

Evaluation der Migration von Kubernetes

von

Tim Wall

Mat. Nr. 5014365

Ein Praxisbericht im Rahmen der Praxisphase
an der htw saar im Studiengang Praktische Informatik
in Kooperation mit der psb intralogistics GmbH

Pirmasens, den 22. Juni 2026

Inhaltsverzeichnis

1	Betrieb	1
2	Arbeitsplatz	2
3	Aufgabenbereich	2
4	Methodik	4
5	Aufgabenbearbeitung	4
5.1	Distribution	4
5.2	Erste Berührungspunkte	5
5.3	Helm	6
5.4	Operatoren	6
5.5	GatewayAPI	6
5.6	selektron	7
5.7	Nutzeroberfläche	8
5.8	Windows	8
5.9	GitOps & FluxCD	9
5.10	High Availability, Longhorn und CloudNativePG	10
5.11	Cert-Manager	11
5.12	Observability	12
5.13	External Secrets Operator	12
5.14	Airgap-Installation	12
6	Evaluation	13
7	Fazit	14

Vorwort

Die Praxisphase ist ein Zeitraum innerhalb des Studiums, um eine Brücke zwischen dem theoretischen Unterricht und der praktischen Umsetzung in der Realität zu bilden. Planmäßig findet diese im sechsten Semester in den ersten 12 Wochen statt. Meine Praxisphase begann am 01.04 bei der psb intralogistics GmbH in Pirmasens, und dauert bis zum 30.06.

Innerhalb diesem Zeitraum wurden mir Einblicke in die Unternehmensstruktur, deren Softwareschöpfungsprozess und eine eigene Projektarbeit gewährt. Auch werden extracurriculare Inhalte und Soft Skills vermittelt. Somit wird fachliche, sowie persönliche Entwicklung gefördert.

Der Praxisbericht ist eines der resultierenden Elemente, in welchem das Unternehmen, der Arbeitsplatz, dessen Stellung, durchgeführte Aufgaben, gewonnene Erkenntnisse und die unterliegende theoretische Basis dokumentiert, evaluiert und bewertet wird.

1 Betrieb

Die psb intralogistics GmbH mit Sitz in Pirmasens ist ein traditionsreiches, mittelständisches Familienunternehmen, das 1887 als Schlosserei gegründet wurde. Heute zählt es zu den führenden europäischen Anbietern von automatisierten Intralogistik-Gesamtsystemen.

Das Unternehmen zeichnet sich durch eine sehr hohe vertikale Integration aus: Von der ersten Planung über die mechanische Produktion und Softwareentwicklung bis hin zur Montage und Wartung erfolgt alles „aus ei(ge)ner Hand“ [10]. Bemerkenswert ist dabei, dass alle Fertigungsschritte in Pirmasens ablaufen, wodurch maximale Kontrolle und Qualitätssicherung gewährleistet werden kann. Das Unternehmen beschäftigt knapp unter 550 Mitarbeiter, darunter Ingenieure, Softwareentwickler, SPS-Programmierer, Fertigung, Montage, Materialbeschaffung, Logistik, Vertrieb, IP-Verwaltung (geistiges Eigentum) sowie Managementebene.

psb ist ein familiengeführtes Unternehmen, aktuell in der 4. Generation durch Werner Klein. Als mittelständisches Unternehmen wird auf kurze Entscheidungswege und ein Patensystem für neue Mitarbeiter gesetzt, um Know-how schnell zu vermitteln und Wissenssilos zu vermeiden.

Lösungen sind auf jeden Kunden maßgeschneidert und verbinden Lagerung, Transport, Kommissionierung, Sortierung und Software zu einem einheitlichen Komplettsystem.

Je nach Größe der zu lagernden Elemente und des gewünschten Systems bietet psb vorgefertigte Baukastensysteme, die individuell zugeschnitten werden. Als klassische Regalbediengeräte werden der *sprinter*, gedacht für leichte Kartons, Kleinbehälter und Tablare, der *runloader*, gedacht für Kassetten oder Hängeware bis 300 kg Zuladung und der *maxloader*, gedacht für Paletten bis 1250 kg Zuladung, angeboten. Aber auch Shuttlesysteme wie der *vario.sprinter*, Paletten-Shuttles oder Mischsysteme wie Hängewaren-Shuttles werden zum Lagern angeboten.

Der Transport wird durch Rollbahnen, Mehrstrangförderern und einer breiten Auswahl an Verbindungsstücken realisiert. Automatische Kommissionierung wird von *autopick*, einem KI-geführten, mit Vakuumsaugern ausgestatteten Roboterarm übernommen. Manuelle Kommissionierung erfolgt auftragsgerecht sequenziert durch *rotapick*-Systeme, im Gleichteilüberfluss durch Multi-Order-Stationen, Auftragsbasiert in 1-zu-1-Kommissionierung oder Flächendeckend mit Durchlaufregalen.

Die Orchestration der zuvor genannten Systeme und Module erfolgt durch eine in-house entwickelte Software-Suite mit dem Namen *selektron*, bestehend aus Warehouse Management System (WMS), Materialflusssteuerung (MFC), Anlagensteuerung (FLC), Visualisierung (SCADA), Produktionssteuerung und Anbindung an bestehende Kundensysteme, z.B. Enterprise Resource Planning (ERP), Produktionsplanung (PPS), Inventurverwaltung (WWS). Diese Suite wird mit einem 24/365 Service abgerundet, welcher durch Fernwartung schnelle Störungsbeseitigung ermöglicht.

Im globalen Intralogistik-Markt hebt sich psb gegenüber anderen Anbietern mit dem hohen Manufakturgrad, und maximaler vertikaler Integration und persönlicher Kundebetreuung ab.

2 Arbeitsplatz

Das Firmengelände besteht aus einem Verwaltungsgebäude und zwei Fabrikhallen. Ein weiteres Gebäude, das Zentrallager, befindet sich aktuell im Bau. Das dreistöckige Verwaltungsgebäude besitzt im Erdgeschoss den Kundenempfang, Human Resources, SPS-Entwicklung, Entwurf des Firmeninternen Verwaltungssystems UCM als auch Gesprächsräume für Kundentermine. Der erste Stock besteht aus dem IT-Bereich und der Projektabwicklung. Innerhalb der Projektabwicklung wird zwischen Steuerung, SCADA, Fördertechnik, Hängefördertechnik, Lagertechnik, Mechanischer Konstruktion, Elektrischer Konstruktion, Produktdesign, Service, Projektmanager, Dokumentation und den Software-Ingenieuren bestehend aus Warehouse Management, Material Flow, Nutzeroberfläche, Softwarewartung, unterschieden. Das oberste Stockwerk ist für den Vertrieb, Betriebsrat und Management vorgesehen.

