Projektdokumentation

Modul: 347 - Dienst mit Container anwenden
Projektname: LB M347
Autor:innen: Blumer, Natascha; Ebbinghaus, Max; Ritter, Sascha;
Datum: 01.07.2025
Version: 1.0

Einleitung

Dieses Dokument bildet die offizielle **Lern- und Leistungsdokumentation der LB für das Modul 347 – Dienst mit Container anwenden**.
Ziel des Projekts war es, eine vollständige, containerisierte Infrastruktur aufzubauen, die mehrere zentrale Applikationen vereint und dabei die wichtigsten Konzepte von Kubernetes, Container-Orchestrierung und automatisiertem Deployment praktisch umsetzt.

Ausgangslage

Die **M347 GmbH** fungierte dabei als fiktives Unternehmen, das eine modulare, skalierbare und lokal betriebene IT-Umgebung benötigt.
Im Fokus standen dabei folgende Kernkomponenten:

- **Redmine** als Projekt- und Aufgabenmanagementsystem
- **MediaWiki** als zentrale Wissens- und Dokumentationsplattform
- **WordPress** als Firmenblog für News und interne Mitteilungen
- **Prometheus** zur Erfassung und Überwachung von Metriken
- **Grafana** zur grafischen Visualisierung dieser Metriken

Alle Services wurden containerisiert, mit **Kubernetes** orchestriert und über einen zentralen **Ingress-Controller** zugänglich gemacht. Besonderer Wert wurde auf eine saubere Trennung der Komponenten (Namespaces), die Verwendung von **Persistent Volumes** für Datenhaltbarkeit, sowie den Einsatz von **ConfigMaps** und **Secrets** für eine sichere und flexible Konfiguration gelegt.

Diese Dokumentation beschreibt sowohl die technische Umsetzung der einzelnen Komponenten als auch die getroffenen Architekturentscheidungen, Herausforderungen während der Umsetzung und die abschliessende Qualitätssicherung in Form von Testplänen und Arbeitsjournalen.

Infrastruktur

Für den Betrieb unserer containerisierten Anwendung haben wir eine klar strukturierte Infrastruktur in Kubernetes aufgebaut.
Eine ausführliche Beschreibung mit Diagramm findet sich hier:

- Zur Infrastruktur

Eingesetzte Technologien

Für den Aufbau und Betrieb der Infrastruktur wurden folgende Kerntechnologien eingesetzt:

- **Docker**
  Dient als Container-Engine, um die einzelnen Applikationen in isolierten Containern auszuführen. In unserem Setup wird Docker nicht direkt verwendet, um Container manuell zu starten, sondern als Runtime-Backend für Kubernetes (via Minikube). Dadurch profitieren wir von der Portabilität und den bekannten Tools aus dem Docker-Ökosystem, während die Steuerung vollständig über Kubernetes erfolgt.

- **Kubernetes**
  Orchestriert die verschiedenen Container, übernimmt die Verwaltung von Deployments, Services, Volumes und Skalierung. Kubernetes stellt sicher, dass alle Komponenten zuverlässig und im gewünschten Zustand laufen.

- **Minikube**
  Dient als lokale Laufzeitumgebung für Kubernetes. Mit Minikube kann ein vollständiger Kubernetes-Cluster auf einem lokalen Rechner simuliert werden, ideal für Entwicklungs- und Testzwecke.

- **Helm**
  Wird als Paketmanager für Kubernetes genutzt, um den Ingress Controller schnell und standardisiert bereitzustellen. Helm vereinfacht das Verwalten, Updaten und Rollbacken von Kubernetes-Ressourcen erheblich.

- **Kubectl**
  Das zentrale Kommandozeilenwerkzeug zur Steuerung des Kubernetes-Clusters. Mit `kubectl` werden Deployments durchgeführt, Pods überwacht, Logs gelesen und die gesamte Clusterverwaltung gesteuert.

Konfiguration

In diesem Abschnitt sind die technischen Details jeder Komponente dokumentiert. Für jede Applikation oder Infrastrukturkomponente existiert ein eigenes Konfigurationsdokument mit Informationen zu Deployments, Services, Volumes, Secrets, Ingress-Routen und weiteren Ressourcen.
Ziel ist eine übersichtliche, modulare und nachvollziehbare Darstellung der jeweiligen Kubernetes-Konfiguration inklusive YAML-Files und Besonderheiten.

