

Infrastruktur

Diagramm der Infrastruktur

Für unser Projekt haben wir eine vollständig containerisierte Infrastruktur in Kubernetes umgesetzt. Die gesamte Umgebung läuft lokal auf einem einzigen Node. Sie umfasst insgesamt **9 Pods** inklusive einem zentralen **Ingress Controller**, **8 Services**, **8 PVCs** sowie dedizierte Komponenten für Konfiguration und Secrets.

Bestandteile der Infrastruktur

Anwendungspods

- **Pod 1:** Redmine
- **Pod 3:** WordPress
- **Pod 5:** MediaWiki
- **Pod 7:** Grafana

Alle Anwendungspods sind über einen **ClusterIP-Service** verfügbar und werden durch den Ingress Controller angesprochen. Jeder Anwendungspod läuft in einem eigenen Deployment mit **replicas: 1** und nutzt gemeinsam mit seinem zugehörigen Datenbankpod **eine ConfigMap** und **ein Secret**, um eine konsistente, modulare und sichere Konfiguration zu ermöglichen.

Zusätzlich verwendet jeder dieser Pods einen **eigenen PVC**, um Daten wie Uploads (z. B. Bilder bei WordPress), Medieninhalte oder Dashboard-Einstellungen (z. B. bei Grafana) persistent zu speichern.

Datenbankpods

- **Pod 2:** PostgreSQL (für Redmine)
- **Pod 4:** MariaDB (für WordPress)
- **Pod 6:** MariaDB (für MediaWiki)
- **Pod 8:** Prometheus (zur Speicherung von Metriken)

Jede Datenbank wird über einen eigenen **ClusterIP-Service** angesprochen und nutzt ein **Persistent Volume** über einen zugehörigen **PVC (Persistent Volume Claim)**. Auch hier ist jeweils ein einzelnes Replikat (**replicas: 1**) im Einsatz.

Infrastrukturpods

- **Pod 9: Ingress Controller (nginx)**
Der Ingress Controller übernimmt die zentrale Exponierung aller Anwendungen nach aussen. Die Weiterleitung erfolgt anhand definierter Hostnamen wie **redmine.m347.ch** oder **mediawiki.m347.ch**.

Konfiguration: ConfigMaps & Secrets

Für Konfigurationswerte und sensible Daten (z. B. Zugangsdaten oder Umgebungsvariablen) haben wir **pro Anwendungspod** eine **eigene ConfigMap** und ein **eigenes Secret** definiert.

Dieses Vorgehen unterstützt eine **wiederverwendbare, modulare und sichere Konfiguration** einzelner Komponenten – z. B. Datenbank-URLs, Passwörter oder Metriken-Konfigurationen.

Erreichbarkeit

Alle Applikationen sind nicht über Pfade wie `http://localhost/mediawiki`, sondern über **eigene Hostnamen** erreichbar:

- `http://redmine.m347.ch`
- `http://mediawiki.m347.ch`
- `http://wordpress.m347.ch`
- `http://prometheus.m347.ch`
- `http://grafana.m347.ch`

Warum Hostnamen statt Pfade?

Viele Webapplikationen, insbesondere WordPress und MediaWiki, sind **nicht dafür ausgelegt, in Unterverzeichnissen zu laufen**. Ohne dedizierte Hostnamen können z. B. Stylesheets (CSS), Weiterleitungen oder Plugins fehlerhaft funktionieren.

Wir haben uns deshalb bewusst für die Verwendung von eigenen Hostnamen (z. B. `wordpress.m347.ch`) entschieden, um eine saubere und zuverlässige Erreichbarkeit zu gewährleisten. Durch die Nutzung eines **externen DNS-Dienstes** entfällt zudem die Notwendigkeit, die lokale Hosts-Datei manuell anzupassen.

Die Weiterleitung und Erreichbarkeit werden dabei zentral über den Ingress-Controller gesteuert.