Die Sitzplätze des Großraumbüros sind nach Teams mithilfe von Trennwänden gruppiert, um Kommunikationswege kurz zu halten, den Austausch zu fördern, jedoch bei Gesprächen andere Gruppen nicht zu stören, und bestehen aus jeweils vier bis fünf Leuten. Die Leute innerhalb eines Teams behandeln die gleichen Kundenprojekte. Diese „Sitzordnung“ ist trotzdem nicht bindend und jeglich eine Empfehlung, um Kommunikation zwischen den Gruppen nicht zu blockieren. Die Gruppe Projektabwicklung Software-Engineering Graphical User Interface, welcher ich untergewiesen bin, ist auf 4 Teams verteilt.

Das Software-Engineering Department befindet sich im ersten Stock auf der Südseite, und ist mit einer Feuerschutztür von den anderen Projektabwicklungs-Abteilungen physisch getrennt. Beim Betreten der Abteilung fällt zunächst der Blick auf die Meetingräume, gefolgt von der angrenzenden Kücheneinheit. Direkt gegenüber befindet sich mein zugewiesener Arbeitsplatz, welcher mit einer Docking-Station meinen Arbeitslaptop mit Bildschirmen und Peripherie verbindet. Meine direkten Nachbarn befassen sich mit dem gleichen Themengebiet meiner Praxisphase und ermöglichen durch physische Nähe schnelle, effiziente, E-Mail unabhängige Kollaboration und Problemlösung, besonders bei Trivialitäten.

3 Aufgabenbereich

Mein Arbeitsplatz befindet sich im Zentrum des Software-Engineering-Department, der GUI-Abteilung. Während der Name auf alleiniges Ausarbeiten von grafischen Nutzeroberflächen (Graphical User Interface, kurz GUI) vermuten lässt, ist diese Abteilung auch für die Integration der verschiedenen Backends von selektron zuständig. Dies wird mithilfe des in der Abteilung entwickelten Kernmodul der selektron-Suite, selektron core, bewältigt. selektron core besteht dabei aus den Komponenten Administration, API Gateway, Authentication, Authorisation und Discovery. Jedes der Backends besteht ebenfalls aus mehreren Komponenten. Diese werden jeweils für Kundensysteme vom Basis-Entwicklungsstand einzeln angepasst und zugeschnitten, damit es perfekt auf deren Anforderungen ausgelegt ist. Oft haben Kunden jedoch

gleiche oder ähnliche Bedürfnisse, welche im aktuellen System bei jedem Kunden extra und damit mehrfach Implementiert werden. Diese Implementationen betreiben oft die gleiche Aufgabe, sind jedoch anders Entwickelt. Durch diese resultierenden Redundanzen steigt der Wartungsaufwand. Ebenfalls steigt der Aufwand, die zugeschnittenen Systeme bei dem Kunden zu konfigurieren und einzusetzen, weil jeder Kunde eine einzigartige IT-Infrastruktur hat, wodurch das Deployment händisch mit einer Ansammlung von über die Jahre entwickelten und gewachsenen Skripten und viel Filigranarbeit durchzuführen ist. Dieses händische Softwaredeployment ist nicht nur aufwändig, sondern senkt dazu die Wartbarkeit, da über die Jahre immer wieder Updates, Migrationen und sonstige kurzfristige Änderungen auftreten, welche nur zum Teil dokumentiert, und damit schwer Nachvollziehbar sowie absolut nicht Auditierbar sind. Folgend ist die Einarbeitung von Servicemitarbeitern aufwändig, da alles über das Terminal mit Skripten und Befehlen verwaltet werden muss, und nicht alle Fälle von diesen existierenden Skripten übernommen werden können. Diese Servicefälle müssen im Schlimmstfall bis zum nächsten Arbeitstag warten, was für den Kunden verlorene Zeit und damit verlorenes Geld bedeutet. Würde im aktuellen System eine Ausfallsicherheit existieren, könnte diese zumindest den Service bei einigen trivialen Fällen entlasten, jedoch ist aktuell keine vorhanden.

Jedes der Module der selektron-Suite wurde innerhalb der letzten Jahre in Container-Images gebündelt und daraufhin das Deployment mit Containerverwaltungstools wie Docker oder Podman durchgeführt, um das Deployment weitestgehend zu vereinfachen und unabhängiger von der Kundeninfrastruktur zu gestalten. Trotzdem gehen viele der genannten Probleme über die simple Containerisierung von Software hinaus, und bleiben somit bestehen.

Obwohl das Evaluieren von Kubernetes als Containerorchestrierung für den Software-Stack der psb keine klassische Aufgabe im Bezug auf Nutzeroberflächen darstellt, ist die Aufgabe in der Firmenstruktur dieser untergliedert, da es als Integration von Systemen gilt. Kubernetes führt im Kern, ähnlich wie Docker und Podman, Container aus, geht jedoch weit über das Ausführen und triviale Verwalten von diesen hinaus und wird daher aus als Orchestration bezeichnet. In Anbetracht, dass die Firma aktuell noch keine Forschung über Kubernetes, außer dem Fakt, dass es existiert und die aktuelle Problemlage lösen könnte, betrieben hat, sollte dies meine Aufgabe werden.

Das Ziel ist es, herauszufinden, welche Features Kubernetes und dessen Ökosystem bietet, zu analysieren wie diese funktionieren, als auch zu überprüfen, ob diese konkret für das Projekt Sinn ergeben. Mithilfe der Erkenntnisse soll Basiswissen über die Containerorchestration sowie ein möglichst automatisch deploybares Walking Skeleton auf Grundlage des selektron-Basisprojektes entstehen. Darüber hinaus soll geprüft werden, inwieweit Kubernetes eine Integration von Windows-basierten Systemen ermöglicht und welche technischen Einschränkungen dabei bestehen. Die gewonnenen Erkenntnisse dienen als Entscheidungsgrundlage für eine mögliche Migration der bestehenden Systemarchitektur hin zu einer Kubernetes-basierten Lösung.

4 Methodik

Während der gesamten Praxisphase wurde, soweit realisierbar, ein iterativer Zyklus verfolgt. Dieser beginnt initial mit der Discovery-Phase, in welcher neue Produkte identifiziert werden, die zu den gegebenen Zielen passen. Anschließend werden sie in einem Backlog festgehalten. Dieser wird in Absprache mit den Teamkollegen geordnet.

Sofern ein Produkt die Spitze des Backlogs erreicht, werden Eigenschaften über das Produkt und dessen Funktionsweise ermittelt. Sofern die Ziele des Produktes mit den Zielen des Betriebes übereinstimmen, beginnt die Evaluationsphase, in welcher das Produkt in einer isolierten Umgebung ausprobiert wird, um Funktionsweisen und Charakteristiken zu observieren. Diese werden im firmeninternen Confluence-Board, einer kollaborativen Knowledgebase von Atlassian, dokumentiert.