Konfiguration der einzelnen Komponenten

- Redmine - Zur Konfiguration
- MediaWiki - Zur Konfiguration
- WordPress - Zur Konfiguration
- Prometheus - Zur Konfiguration
- Grafana - Zur Konfiguration
- Ingress - Zur Konfiguration

Automatisierte Installation

> [!NOTE] Für den Betrieb der Umgebung müssen Docker Desktop, Minikube und Helm bereits installiert sein.

Um die Umgebung schnell und komfortabel bereitzustellen, haben wir ein **Batch-Skript** für die automatisierte Installation erstellt. Dieses übernimmt alle notwendigen Schritte und deployt das gesamte Projekt in einem Durchlauf.

- Automatisierte Installation - Zum Batchfile

Alternativ kann die Installation auch **manuell** Schritt für Schritt durchgeführt werden (siehe weiter unten).

Automatisierte Deinstallation

Analog dazu steht ein separates **Batch-Skript** für die vollständige und saubere Deinstallation des Projekts zur Verfügung.

- Automatisierte Deinstallation - Zum Batchfile

Auch die Deinstallation kann wahlweise **manuell** erfolgen (siehe weiter unten).

Anleitung zur manuellen Installation

> [!NOTE] Für den Betrieb der Umgebung müssen Docker Desktop, Minikube und Helm bereits installiert sein.

Bevor Anwendungen im Kubernetes-Cluster bereitgestellt werden können, muss die lokale Infrastruktur gestartet und überprüft werden. Dieser Abschnitt beschreibt die nötigen Schritte zur Initialisierung des Minikube-Clusters.

1. Minikube starten

Mit folgendem Befehl wird ein lokaler Kubernetes-Cluster erstellt:

```
minikube start
```

Minikube lädt dabei die benötigten Komponenten und Images herunter. Beim **ersten Start** kann dieser Vorgang **mehrere Minuten (5–10 min)** in Anspruch nehmen.

> [!WARNING] Sollte eine Fehlermeldung erscheinen, liegt dies häufig daran, dass **Docker Desktop nicht aktiv** ist.
> In diesem Fall: Docker starten und den Befehl erneut ausführen.

2. Clusterstatus überprüfen

Nach erfolgreichem Start lässt sich der Status des Clusters wie folgt prüfen:

```
kubectl get nodes
```

Wenn ein Node mit dem Status `Ready` erscheint, ist der Cluster betriebsbereit.

Manuelle Installation der einzelnen Komponenten

Für den Betrieb der Umgebung werden alle Anwendungen und Dienste in **eigenständigen Namespaces** betrieben. Dies ermöglicht eine strukturierte Trennung, eine gezielte Ressourcenzuweisung und erleichtert die Verwaltung der Konfiguration pro Komponente.

Der Ingress-Controller selbst wird **namespace-unabhängig** bereitgestellt und verwendet denselben zentralen Zugriffspunkt für alle Anwendungen.

Reihenfolge der Komponenteninstallation

Die Komponenten werden in logischer Abfolge installiert, wobei Anwendungen mit Datenbank zuerst kommen.

Die empfohlene Installationsreihenfolge lautet:

1. **Redmine**
2. **MediaWiki**
3. **WordPress**
4. **Prometheus**
5. **Grafana**
6. **Ingress**

Diese Reihenfolge stellt sicher, dass notwendige Abhängigkeiten (z. B. Datenbanken) vor der Anwendung bereitstehen und Referenzen (z. B. Service-Namen) korrekt aufgelöst werden können.

Reihenfolge beim Deployment der YAML-Dateien

Für jede Komponente werden die zugehörigen Ressourcen in einer festgelegten Reihenfolge installiert. Dabei wird jeweils **zuerst die Datenbank**, **danach die Anwendung** bereitgestellt.

Die empfohlene Installationsreihenfolge lautet:

1. **Konfiguration** (`ConfigMap` & `Secret`)
   Enthält Umgebungsvariablen und Zugangsdaten für App und Datenbank.

2. **Speicher** (`PersistentVolumeClaim`)
   Fordert persistenten Speicher für Datenbank und Anwendung an.

3. **Deployments**
   Startet die Container für Datenbank und Anwendung mit zugehöriger Konfiguration und Speicher.

4. **Services**
   Stellt interne Netzwerkzugänge zur Verfügung, über die App und Datenbank kommunizieren können.

