reference. `https://prometheus-operator.dev/docs/api-reference/api/`. Zugriff am 16. Oktober 2025.

[The25a] The Helm Authors. Helm documentation. `https://helm.sh/docs/`, 2025. Zugriff am 11. September 2025.

[The25b] The Helm Authors. Helm documentation — chart hooks. `https://helm.sh/docs/topics/charts_hooks/`, 2025. Zugriff am 11. September 2025.

[The25c] The Helm Authors. Helm documentation – changes since helm 2. `https://helm.sh/docs/faq/changes_since_helm2/`, 2025. Zugriff am 11. September 2025.

[The25d] The Kubernetes Authors. Horizontal pod autoscaler. `https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/`, 2025. Zugriff am 11. September 2025.

[The25e] The Kubernetes Authors. Kubernetes basics: Using a service to expose your app. https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/, 2025. Zugriff am 11. September 2025.

[The25f] The Kubernetes Authors. Kubernetes concepts: Components. https://kubernetes.io/docs/concepts/overview/components/, 2025. Zugriff am 11. September 2025.

[The25g] The Kubernetes Authors. Kubernetes concepts: Configmaps and secrets. https://kubernetes.io/docs/concepts/configuration/secret/, 2025. Zugriff am 11. September 2025.

[The25h] The Kubernetes Authors. Kubernetes concepts: Cronjob. https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/, 2025. Zugriff am 11. September 2025.

[The25i] The Kubernetes Authors. Kubernetes concepts: Custom resources. https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/, 2025. Zugriff am 11. September 2025.

[The25j] The Kubernetes Authors. Kubernetes concepts: Deployments. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/, 2025. Zugriff am 11. September 2025.

[The25k] The Kubernetes Authors. Kubernetes concepts: Ingress. https://kubernetes.io/docs/concepts/services-networking/ingress/, 2025. Zugriff am 11. September 2025.

[The25l] The Kubernetes Authors. Kubernetes concepts: Jobs. https://kubernetes.io/docs/concepts/workloads/controllers/job/, 2025. Zugriff am 11. September 2025.

[The25m] The Kubernetes Authors. Kubernetes concepts: Network policies. https://kubernetes.io/docs/concepts/services-networking/network-policies/, 2025. Zugriff am 11. September 2025.

[The25n] The Kubernetes Authors. Kubernetes concepts: Overview. https://kubernetes.io/docs/concepts/overview/, 2025. Zugriff am 11. September 2025.

[The25o] The Kubernetes Authors. Kubernetes concepts: Persistentvolume. https://kubernetes.io/docs/concepts/storage/persistent-volumes/, 2025. Zugriff am 11. September 2025.

[The25p] The Kubernetes Authors. Kubernetes concepts: Pod security admission. `https://kubernetes.io/docs/concepts/security/pod-security-admission/`, 2025. Zugriff am 11. September 2025.

[The25q] The Kubernetes Authors. Kubernetes concepts: Pods. `https://kubernetes.io/docs/concepts/workloads/pods/`, 2025. Zugriff am 11. September 2025.

[The25r] The Kubernetes Authors. Kubernetes concepts: Replicaset. `https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/`, 2025. Zugriff am 11. September 2025.

[The25s] The Kubernetes Authors. Kubernetes concepts: Resource management for pods and containers. `https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/`, 2025. Zugriff am 11. September 2025.

[The25t] The Kubernetes Authors. Kubernetes concepts: Service accounts. `https://kubernetes.io/docs/concepts/security/service-accounts/`, 2025. Zugriff am 11. September 2025.

[The25u] The Kubernetes Authors. Kubernetes concepts: Services. `https://kubernetes.io/docs/concepts/services-networking/service/`, 2025. Zugriff am 11. September 2025.

[The25v] The Kubernetes Authors. Kubernetes concepts: Statefulsets. `https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/`, 2025. Zugriff am 11. September 2025.

[The25w] The Kubernetes Authors. Kubernetes concepts: Volumes. `https://kubernetes.io/docs/concepts/storage/volumes`, 2025. Zugriff am 11. September 2025.

[The25x] The Kubernetes Authors. Kubernetes docs: Configure a security context for a pod or container. `https://kubernetes.io/docs/tasks/configure-pod-container/security-context/`, 2025. Zugriff am 11. September 2025.

[The25y] The Kubernetes Authors. Kubernetes docs: Configure liveness, readiness and startup probes. `https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/`, 2025. Zugriff am 11. September 2025.

[The25z] The Kubernetes Authors. Kubernetes docs: Performing a rolling update. `https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro`, 2025. Zugriff am 11. September 2025.

[TP20] Phyu Phyu Than and Myat Pwint Phyu. Continuous integration for laravel applications with gitlab. In *Proceedings of the 1st International Conference on Advanced Information Science and System*, AISS '19, New York, NY, USA, 2020. Association for Computing Machinery.

[Tri] Trivy Authors. Trivy official documentation. `https://trivy.dev/v0.67/docs/`. Zugriff am 14. Oktober 2025.

[Wel18] Tim Wellhausen. Expand and contract: A pattern to apply breaking changes to persistent data with zero downtime. `https://www.tim-wellhausen.de/papers/ExpandAndContract.pdf`, 2018. Version 1.0, July 18, 2018.

[Zac] Zachary Rice and contributors. Gitleaks: Protect and discover secrets using gitleaks. `https://github.com/gitleaks/gitleaks`. Zugriff am 14. Oktober 2025.