Täglich wird der aktuelle Stand dem Fachbetreuer mitgeteilt. Wöchentlich gibt es ein Meeting mit dem Fachbetreuer, dem organisatorischen Betreuer, meinem Gruppenleiter und weiteren Stakeholdern, um sich über ausprobierte Produkte, deren Eigenschaften und mögliche Integration in den Stack zu besprechen. Sofern Alternativen zu den Produkt bestehen erfolgt ein Vergleich mit diesen. Somit wird Techstack-Creep, welches mit übermäßiger Komplexität, Ineffizienzen und Sicherheitsrisiken verbunden ist, verhindert. Gleichzeitig wird garantiert, dass wertschöpfende Produkte nicht ausgelassen werden. Bei grünem Licht wird das Produkt in die bestehende Testlandschaft integriert und weitere Optimierungen und Anpassungen für selektron erarbeitet.

5 Aufgabenbearbeitung

Nachdem Aufgabenstellung und Methodik geklärt sind, folgt nun die Bearbeitung der Aufgabe. Diese beschreibt den Hauptteil meiner Zeit der Praxisphase.

5.1 Distribution

Initial wurde Forschung über die Funktionsweise von Kubernetes unternommen, da weder meine Betreuer, noch ich Wissen in diesem Bereich hatten. Dabei fiel zuerst Augenmerk darauf, dass es im Gegensatz zu Docker oder Podman nicht „ein“ Kubernetes gibt, sondern verschiedene Distributionen[21], ähnlich wie bei Linux. In Anbetracht dessen, dass die Logikcontroller der psb direkt im Lager On-Premise laufen, lassen sich schnell viele der Cloud-basierten Lösungen wie AWS EKS, Azure AKS oder Google GKE wegfiltern. Übrig bleiben On-Premise oder Edge-Computing Distributionen. Diese unterscheiden sich, auch ähnlich wie Linux-Distributionen, weniger darin, welche Software laufen kann, sondern in dem Standardumfang und bestimmten Unique Selling Points, wie einem einfachen Installationsprozess oder hohem Compliance-Grad out-of-the-box. Kandidaten sind Rancher k3s, eine Edge-Compute-Distribution mit Fokus auf Hauptspeichereffizienz auf

Kosten einiger High-Availability-Funktionen, welche mit einem Befehl installierbar ist[22], Rancher RKE2, einer Distribution mit Fokus auf hohen Sicherheitsstandards, Compliance und Enterprise-Support[25], RedHat OpenShift, eine allumfassende Plattform mit Kubernetes im Kern welche GitOps, CI/CD Pipelines, Infrastrukturmanagement, Monitoring und Enterprise-Support umfasst[12], OKD, die Communitygetriebene Version von OpenShift ohne Enterprise-Support[11] und zuletzt kubeadm[18], welches den minimalen Kern von Kubernetes deployt und den Rest der Konfiguration dem Nutzer überlässt. Für das Evaluieren wurde entschlossen, eine möglichst Originalgetreue Distribution von Kubernetes zu benutzen, wodurch OpenShift und OKD mit ihren meinungsstarken Standardkonfigurationen sowie großen Zusatzkomponenten wegfallen und die Entwicklungen von Rancher als auch kubeadm übrig blieben. Da k3s mit einfacher Installation, besonders im Gegensatz zu kubeadm, überzeugt, wurde es als initiale Technische Grundlage ausgewählt, besonders mit dem Hintergedanken, dass für den Kubernetes-API-Kern geschriebene Komponenten auf jeder Distribution lauffähig sind. Diese Portabilität soll später ebenfalls erlauben, weitaus Kundensystem-agnostischer zu arbeiten. Im Verlauf wurde kubeadm sowie RKE2 geprüft. Falls die Migration zustande kommt, wird RKE2 als Produktionsumgebung mit Möglichkeit für High-Availability die Basis bilden.

5.2 Erste Berührungspunkte

Nach der Wahl der Distribution folgte das Einarbeiten in Kubernetes. Dabei wurde mit den kompletten Grundlagen begonnen. Das *Cluster*, welches ein vollständiges Kubernetes-System mit allen Komponenten beschreibt. Die *Node*, welche einen einzelnen Rechner bzw. Server in einem Cluster abbildet, und das *kubelet*, welches die Node mit dem Cluster verbindet. Den *Pod*, der Basiseinheit in Kubernetes, bestehend aus Containern (Softwaremodulen), Speicheranbindungen und Rechten. *Deployments*, welche den Lebenszyklus von einer Gruppe gleicher Pods verwalten, darunter Start, Neustart und Updates. *Services*, welche Pods feste Adressen zuweisen, damit andere Pods diese ansprechen können. Verwaltet wird ein Cluster von der dazugehörigen *Control-Plane*, bestehend aus *kube-apiserver*, welcher Befehle annimmt und an das Cluster überträgt, *kube-scheduler*, welcher Pods auf Nodes zuweist, *etcd*, welches alle Konfigurationen des Clusters speichert, *kube-controller-manager*, welcher die Hauptlogik von Kubernetes übernimmt und den *cloud-controller-manager*, welcher das Cluster in der Cloud-Infrastrukturmgebung widerspiegelt, sofern Verfügbar. Kommunikation mit dem Cluster verläuft über das Kommandozeilentool *kubectl*. Damit diese und weitere Informationen an einer zentralen Stelle für den späteren Gebrauch auffindbar sind, wurde ein Nachschlagewerk geschrieben und gepflegt, welches im Rahmen der Praxisphase weitere Ausarbeitungen erhalten hat.

Nach dem initialen Einarbeiten wurde der Aufgabe nachgegangen, das Monitoring-Tool Graylog, welches selektron benutzt, um alle Logs der verschiedenen Services zusammenzuführen, auf Kubernetes zu portieren, um praktische Erfahrung zu sammeln.

5.3 Helm

Schnell wurde klar, das Kubernetes Komponenten manuell verwalten Limitationen aufweist. Ein Hauptgrund ist, dass der Befehl, um Komponenten aus einer Deklarationsdatei anzuwenden, sie hinzufügen, aber nicht löschen kann, und man somit aus der Deklarationsdatei entfernte Komponenten selbst zu löschen hat. Um diese Limitation zu umgehen, wurde zeitnah der Paketmanager Helm[14] benutzt. Pakete sind in Form von *Charts* realisiert, welche konkret Sammlungen an Template-Deklarationsdateien darstellen, die mit einer Values-Datei zu vollwertigen Komponentendeklarationsdateien zusammengesetzt werden, um als *Release* auf das Cluster gespielt zu werden. Dies hat den Vorteil, dass Helm-Charts mit einem Befehl installiert, geupgraded und deinstalliert werden können und eine Versionierung zwischen den Releases existiert die Rollbacks ermöglicht. Auch können Upgrades *atomic* durchgeführt werden, das heißt dass im Falle einer Fehlgeschlagenen Komponente alles auf den vorherigen Zustand zurückgesetzt wird. Glücklicherweise pflegen die Entwickler von Graylog ebenfalls einen Helm-Chart, wodurch ich von meiner spartanischen Erstimplementation auf eine allumfassend bessere Version wechseln konnte. Bei der Installation hatte ich gleichzeitig das erste mal Kontakt mit Operatoren, konkret dem vom Graylog-Helm-Chart genutzten MongoDB-Operator.