Installationsanleitungen

Die folgenden Seiten beschreiben die Installation jeder Komponente im Detail:

- Redmine - Zur Installationsanleitung
- MediaWiki - Zur Installationsanleitung
- WordPress - Zur Installationsanleitung
- Prometheus - Zur Installationsanleitung

- Grafana - Zur Installationsanleitung
- Ingress - Zur Installationsanleitung

Manuelle Deinstallation

Um das Projekt **manuell zu entfernen**, reicht es aus, die erstellten Namespaces und die Ingress-Klasse zu löschen. Damit werden alle enthaltenen Ressourcen (Deployments, Services, PVCs etc.) automatisch mit entfernt.

```
kubectl delete namespace ingress-nginx
kubectl delete ingressclass nginx
kubectl delete namespace m347-grafana
kubectl delete namespace m347-prometheus
kubectl delete namespace m347-wordpress
kubectl delete namespace m347-mediawiki
kubectl delete namespace m347-redmine
```

Testplan & Testergebnisse

Zur Qualitätssicherung und Funktionsüberprüfung unserer Kubernetes-Infrastruktur wurden gezielte Tests für jede zentrale Systemkomponente durchgeführt. Dabei lag der Fokus auf folgenden Aspekten:

- **Korrekte Netzwerkverbindungen** (z. B. über ClusterIP oder Ingress),
- **Erwartungsgemässe Funktionalität** der Anwendungen,
- **Sicherstellung der Datenpersistenz** mittels PVCs.

Die Tests wurden **komponentenweise geplant und ausgeführt**.
Für jede Anwendung bzw. Infrastrukturkomponente wurde ein separates Testdokument erstellt, das sowohl die Testplanung als auch die zugehörigen Testergebnisse enthält:

- Redmine - Zum Testing
- MediaWiki - Zum Testing
- WordPress - Zum Testing
- Prometheus - Zum Testing
- Grafana - Zum Testing
- Ingress - Zum Testing

Arbeitsjournal und persönliches Fazit

Im Rahmen dieses Projekts hat jedes Gruppenmitglied ein individuelles **Arbeitsjournal** geführt. Darin wurden die persönlichen Arbeitsschritte, Herausforderungen sowie technische Erkenntnisse dokumentiert. Ergänzend dazu hat jede Person ein **persönliches Fazit** verfasst, in dem die individuellen Lernerfolge, Erfahrungen mit Kubernetes sowie das Zusammenspiel im Team reflektiert werden.

Die folgenden Links führen direkt zu den jeweiligen Arbeitsjournalen:

- NB - Zum Arbeitsjournal
- ME - Zum Arbeitsjournal
- SR - Zum Arbeitsjournal

Hilfestellungen

Während der Bearbeitung des Projekts haben wir verschiedene Formen der Unterstützung in Anspruch genommen – fachlich, methodisch und sprachlich. Der folgende Abschnitt dokumentiert, welche **externen Hilfestellungen** zum Einsatz kamen und in welcher Form sie das Projekt beeinflusst haben.

Yves Kaufmann (Dozent M347)

- Unterstützung bei der Konstruktion des Infrastrukturdiagramms

Thomas Kägi (Ausbildner, NB)

- Unterstützung bei der Informationssammlung und beim Brainstorming
- Unterstützung bei der Fehlersuche, beim Debugging und bei der Lösungsfindung
- Beratend bei Fragen und Unklarheiten zur Materie
- Unterstützung beim Erstellen der Projektstruktur
- Unterstützung beim Erstellen der Batchfiles
- Testen der Installation und Gegenlesen

ChatGPT

- Rechtschreibekorrektur und Formulierungshilfe beim Verfassen der Dokumentation
- Unterstützung bei der Installation von Helm
- Fehlersuche
- Inspirationsgebend bei den Testfällen
- Unterstützung beim Aufsetzen von Prometheus und Grafana
- Unterstützung beim Aufsetzen von Redmine und WordPress
- Hilfreich beim Nachschlagen von Konventionen
- Unterstützung beim Einrichten des Dashboards in Grafana

Weitere Quellen

- https://kubernetes.io/docs/concepts/services-networking/service/
- https://hub.docker.com/r/bitnami/wordpress
- https://hub.docker.com/r/prom/prometheus
- https://hub.docker.com/r/grafana/grafana
- https://codebeautify.org/yaml-validator