

Evaluation der Migration von Kubernetes

von

Tim Wall

Mat. Nr. 5014365

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

Pirmasens, den 8. Mai 2026

Inhaltsverzeichnis

1	Betrieb	1
2	Arbeitsplatz	2
3	Aufgabenbereich	2
4	Vorgehensweise	4
5	Methodik Erkenntnisse	7
6	Evaluation	7
7	Fazit	7

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 24.06.

Innerhalb diesem Zeitraum wird mir Einblick in die Unternehmensstruktur, deren Softwareschöpfungsprozess und eine eigene Projektarbeit gewährt. Auch werden außercurriculäre 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.

Unterschrift Student

Unterschrift Unternehmen

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“[1]. 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 durchbrechen.

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 existieren der sprinter, gedacht für leichte Kartons, Kleinbehälter und Tablets, der runloader, gedacht für Kassetten oder Hängeware bis 300 kg Zuladung und der maxloader, gedacht für Paletten bis 1250 kg. 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 den Big Player mit dem hohen Manufakturgrad, vertikaler Integration und persönlicher Kundbetreuung ab.

2 Arbeitsplatz

Der Firmenbereich besteht aus einem Verwaltungsgebäude und zwei Fabrikhallen. Das dreistöckige Verwaltungsgebäude besitzt im Erdgeschoss Kundenempfang, Human Resources, SPS-Entwicklung, UCM-Entwurf und 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 Plätze sind nach Teams innerhalb Trennwänden gruppiert, um Kommunikationswege kurz zu halten, den Austausch zu fördern, jedoch bei Gesprächen andere Gruppen nicht zu belästigen, und bestehen aus jeweils vier bis fünf Leuten. Die Leute innerhalb eines Teams behandeln die gleichen Kundenprojekte. Diese „Sitzverteilung“ ist trotzdem nicht bindend und jeglich eine Empfehlung, um Kommunikation zwischen den Gruppen nicht zu blockieren. Die Gruppenbezeichnung PA INF GUI, kurz für 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 wird mit einer Feuerschutztür von den anderen Projektabwicklung logisch getrennt. Bei Eintritt in die Abteilung wird man von den Meetingsräumen sowie darauffolgende Kücheneinheit und Sanitäranlagen begrüßt. Die Küchentheke besteht aus zwei Kaffeemaschinen, einem Wasserkocher, Kühlschrank sowie Spülmaschine und -becken. 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 Herz der PA INF GUI. 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 selektron WMS, selektron MFCS, selektron RPS 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, da 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 Deployment senkt 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. Ebenfalls 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 dann im Schlimmsten Fall bis zum nächsten Arbeitstag warten, was für den Kunden verlorene Zeit und damit verlorenes Geld bedeutet. Ebenfalls existiert im aktuellen System keine automatische Ausfallsicherheit, welche den Service bei trivialen Fällen entlasten könnte.

Jedes der zuvor genannten Module wurde innerhalb der letzten Jahre in Container-Images gebündelt und mit Containerverwaltungstools wie Docker oder Podman, als auch durchgeführt, um das Deployment weitestgehend zu vereinfachen und Kundeninfrastruktur-unabhängiger zu gestalten. Trotzdem bleiben ziemlich alle genannten Probleme bestehen, da sie über die Containerisierung hinaus gehen.

Mein Aufgabenbereich, das Evaluieren von Kubernetes als Containerorchestrierung für den Software-Stack der psb, ist keine klassische Aufgabe im Bezug auf die Nutzeroberfläche, ist in der Firmenstruktur trotzdem dieser Abteilung untergliedert. Kubernetes führt im Kern, ähnlich wie Docker und Podman, Container aus, geht jedoch weit über das Ausführen sowie Verwalten von diesen hinaus und wird daher als Orchestration bezeichnet. Lastverteilung, Selbstheilung, Ausfallsicherheit, Koordination, Skalierungsmöglichkeiten beschreiben die Möglichkeiten, die Kubernetes im Kern bietet. Kubernetes ist jedoch leicht erweiterbar, wodurch auch Versionierung und Rollbacking über Helm, automatisches Deployment durch FluxCD und HTTP Routing über GatewayAPI abgedeckt sind. **TODO: POTENTIELLE SECRET-INJECTION MIT ESO** Da 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, einen möglichst automatisch deploybares "Walking Skeleton" aus dem aktuellen Basisprojekt zu gestalten, und dabei die verschiedenen Funktionen und Erweiterungen die Kubernetes liefert, zu analysieren wie diese funktionieren als auch zu überprüfen, ob diese konkret für das Projekt Sinn machen. 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 Vorgehensweise