5.4 Operatoren

Operatoren bilden einen der Hauptwege ab, Kubernetes zu erweitern. Sie bestehen aus *Custom-Resource-Definitions* (kurz CRDs), mit welchen ein Wunschzustand definiert werden kann, und mindestens einem Pod, welcher Instanzen der *Custom Resources* (CRs) überwacht, und daraufhin das Cluster entsprechend bedient. Der MongoDB-Operator als Beispiel überwacht das Cluster nach MongoDB-Komponenten, in welchen eine Datenbank definiert wird, und erstellt daraufhin dynamisch die Datenbank nach den in der CR angegebenen Anforderungen. Falls die CR-Komponente editiert oder gelöscht wird, wird die dadurch erstellte Datenbank dementsprechend editiert oder gelöscht. Somit Verwaltet der Operator den Lebenszyklus der angefragten Resource, wodurch sich der anfragende Service (in dem Fall der Graylog-Helm-Chart) nicht um die genaue Implementation, sondern nur den Finalzustand kümmern muss. Dieser Ansatz wird „Infrastructure as Code“ bzw. „Configuration as Code“ genannt, und zieht sich durch Kubernetes und das dazugehörige Ökosystem durch. In Anbetracht dessen, dass Infrastruktur (in diesem Fall Datenbanken) mithilfe von Code statt manueller Prozesse bereitgestellt wird, und somit Konsistenz, Auditierbarkeit, Transparenz und Skalierbarkeit gewährt wird, finde ich diesen Ansatz zukunftsorientiert.

5.5 GatewayAPI

Nachdem Graylog auf dem Cluster lief, war es jedoch noch nicht von extern erreichbar. Dafür ist *Ingress* zuständig. Ingress ist ein Spec, verwaltet von dem Kubernetes-Team[17], welcher Layer 7 HTTP/S Routing verwaltet. HTTP/S ist das gleiche Pro-

tokoll, das ein Webbrowser verwendet um Webseiten zu öffnen. Implementationen wie nginx oder Traefik implementieren diesen Spec, und Stellen Controller bereit, welche das konkrete Routing verwalten. Da selektron jedoch nicht nur Layer 7, sondern auch für Kundenspezifische Lösungen Layer 4 Routing benötigt, wurde auf Ingress' Nachfolger, *GatewayAPI* gesetzt[23]. GatewayAPI ist genau wie Ingress ein Spec, welcher von externen Implementationen mithilfe von Controllern verwaltet wird. Dabei wurde aus den Defiziten von Ingress gelernt, woraufhin *Routen*, *Gateways* und *GatewayClasses* voneinander getrennt werden, um stärkere Trennung von Angelegenheiten (Seperation of Concerns) zu realisieren. Dazu unterstützt GatewayAPI nicht nur HTTP/S, sondern auch gRPC, TCP und UDP-Verbindungen. Zusätzlich kann mithilfe von GatewayAPI neben Nord-Süd-Datenverkehr (Nutzer mit Cluster) auch Ost-West-Datenverkehr (Pod zu Pod) geroutet werden.

Da Graylogs Chart aktuell nativ kein GatewayAPI unterstützt, wurde außerhalb des Charts die Notwendige HTTPRoute deklariert. Um abschließend Graylog mit einer externen Startroutine zu konfigurieren, wurde ein Kubernetes Job geschrieben, welcher alle notwendigen Eingabequellen mit passendem Port einrichtet.

5.6 selektron

Nachdem Graylog auf dem Cluster lief und ein initiales Verständnis für Kubernetes, Helm und GatewayAPI existierte, wurde sich der Portierung selektron auf einen Helm-Chart gewidmet. Als Grundlage für die Portierung wurde die Konfiguration eines Kundensystems übernommen. Während die initiale Portierung schnell verlief, gab es eine Liste an Fehlern zu bewältigen. Da der selektron-Chart in einen anderen *Namespace* als Graylog deployed wird, war Kommunikation zwischen den beiden Applikationen nicht möglich. Namespaces sind logisch getrennte Abschnitte innerhalb eines Clusters, um deren Inhalte nicht zu vermischen. Ein *ExternalName*-Service wurde genutzt, um die Brücke zwischen den Namespaces zu bilden. Ebenfalls hat der Discovery- und Load-Balancing-Container von selektron, basierend auf Spring Boots Netflix Eureka, mit der Kubernetes Service-Schicht nicht kooperiert, da diese Domain-Namen und Container trennt, Eureka jedoch erwartet, dass jeder Container einen eigenen Domain-Namen hat. Glücklicherweise erlaubt Eureka für eine Entkopplung der beiden Komponenten. Grundsätzlich ist jedoch Eureka im Falle der Portierung von selektron auf Kubernetes nichtmehr von Nöten, da der komplette Aufgabenbereich von Kubernetes' Service-Schicht bereits übernommen wird. Da es sich um einen Prototypen handelt, bleibt dieses Modul jedoch bestehen.

Generell ist der selektron-Chart Modular aufgebaut und erlaubt in einer zentralen Datei das Ändern aller relevanten Parametern. Durch die hohe Konfigurabilität und Helms Unterstützung von *Overlays*, partiellen Konfigurationsdateien, welche mit der Kern-Konfiguration zusammengeführt werden, um Wiederholungen zu vermeiden, können Änderungen einfach und gezielt vorgenommen werden.