Eingesetzte Services

Alle Services wurden auf den Typ `ClusterIP` gesetzt. Das bedeutet, sie sind nur intern im Cluster erreichbar. Der externe Zugriff erfolgt ausschliesslich über den **Ingress Controller**.

Vorteile dieses Ansatzes:

- **Zentrale Steuerung der Exponierung:** Nur der Ingress Controller ist von aussen erreichbar, was die Sicherheit erhöht.
- **Klarer Aufbau:** App-Pods sprechen Datenbank-Pods über interne ClusterIP-Services an.
- **Standardisierte Adressierung:** Dank eigener Hostnamen erfolgt der Zugriff über Klartext-URLs wie `http://redmine.m347.ch`.

Umgang mit Replikation

Alle Deployments in unserem Projekt verwenden aktuell:

```
replicas: 1
```

Dies bedeutet, dass pro Komponente jeweils **ein einzelner Pod** betrieben wird.

Für unser Projekt ist diese Konfiguration zweckmässig, da die Umgebung lokal auf einem **Single-Node-Cluster** betrieben wird und Hochverfügbarkeit in diesem Kontext keinen praktischen Nutzen bietet.

In einer produktiven Umgebung würde man für skalierbare Anwendungen (z. B. WordPress oder MediaWiki) typischerweise `replicas: 2` oder mehr definieren, um eine höhere Verfügbarkeit, Lastverteilung und unterbrechungsfreie Updates zu ermöglichen.

Für Datenbankpods setzen wir bewusst auf **ein einzelnes Replikat**, da echte Datenbankreplikation komplexer ist und hier nicht notwendig war.

Kommunikation & Zugriff

Die Kommunikation erfolgt gemäss folgendem Muster:

```
User/Browser → Ingress Controller
Ingress Controller → ClusterIP-Service (App)
App-Pod → ClusterIP-Service (DB)
DB-Pod → PVC → PV
ConfigMap/Secret → von Pods referenziert
```

Übersicht der Persistent Volumes & Claims

Wir haben uns bewusst für **acht separate PVCs** entschieden, einen für jede datenhaltende Komponente:

PVC	Zugehöriger Pod	Zweck
App-Daten Redmine	Pod 1 - Redmine	Speichert Anhänge und Konfigurationen
Claim für Redmine-DB	Pod 2 – PostgreSQL	Speichert Redmine-Daten
App-Daten WordPress	Pod 3 - WordPress	Speichert Uploads, Plugins, Themes
Claim für WordPress-DB	Pod 4 – MariaDB	Speichert WordPress-Daten
App-Daten MediaWiki	Pod 5 - MediaWiki	Speichert Bilder, Erweiterungen
Claim für MediaWiki-DB	Pod 6 – MariaDB	Speichert MediaWiki-Daten
App-Daten Grafana	Pod 7 - Grafana	Speichert Dashboards und Einstellungen
Claim für Prometheus	Pod 8 – Prometheus	Speichert Zeitreihenmetriken

Vorteile dieses Ansatzes:

- **Datenisolierung:** Jede Anwendung speichert ihre Daten unabhängig.
- **Wartbarkeit:** Daten lassen sich gezielt sichern, wiederherstellen oder verschieben.
- **Erweiterbarkeit:** Einfache Anpassung oder Skalierung einzelner Komponenten ist möglich.

Fazit

Unsere Infrastruktur ist klar strukturiert, modular aufgebaut und folgt den Best Practices für Kubernetes:

- Trennung von Anwendungen, Datenhaltung und Infrastruktur
- Sichere & zentrale Exponierung über Ingress und eigene Hostnamen
- Datenpersistenz durch dedizierte PVCs
- Konfigurationsverwaltung durch ConfigMap & Secret
- Skalierbarkeit durch **replicas**-Definition pro Pod

Das beigefügte Architekturdiagramm veranschaulicht die Infrastruktur übersichtlich und dient als technische Grundlage für Deployment, Betrieb und Wartung.