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[8], ähnlich wie bei Linux. Da 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 wegfiltren. Übrig bleiben die 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[9], Rancher RKE2, einer Distribution mit Fokus auf hohen Sicherheitsstandards, Compliance und Enterprise-Support[12], RedHat OpenShift, eine allumfassende Plattform mit Kubernetes im Kern welche GitOps, CI/CD Pipelines, Infrastrukturmanagement, Monitoring und Enterprise-Support umfasst[3] und zuletzt OKD, die Communitygetriebene Version von OpenShift ohne Enterprise-Support[2]. Für das Evaluieren wurde entschlossen, eine möglichst Originalgetreue Distribution von Kubernetes zu benutzen, wodurch OpenShift und OKD mit ihren meinungsstarken Standardkonfigurationen wegfallen und die Entwicklungen von Rancher übrig blieben. Da k3s mit einfacher Installation überzeugt wurde es als initiale Technische Grundlage ausgewählt, besonders mit dem Hintergedanken, dass für den Kubernetes-Kern geschriebene Komponenten auf jeder Distribution lauffähig sind. Diese Portabilität soll später ebenfalls erlauben, weitaus Kundensystem-agnostischer zu arbeiten.

Nach der Wahl der Distribution folgte das Einarbeiten in Kubernetes. Dabei wurde mit den kompletten Grundlagen begonnen: wie bediene ich das Kommandozeilen-tool `kubectl`, was ist ein Pod, wie ist er definiert, wie binden Services sich an Pods, et cetera. Gleichzeitig wurde meiner ersten 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. Währenddessen wurde ein Nachschlagewerk geschrieben und gepflegt, um diese Daten zusammengefasst an einer Stelle zu haben. Schnell wurde klar, das Kubernetes Komponenten manuell schreiben und verwalten möglich ist, jedoch Limitationen aufweist. Ein Kernnachteil ist, dass `kubectl apply` nur Komponenten hinzufügen, aber nicht löschen kann, und man somit nichtmehr definierte Komponenten selbst verwalten muss. Darauf folgt das benutzen von Helm als Paketmanager, welches Charts definiert. Charts sind Sammlungen an Komponentendefinitionsvorlagen, die mit einer Values-Datei zu vollwertigen Kubernetes-Komponentendefinitionen werden, um als konfigurierten 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. Glücklicherweise pflegen die Entwickler von Graylog ebenfalls solch einen Chart, wodurch ich von meiner spartani-

schon 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. Operatoren sind Kubernetes-Erweiterungen bestehend aus Custom-Resource-Definitions (kurz CRDs) und mindestens einem Pod, welcher Instanzen der definierten CRDs (Custom Resources, CRs) überwacht, und daraufhin das Cluster entsprechend bedient. Der MongoDB Operator überwacht z.B. das Cluster nach MongoDB-Komponenten, in welchen eine Datenbank definiert wird, und erstellt daraufhin dynamisch die Datenbank nach den Anforderungen. Falls die Komponente editiert oder gelöscht wird, wird die dadurch erstellte Datenbank dementsprechend editiert oder gelöscht. Somit verwaltet der Operator den Lebenszyklus der angefragten Resource, und der anfragende Service muss sich nicht um die genaue Implementation, sondern nur den Finalzustand kümmern. Dieser Ansatz wird „Infrastructure as Code“ genannt, da Infrastruktur (in diesem Fall Datenbanken) mithilfe von Code statt manueller Prozesse bereitgestellt wird, und somit Konsistenz, Auditierbarkeit, Transparenz und Skalierbarkeit gewährt.

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[5], welcher Layer 7 HTTP/S Routing erlaubt. Dieser Spec wird von Ingress-Controllern implementiert, z.B. nginx oder Traefik. Da selektron jedoch nicht nur Layer 7, sondern auch Layer 4 Routing benötigt, um mit der SPS im Lager zu kommunizieren, wurde auf Ingress' Nachfolger, GatewayAPI gesetzt[10]. GatewayAPI ist genau wie Ingress ein Spec, welcher von extern programmierten GatewayControllern, z.B. Istio oder Traefik, implementiert wird. Er lernt aus den Defiziten von Ingress und trennt Routen, Gateways und GatewayClasses voneinander. Ebenfalls erlaubt er nicht nur HTTP/S, sondern auch gRPC, TCP und UDP-Verbindungen. 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.