5.7 Nutzeroberfläche

Eine weitere Anforderung ist eine Verwaltungsübersicht für das Kubernetes-Cluster, da Kommandozeilenwerkzeuge wie `kubectl` und `helm` zwar alle Funktionalitäten von Kubernetes abdecken, sich jedoch Personalschulung für den Support als aufwändig erweist und der Aufwand für simple und repetetive Aufgaben größer ist als notwendig. Als Haupt-Visualisierung wurde sich auf *Headlamp*[24], den offiziellen Nachfolger des Kubernetes-Dashboards[16] geeignet. Zusatz-Visualisierungen für Entwickler und Cluster-Operatoren bilden *Radar*[15], ein modulares Dashboard als Ersatz für Headlamp, und *k9s*[19], ein grafisches Kommandozeilentool. *k9s* ist dabei besonders auf Geschwindigkeit und Tastaturbasiertes bedienen getrimmt, vergleichbar mit dem Text-Editor `vim`, und umfasst eher Aufgaben im Bereich des Developments von einzelnen Komponenten auf dem Cluster (Komponenten anzeigen & editieren, Logs durchforsten). Headlamp bringt dagegen mehr Bedienkomfort, eine Standardmäßige Auflistung aller existierenden Kubernetes-Komponenten und CRDs (im Gegensatz zu *k9s*' Such-basiertes Modell zur Komponentenauswahl), eine Map, welche existierende Komponenten in einem Graph miteinander verbindet um sie Logisch zu gruppieren, sowie Plugin-Support. Radar bringt ähnlich wie Headlamp hohen Bedienkomfort und von sich aus Module für klassische Cluster-Erweiterungen wie Helm, FluxCD, ArgoCD und GatewayAPI, ist jedoch mit seinen Features eher auf Infrastrukturbereitsteller und DevSecOps als auf Anwendungsentwicklung ausgelegt. Da alle der genannten Übersichten auf Basis von *Kubeconfigs* mit dem Cluster kommunizieren, sind sie sofern `kubectl` eingerichtet ist, ohne weiteren Konfigurationsaufwand benutzbar. Kubeconfig beschreibt dabei den offiziellen Weg, Authentifikationsdaten innerhalb einer Datei zu speichern, um sich gegenüber einem Cluster zu Authentifizieren. Besonders Headlamp hat für Überzeugung im Team gesorgt, da es auch ohne Kubernetes-Grundlagen vergleichsweise sehr einfach zu bedienen ist, und man mögliche Defizite mithilfe von Plugins ausmerzen kann. Klassische Serviceaufgaben, wie das Neustarten und Stoppen von Containern, oder das Ausführen von Kommandos innerhalb eines Containers, können somit über eine nutzerfreundliche Oberfläche mit wenigen Clicks realisiert werden, anstatt auf Skripte die auf einem Server versteckt liegen, zurückgreifen zu müssen.

5.8 Windows

Kernbestandteil der mir initial gegebenen Anforderungsliste war die Evaluation der Nutzbarkeit von Kubernetes auf Windows-basierten Systemen. Während Kubernetes es unterstützt, Worker-Nodes auf Windows laufen zu lassen, gibt es keine Control-Plane die für Windows konzipiert ist, da Windows im Umfeld der Containerisierung keine Vorteile gegenüber Linux bietet, und der Mehraufwand nicht rechtfertigbar ist. Die Problemstellung lässt sich trotzdem lösen, da der Windows-Kernel seit Windows 10 bzw. Windows-Server 2016 *Hyper-V* mitliefert[20]. *Hyper-V* ist Microsofts Type-1 (Bare-Metal) *Hypervisor*, vergleichbar mit Linux' KVM. Hypervisor sind Software, die es erlauben, auf Host-Systemen mehrere Virtuelle Maschinen (VMs), sogenannte Guest-Systeme, laufen zu lassen. Dabei unterscheidet man zwischen Type-1 und Type-2 Hypervisoren. Der Vorteil von Type-1 Hypervi-

sorn ist, dass diese parallel zum Betriebssystem laufen und daher unabhängig von dem Geschehen auf dem Host-Systemen sind, während Type-2 Hypervisor auf dem Host-Betriebssystem laufen. Damit sind Type-2 Hypervisor trivialer zu installieren und bedienen, jedoch langsamer. Der Clou ist, mithilfe von Hyper-V eine Linux-VM aufzusetzen auf welcher das Cluster läuft. Nachteil bei dieser Konfiguration ist, dass durch die Virtualisierung von Hardware ein Performance-Verlust, sogenannter *Overhead*, entsteht. Dieser ist besonders in I/O-Operationen, also dem Lesen und Schreiben von Daten, spürbar. Auch ist Networking innerhalb Hyper-V sehr fragil, sofern keine Hyper-V External Switches verwendet werden. Diese müssen jedoch innerhalb der Kundenfirewall als externes Gerät eingerichtet werden, was die Kundentoleranz maßgeblich senkt. Da ein nicht zu unterschätzender Anteil der Kunden auf Windows-basierte Serversysteme setzt, könnte dies eine große Einschränkung bei der vollständigen Migration von selektron erweisen. Somit bleibt die Hyper-V Lösung eine Zwischenlösung.

5.9 GitOps & FluxCD

Drift beschreibt Änderungen an der Server-Konfiguration, welche undokumentiert sind, ähnlich wie bei der Bootsahrt, bei der ungeplante Änderungen vom Kurs als Drift bezeichnet werden. Um zu verhindern, dass Drift entsteht, benötigt man eine Wahrheitsquelle. Das Konzept von *GitOps* verfolgt den Ansatz, eine Git-Repository als Wahrheitsquelle zu definieren. Diese Git-Repository beschreibt den Wunschzustand der gesamten Infrastruktur. Operatoren auf dem Cluster synchronisieren die laufende Umgebung in definierten Abständen mit diesem Wunschzustand. Dieser Vorgang ist dabei Teil des *CI/CD-Prozess* (Continuous Integration, Continuous Delivery), konkret übernimmt es den Delivery-Aspekt. Anders als bei einer klassischen CD-Pipeline, bei der Push-basiert von einem Build-Server auf den Deployment-Server (hier das Cluster) deployt wird, wird bei der GitOps-Pipeline Pull-basiert von einem auf dem Deployment-Server laufenden Operator intervallbasiert die Git-Repository nach Änderungen überwacht, und im Falle einer Differenz zwischen Repository-Zustand und Server-Zustand die notwendigen Änderungen vorgenommen. Diesen Prozess nennt man *Reconciliation*. Somit sind Serverzustand und Git-Zustand synchronisiert, Änderungen am Server auditierbar (Wer hat etwas geändert), differenzierbar (Was wurde genau geändert), rollbackbar im Falle eines Problems und reproduzierbar. Dabei wurde sich für *FluxCD*[9] entschieden, eine der GitOps-Implementationen für Kubernetes. Eine Mögliche Alternative ist *ArgoCD*, welches von Haus aus eine Nutzeroberfläche und weitere fortgeschrittene Features bietet, jedoch ist FluxCD weniger ressourcenintensiv, nativer in das Kubernetes-Tooling verzahnt und integriert mit einem existierenden Headlamp-Plugin[13] nativ in die gewünschte Übersicht. Dabei hat FluxCD zwei für selektron relevante Pipelines: die GitOps-Pipeline und die Image-Controller-Pipeline.

Die *GitOps Pipeline* besteht aus `source-controller`, `kustomize-controller` und `helm-controller`.

Der `source-controller`[7] überprüft in gegebenem Intervall angegebene Quellen Git-, Helm- oder OCI-Repositories und speichert sie als Cluster-lokales Artefakt.

Der `kustomize-controller`[6] sucht nach auf dem Cluster existierenden Kompo-

nenten des Typs `Kustomization`, einer CR von FluxCD. In Abständen die in der Komponente definiert sind, führt der Controller auf dem gegebenen Pfad einen *Kustomize*-Befehl aus, produziert, ähnlich wie Helm, Komponentendeklarationsdateien, vergleicht diese mit dem aktuellen Stand, und wendet sie bei Änderungen an. Damit wird nach jedem Intervall jeglicher Drift in den Konfigurationsdateien überschrieben. Die Kustomization-Komponente kann mit Parameter `prune` alle unter das Kustomization fallenden nichtmehr existierenden Komponenten automatisch aufräumen. Mit weiteren Parametern ist es möglich, die Kustomization so zu gestalten, dass alle Komponenten bereit sein müssen, bevor die Kustomization selbst als Bereit gilt. Somit fallen Fehlkonfigurationen schneller auf.

Der `helm-controller`[4] greift analog zum Kustomize-Controller bei `HelmReleases` statt `Kustomizations`, und setzt dabei Helm-Charts automatisch mit den gegebenen Values auf.

Die *Image-Automation-Pipeline*[5] besteht aus dem `image-reflector-controller`, welcher Container-Images überwacht und deren Metadaten speichert. Er wertet mit nutzerdefinierten Regeln die gespeicherten Metadaten aus, und entscheidet welches das zu benutzende Image ist. Mithilfe des `image-automation-controllers` und einer `ImageUpdateAutomation`-Komponente wird in einer Angegebenen Git-Repository der zu benutzende Tag committed. Somit übernimmt im Falle von Image-Updates FluxCD automatisch die Migration.

Mithilfe der beiden Pipelines kann FluxCD einen massiven Teil des Deployments automatisieren und gleichzeitig die Synchronisation der Wunschkonfiguration aus einer gewählten Quelle und dem Clusterstatus übernehmen, um undokumentierte Zugriffe zurückzusetzen und einen höheren Transparenzgrad zu gewährleisten.

5.10 High Availability, Longhorn und CloudNativePG

Viele der Kunden sind zunehmend an Ausfallsicherheit interessiert. Sofern eine Distribution ein High-Availability Control-Plane zur Verfügung stellt, ist dies für das Cluster realisierbar. Für *Stateless Workloads*, also Software die keine Daten zwischenspeichert, ist die Realisierung trivial: die Container müssen auf mehrere Nodes verteilt werden, und im Falle eines Ausfalls realisiert Kubernetes durch Health-Checks dass der Workload nichtmehr Aufnahmefähig ist und verschiebt die Anfragen. Bei *Stateful Workloads* wie Datenbanken ist der Prozess komplexer. Da bei einem Ausfall des Workloads oder der Node die an den Workload gebundenen Daten nichtmehr aufrufbar sind dürfen die Daten nicht nur an einem lokalen Punkt gespeichert werden. Dafür existieren *Distributed Storage Provider*. Ein Distributed Storage Provider kümmert sich um die Replizierung von Daten innerhalb eines Clusters nach vorgeschriebenen Regeln, Dokumentiert in `StorageClass`-Komponenten. *Longhorn* ist ein einfach zu bedienender Distributed-Storage Provider, welcher die Replikation der Daten auf Clusterebene übernimmt, damit bei einem Ausfall von einer Node der Workload trotzdem mit minimalen bis keinem Datenverlust auf eine andere Node übergeben werden kann. Alternativ dazu gibt es auch Workloads, welche von sich aus die Replikation übernehmen können. Diese sind dem vorherigen darin Überlegen, dass die Software die Datenreplikation besser steuern kann, wodurch

die Wahrscheinlichkeit für invalide Daten sinkt. Dazu gehört auch *PostgreSQL*, die Datenbank welche psb-intern genutzt wird. Da das Replizieren über Primary & Secondary Datenbanken übernommen wird welche manuell zugewiesen werden müssen, bietet es sich an die Provision dieser von einem Operator zu übernehmen. *Cloud-NativePG* (kurz CNPG), ist ein Kubernetes-Operator, welcher die Provision von High-Availability Postgres-Datenbanken mit Failover bei Ausfall übernehmen kann, und somit perfekt geeignet ist. CNPG selbst braucht dabei keinen Distributed Storage, da es durch die Anwendungsinterne Replikationslogik den Speicher zwischen Pods verteilt.

Mit der Kombination aus Longhorn und CNPG können somit alle Stateful Workloads von selektron Highly-Available gemacht werden, um Ausfälle der Produktion zu vermeiden.

5.11 Cert-Manager

Um eine sichere Verbindung zu gewährleisten, nutzen wir TLS. TLS garantiert die Verschlüsselung des Nachrichtenaustausch sowie die Identität des Gegenübers mithilfe von Zertifikaten. Um den möglichen Schaden bei Zertifikatdiebstahl oder Sicherheitslücken zu minimieren, sollen Zertifikate ein möglichst kurzes Ablaufdatum besitzen. Das Browser-Forum empfiehlt einen Zeitraum von 1.5 Monaten für öffentliche Server-Zertifikate[8], wobei das Zertifikat innerhalb des ersten Monats automatisch rotiert (ausgewechselt) wird, und Menschen im Falle eines Fehlers trotzdem noch einen halben Monat haben um zu reagieren. Während unser Anwendungsfall kein öffentliches Zertifikat abbildet, sind 47 Tage ein guter Richtwert. Für TLS-Verschlüsselungen zwischen Microservices wird oft ein Zeitraum von nur 7 Tagen[3] oder weniger empfohlen. Um diesen Richtwert bequem einzuhalten ist die zuvor genannte Automatisierung für Zertifikatsverwaltung notwendig, welche *cert-manager*[1] übernimmt. Cert-manager bildet einen Kubernetes Operator, welcher Zertifikate von einer Ansammlung an Zertifikat-Ausstellern anfragen und an Nutzer innerhalb des Kubernetes-Clusters verteilen kann. Aussteller die für unsere Umgebungen infrage kommen sind das *Automated Certificate Management Environment*, kurz ACME, sofern die Kundenfirma einen eigenen Aussteller bereistellt. Mithilfe von ACME lassen sich Zertifikate für Domains oder öffentliche IP-Adressen automatisch anfragen. Im Falle, dass Kunden keine ACME-Infrastruktur bereitstellen, bieten sich von cert-manager verwaltete Selbstsignierte Verschlüsselungen an. Der Operator von cert-manager kann dabei die Zertifikate signieren oder signieren lassen und in *Secrets* speichern. Secrets sind ein Datentyp von Kubernetes, in welchem Daten verschlüsselt gespeichert werden. Diese Secrets können dann von jeglichen Kubernetes-Komponenten benutzt werden. Besonders Bedienerfreundlich ist das Nutzen von cert-manager-Annotationen auf Gateways, womit das Zertifikat mithilfe der in der Komponente existierenden Informationen automatisch erstellt und verwaltet wird. Annotationen sind dabei ein Kubernetes-nativer Weg, beliebige Informationen an einer Komponente festzuhalten. Sofern ein cert-manager Aussteller mit den Namen `aussteller` existiert, muss eine Gateway-Komponente nur mit der Annotation `cert-manager.io/issuer: "austeller"` markiert werden, wodurch der cert-manager Operator erkennt dass diese Resource ein Zertifikat benötigt.