Nachdem Graylog auf dem Cluster lief und ein initiales Verständnis für Kubernetes existierte, wurde sich der Portierung selektron auf einen Helm-Chart gewidmet. Als Grundlage für die Portierung wurde die Konfiguration eines Kundensystems übernommen. Während die Portierung schnell verlief, gab es eine Liste an Fehlern zu bewältigen. Da der selektron-Chart in einen anderen Namespace als Graylog deployed wird, war initiale Kommunikation zwischen den beiden Applikationen nicht möglich. Ein `ExternalName` Service wurde genutzt, um die Brücke zwischen den Namespaces zu bilden. Ebenfalls hat der Discovery- und Load-Balancing-Service, basierend auf Spring Boots Netflix Eureka, mit der Kubernetes Service-Schicht nicht kooperiert, da diese Domain-Namen und Container trennt, Eureka jedoch erwartet das 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.

Eine weitere Anforderung ist eine Verwaltungsübersicht für das Kubernetes-Cluster, da Werkzeuge wie `kubectl` und `Helm` zwar alle Funktionalitäten von Kubernetes abdecken, sich jedoch Personalschulung als aufwändig erweist und der Aufwand für simple und repetetive Aufgaben größer ist als Notwendig. Als Visualisierung wurde sich auf Headlamp[11], den offiziellen Nachfolger des Kubernetes-Dashboards[4] und k9s[6], ein grafisches Kommandozeilentool geeinigt. 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 (anstatt k9s' Such-basiertes Modell zur Komponentenauswahl), eine Map, welche existierende Komponenten miteinander Verbindet um sie Logisch zu gruppieren, und Plugin-Support. Da beide mit Kubeconfigs arbeiten, sind sie sofern `kubectl` eingerichtet ist, ohne weiteren Konfigurationsaufwand benutzbar. Besonders Headlamp hat für Überzeugung im Team gesorgt, da es auch ohne Kubernetes-Grundlagen vergleichsweise sehr einfach zu bedienen ist.

Kernbestandteil der mir initial gegebenen Anforderungsliste war die Evaluation der Nutzbarkeit von Kubernetes auf Windows-basierten Systemen. Während Kubernetes 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[7]. 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 Hypervisoren ist, dass diese parallel zum Kernel laufen und unabhängig Userspace des Host-Systems sind, während Type-2 Hypervisor auf dem Host-Betriebssystem laufen. Damit sind Type-2 Hypervisor einfacher zu bedienen, jedoch langsamer und unsicherer. 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 ein Performance-Overhead, besonders in I/O-Operationen entsteht, sowie dass Networking innerhalb Hyper-V sehr fragil ist, weswegen nur externe Switches wie erwartet funktionieren. 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 Portierung von selektron erweisen.

Um zu verhindern, dass die Server-Konfiguration undokumentiert Verändert wird, wird das Deployment, sofern möglich, von FluxCD übernommen. FluxCD ist eine erweiterung von

Eine klassische Serviceaufgabe wäre das Neustarten und Stoppen von Containern.

5 Methodik Erkenntnisse

6 Evaluation

7 Fazit

Literatur

- [1] psb intralogistics GmbH. „*Das Unternehmen*“. URL: <https://www.psb-gmbh.de/psb/das-unternehmen/> (besucht am 29.04.2026).
- [2] OKD Contributors & Red Hat. „*OKD Kubernetes Platform*“. URL: <https://okd.io/> (besucht am 29.04.2026).
- [3] Red Hat. „*Red Hat OpenShift*“. URL: <https://www.redhat.com/en/technologies/cloud-computing/openshift> (besucht am 29.04.2026).
- [4] 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).
- [5] Kubernetes. „*Ingress*“. 24. Nov. 2025. URL: <https://kubernetes.io/docs/concepts/services-networking/ingress/> (besucht am 04.05.2026).
- [6] Imhotep Software LLC. „*K9s - Manage Your Kubernetes Clusters in Style*“. 2025. URL: <https://k9scli.io/> (besucht am 08.05.2026).
- [7] 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).
- [8] Nubenetes. „*Kubernetes Distributions & Installers*“. URL: <https://nubenetes.com/matrix-table/> (besucht am 29.04.2026).
- [9] k3s Project Authors. „*Lightweight Kubernetes*“. URL: <https://k3s.io/> (besucht am 29.04.2026).
- [10] Kubernetes SIG-Network. „*Introduction: Kubernetes Gateway API*“. URL: <https://gateway-api.sigs.k8s.io/> (besucht am 07.05.2026).
- [11] Kubernetes SIG-UI. „*Headlamp*“. 2026. URL: <https://headlamp.dev/> (besucht am 08.05.2026).
- [12] Rancher by SUSE. „*Introduction: RKE2*“. URL: <https://docs.rke2.io/> (besucht am 29.04.2026).