5.12 Observability

Neben Graylog nutzt selektron verschiedene Werkzeuge, um Fehler besser nachvollziehen zu können. *Prometheus* sammelt dabei laufend Messwerte aus dem System, zum Beispiel wie stark Server ausgelastet sind oder wie schnell Anfragen verarbeitet werden. Damit diese Daten nicht verloren gehen, werden sie in *Mimir* gespeichert. *Grafana* visualisiert die gespeicherten Informationen in nutzerdefinierten Diagrammen, sodass Entwicklungen und Auffälligkeiten erkennbar sind. *Alertmanager* sendet Warnmeldungen an den Support und Entwickler, sofern Werte außer der Norm auftreten. Dazu kommt *Tempo*, das sogenannte *Traces* sammelt. Traces zeigen, welchen Weg eine Anfrage durch das System nimmt und an welcher Stelle es zu Problemen kommt. Ergänzt wird die Datenerhebung durch *node-exporter* und *Telegraf*, die zusätzliche Daten direkt von Servern und Anwendungen liefern. Neu ist *kube-state-metrics*, welches Informationen bezüglich Kubernetes in das existierende Metrikerhebungs-Ökosystem integriert. Alle der Komponenten werden mithilfe von 3 Helm-Charts ausgeliefert. Zusammen erlauben die Werkzeuge für eine schnelle Fehlerfindung um Systeme langfristig stabil zu betreiben.

5.13 External Secrets Operator

Während Helm und FluxCD für ein einfaches Deployment sorgen, und cert-manager die Zertifikate für Kommunikationsverschlüsselung verwaltet, ist immernoch ein manueller Teil auf dem Cluster zu konfigurieren: Secrets. Secrets sollen aus Sicherheitsgründen nicht im Klartext gespeichert werden, wofür verschiedene Lösungen existieren. Eine beliebte Lösung sind Secret-Manager wie *Vault*, welche Secrets verschlüsselt an einer zentralen Zugriffsstelle speichern. Die Aufgabe des Transports aus dem Secret-Manager in das Cluster übernimmt der *External Secrets Operator*[2] (kurz ESO), welcher beliebige Secret-Stores, unter anderem Vault, anbindet. ESO ist dabei weitestgehend anbieteragnostisch, und erlaubt es auch nicht offiziell unterstützte Secret-Manager mit einer selbstdefinierten Integration anzubinden, wodurch der psb-interne Passwortmanager ebenfalls integriert werden kann. In den Helm-Charts bleiben somit Referenzen auf nicht-definierte Kubernetes-Secrets, in Git-Repositories bleiben External-Secret-Definitionen welche Referenzen von Secrets im Secret-Manager auf zu erstellende Kubernetes-Secrets beinhalten, und im Secret-Manager liegen die wirklichen Daten. Vorteilhaft ist der Fakt, dass weder in Helm noch in Git Secrets abgelegt werden, und alle genutzten Secrets an einer zentralen Anlaufstelle auffindbar sind. Im Fall, dass Passwörter ausgewechselt werden müssen, ist die Änderung an einer zentralen Stelle zu bewältigen. Falls Kunden Passwörter selbst verwalten möchten, ist eine Anbindung an den Kundeneigenen Secret-Manager möglich.

5.14 Airgap-Installation

Viele der Kunden haben strikte IT-Infrastrukturregeln, in welchem Zugang in das Internet stark beschränkt bis komplett deaktiviert ist. Dafür ist es notwendig, dass

das komplette System ohne Internet aufgesetzt werden kann. Da Kubernetes eine Cloud-Native Lösung ist, stehen Probleme wie dieses nicht auf der Prioritätsliste der Kernentwickler, jedoch existieren Lösungen. Deren Ansätze fallen jedoch außerhalb des Zeitrahmens der Praxisphase und werden in der auf der Praxisphase aufbauenden Bachelorarbeit abgehandelt.

6 Evaluation

Der iterative Zyklus, welcher im Rahmen der Praxisphase angewandt wurde basiert auf dem Prinzip der agilen Softwareentwicklung. Diese Struktur bietet eine solide Grundlage für kontrollierte Identifikation, Bewertung und Integration neuer Technologien in ein System. Dennoch ist dieser Prozess nicht unfehlbar.

Die Identifikationsphase, welche neu gefundene Produkte den übergeordneten Zielen zuweist und anschließend innerhalb eines Backlogs priorisiert gewährt eine strukturierte Abarbeitung. Durch das Einbeziehen von Teamkollegen werden unterschiedliche Perspektiven einbezogen, wodurch die Priorisierung auf fundierter Grundlage steht. Gleichzeitig besteht die Gefahr, dass subjektive Erfahrungen oder betriebliche Hierarchien die Priorisierung negativ beeinflussen und relevante Lösungen für Probleme in Vergessenheit geraten.

Die Evaluationsphase stellt einen den nächsten Bestandteil des Prozesses dar. Das Verwenden einer isolierten Testumgebung ermöglicht eine risikoarme Analyse des Produktes und nullifiziert Auswirkungen auf das bestehende Testsystem, wodurch die Stabilität dessen garantiert werden kann. Allerdings sorgt diese Isolation auch zu einer eingeschränkten Evaluationsfähigkeit, da reale Integrationsbedingungen nicht abgebildet werden. Somit kann ein Produkt in der Testumgebung überzeugen, sich jedoch im Testsystem unerwartet Verhalten oder Probleme verursachen.

Das Dokumentieren der Ergebnisse trägt zur Transparenz und Nachvollziehbarkeit der getroffenen Entscheidungen bei. Außerdem werden Wissenssilos vermieden. Zu kritisieren ist jedoch, dass die Dokumentation ohne Dokumentationsstandards vorgenommen wurde, wodurch es zu uneinheitlichen Ergebnissen dank mangelnder Konsistenz in der Detailtiefe kommen kann.

Zentral ist die Kommunikation durch tägliche Rücksprache und wöchentliche Meetings innerhalb des Teams. Das kurze Kommunikationsintervall sorgt für eine hohe Transparenz und schnelle Feedbackzyklen. Gleichzeitig besteht die Gefahr einer zeitlichen Belastung welche den eigentlichen Arbeitsprozess blockiert. Zuletzt können zu viele Beteiligte den Entscheidungsprozess verlangsamen.

Abschließend findet die Integration des evaluierten Produkts in das Testsystem statt, um die Praxistauglichkeit zu überprüfen. Auf Basis des iterativen Ansatzes werden Anpassungen und Optimierungen vorgenommen. Hauptgefahr ist dabei, dass durch unklar definierte Ziele und Erfolgskriterien ein großer Zeitverlust in der Integration und Optimierung droht.

Zusammenfassend lässt sich festhalten, dass die eingesetzte Vorgehensweise auf Basis der iterativen Struktur, die Einbindung von Mitarbeitern und systematischen

Evaluation eine Grundlage für den Migrationsprozess bietet. Gleichzeitig existieren Verbesserungspotentiale, insbesondere in der Standardisierung von Bewertungs- und Dokumentationsprozessen.

7 Fazit

Die Praxisphase gab mir die Möglichkeit, mich mit technischen Themen auseinanderzusetzen und dabei eigene Verantwortung im betrieblichen Kontext zu übernehmen. Praktische Einblicke in den Unternehmensablauf von der Konzeption hin bis zur Implementation erlaubten mir, meine fachlichen Kompetenzen und die Fertigkeiten des lebenslangen Lernens aus den Vorlesungen anzuwenden und im kollaborativen Arbeiten mit Kollegen und Kolleginnen unter Beweis zu stellen. Die gewonnenen Erkenntnisse bilden die Basis für eine aufbauende Bachelorarbeit, welche im direkten Anschluss zur Praxisphase geschrieben wird.

Zusammenfassend sehe ich die Praxisphase als positiven Teil des Studiums und bin froh, diesen bei der psb intralogistics GmbH ablegen zu dürfen.

Unterschrift Student

Unterschrift Unternehmen

Literatur

- [1] cert-manager Authors. „*cert-manager: Cloud native certificate management*“. 12. Juni 2022. URL: <https://cert-manager.io/> (besucht am 12.06.2026).
- [2] external-secrets Authors. „*External Secrets Operator*“. 2025. URL: <https://external-secrets.io/latest/> (besucht am 18.05.2026).
- [3] Daniel Díaz-Sánchez u. a. „TLS/PKI Challenges and Certificate Pinning Techniques for IoT and M2M Secure Communications“. In: *IEEE Communications Surveys & Tutorials* 21.4 (2019), S. 5. DOI: 10.1109/COMST.2019.2914453. URL: <https://centaur.reading.ac.uk/83566/1/TLSPKI.pdf>.
- [4] Flux. „*Helm Controller | Flux*“. 30. Aug. 2022. URL: <https://fluxcd.io/flux/components/helm/> (besucht am 08.05.2026).
- [5] Flux. „*Image reflector and automation controllers | Flux*“. 30. Aug. 2022. URL: <https://fluxcd.io/flux/components/image/> (besucht am 08.05.2026).
- [6] Flux. „*Kustomize Controller | Flux*“. 24. Mai 2023. URL: <https://fluxcd.io/flux/components/kustomize/> (besucht am 08.05.2026).
- [7] Flux. „*Source Controller | Flux*“. 23. Sep. 2025. URL: <https://fluxcd.io/flux/components/source/> (besucht am 08.05.2026).
- [8] CA/Browser Forum. „*Latest Baseline Requirements*“. 12. Juni 2022. URL: <https://cabforum.org/working-groups/server/baseline-requirements/requirements/> (besucht am 12.06.2026).
- [9] Cloud Native Computing Foundation. „*Flux | CNCF*“. 2026. URL: <https://www.cncf.io/projects/flux/> (besucht am 08.05.2026).
- [10] psb intralogistics GmbH. „*Das Unternehmen*“. URL: <https://www.psb-gmbh.de/psb/das-unternehmen/> (besucht am 29.04.2026).
- [11] OKD Contributors & Red Hat. „*OKD Kubernetes Platform*“. URL: <https://okd.io/> (besucht am 29.04.2026).
- [12] Red Hat. „*Red Hat OpenShift*“. URL: <https://www.redhat.com/en/technologies/cloud-computing/openshift> (besucht am 29.04.2026).
- [13] Headlamp. „*headlamp_flux 0.6.0*“. 7. Nov. 2024. URL: https://artifacthub.io/packages/headlamp/headlamp-plugins/headlamp_flux (besucht am 08.05.2026).
- [14] Helm. „*The package manager for Kubernetes*“. URL: <https://helm.sh/> (besucht am 10.06.2026).
- [15] Inc. (d/b/a Skyhook) KoalaOps. „*Radar | The missing Kubernetes UI*“. 2026. URL: <https://radarhq.io/> (besucht am 14.05.2026).
- [16] Kubernetes. „*Deploy and Access the Kubernetes Dashboard*“. 3. Feb. 2026. URL: <https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/> (besucht am 08.05.2026).
- [17] Kubernetes. „*Ingress*“. 24. Nov. 2025. URL: <https://kubernetes.io/docs/concepts/services-networking/ingress/> (besucht am 04.05.2026).
- [18] Kubernetes. „*Kubeadm | Kubernetes*“. URL: <https://kubernetes.io/docs/reference/setup-tools/kubeadm/> (besucht am 15.05.2026).
- [19] Imhotep Software LLC. „*K9s - Manage Your Kubernetes Clusters in Style*“. 2025. URL: <https://k9scli.io/> (besucht am 08.05.2026).

- [20] Microsoft. „*Hyper-V virtualization in Windows Server and Windows*“. 5. Aug. 2025. URL: <https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/overview> (besucht am 08.05.2026).
- [21] Nubenetes. „*Kubernetes Distributions & Installers*“. URL: <https://nubenetes.com/matrix-table/> (besucht am 29.04.2026).
- [22] k3s Project Authors. „*Lightweight Kubernetes*“. URL: <https://k3s.io/> (besucht am 29.04.2026).
- [23] Kubernetes SIG-Network. „*Introduction: Kubernetes Gateway API*“. URL: <https://gateway-api.sigs.k8s.io/> (besucht am 07.05.2026).
- [24] Kubernetes SIG-UI. „*Headlamp*“. 2026. URL: <https://headlamp.dev/> (besucht am 08.05.2026).
- [25] Rancher by SUSE. „*Introduction: RKE2*“. URL: <https://docs.rke2.io/> (besucht am 29.04.2026).