Konfiguration - Prometheus

- **Verwendung**:
  Prometheus wird als Monitoring-Lösung innerhalb des Kubernetes-Clusters betrieben. Die Konfiguration erfolgt über eigenständige YAML-Dateien, die das Deployment, den Netzwerkzugriff, die Speicherung von Metriken und die Integration mit Exportern (z. B. Blackbox Exporter) definieren.
- **Einsatzgrund**:
  Prometheus ist ein etabliertes Open-Source-Monitoring- und Alerting-System, das besonders für die Überwachung von Cloud-nativen Umgebungen entwickelt wurde. Es ermöglicht die kontinuierliche Sammlung von Zeitreihenmetriken, bietet leistungsfähige Abfragemöglichkeiten und eine hohe Flexibilität durch modulare Konfiguration.
- **Rolle im System**:
  Prometheus übernimmt die Aufgabe der Überwachung und stellt Metriken für Anwendungen wie Grafana bereit. Es überwacht sowohl sich selbst als auch die Dienste Redmine, WordPress und MediaWiki über HTTP-Checks mit dem Blackbox Exporter. Die Anwendung speichert alle gesammelten Metriken dauerhaft und ist über Ingress erreichbar.

Ressourcen - Anwendung

Im Folgenden sind alle YAML-Dateien aufgeführt, die zur Bereitstellung und Konfiguration der Anwendung benötigt werden. Sie decken u. a. Container-Deployment, Netzwerkzugriff, Speicheranbindung sowie Konfigurations- und Zugriffsdaten ab.

Ressource	Dateiname	Zweck
Deployment	`deployment.yaml`	Startet den App-Container
Service	`service.yaml`	Interner Netzwerkzugriff
Persistent Volume Claim	`pvc.yaml`	Persistenz für Daten
ConfigMap	`configmap.yaml`	Konfigurationsparameter
Secret	`secret.yaml`	Zugangsdaten oder Tokens
Blackbox Exporter	`blackbox-exporter-deployment.yaml`	Überwacht Services
Ingress	`prometheus-ingress.yaml`	Zugriff via Hostname

Files

Deployment

> Definiert das Deployment für Prometheus inkl. Mount des Volumes, Pfad zur ConfigMap und Container-Ports.

deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: m347-prometheus
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus
          args:
            - "--config.file=/etc/prometheus/prometheus.yml"
          ports:
            - containerPort: 9090
          volumeMounts:
            - name: prometheus-config
              mountPath: /etc/prometheus
            - name: prometheus-storage
              mountPath: /prometheus
      volumes:
        - name: prometheus-config
          configMap:
            name: prometheus-config
        - name: prometheus-storage
          persistentVolumeClaim:
            claimName: prometheus-pvc
```

Erklärung der Konfiguration

- **replicas: 1**
  Startet einen einzelnen Prometheus-Pod.
- **image: prom/prometheus**
  Nutzt das offizielle Prometheus-Image.
- **ports: 9090**
  Stellt den Prometheus-Web-UI und die API auf Port 9090 bereit.
- **volumeMounts**
  Sagt wo im Container das Volume eingebunden wird. Nutzt den Namen aus `volumes` und gibt einen Pfad an.
    - **prometheus-config:** Bindet die Konfigurationsdatei aus der ConfigMap.

    - **prometheus-storage:** Bindet den persistenten Speicher für Zeitreihendaten.

- **volumes**
  Definiert welche Volumes im Pod verfügbar sind. Es beschreibt die Quelle des Volumes, also woher die Daten kommen.

    - **ConfigMap:** Stellt die prometheus.yml im Container bereit.
    - **PersistentVolumeClaim:** Speichert Prometheus-Daten dauerhaft über `prometheus-pvc`.

Service

> Stellt einen internen Kubernetes-Service zur Verfügung, damit z. B. Grafana Prometheus als Datenquelle erreichen kann.

service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: prometheus-service
  namespace: m347-prometheus
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 9090
  selector:
    app: prometheus
```

Erklärung der Konfiguration

- **type: ClusterIP**
  Erstellt einen internen Service, der nur im Kubernetes-Cluster erreichbar ist.

- **selector: app: prometheus**
  Verbindet den Service mit dem Prometheus-Pod anhand des Labels `app: prometheus`.

Persistente Daten (PVC)

> Stellt ein Persistent Volume bereit, in dem Prometheus seine Zeitreihendaten speichert.

pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: prometheus-pvc
  namespace: m347-prometheus
spec:
```

```
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: standard
```

Erklärung der Konfiguration

- **accessModes: ReadWriteOnce**
  Der Speicher darf von genau einem Node gleichzeitig gelesen und beschrieben werden.
- **resources.requests.storage: 10Gi**
  Fordert 10 Gibibyte Speicherplatz an.
- **storageClassName: standard**
  Verwendet die Standard-StorageClass des Clusters

Blackbox-exporter-deployment

blackbox-exporter-deployment .yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blackbox-exporter
  namespace: m347-prometheus
spec:
  replicas: 1
  selector:
    matchLabels:
      app: blackbox-exporter
  template:
    metadata:
      labels:
        app: blackbox-exporter
    spec:
      containers:
        - name: blackbox-exporter
          image: prom/blackbox-exporter:latest
          ports:
            - containerPort: 9115
---
apiVersion: v1
kind: Service
metadata:
  name: blackbox-exporter
  namespace: m347-prometheus
spec:
  selector:
    app: blackbox-exporter
```

```
  ports:
    - protocol: TCP
      port: 9115
      targetPort: 9115
```

Erklärung der Konfiguration

Hier befinden sich zwei yaml-Files in einem. Zum einen wieder ein Deployment (oben) und ein Service (unten).

Deployment

- **replicas: 1**
  Erstellt einen Pod, ausreichend für Entwicklungs- oder Testumgebungen.
- **image: prom/blackbox-exporter:latest**
  Nutzt das offizielle Docker-Image des Blackbox Exporters in der neuesten Version.
- **containerPort: 9115**
  Öffnet Port 9115 im Container, den der Blackbox Exporter standardmässig für HTTP-/TCP-/ICMP-Checks verwendet.

Service

- **port: 9115 & targetPort: 9115**
  Der Service leitet Anfragen auf Port 9115 direkt an den Container-Port weiter. So kann Prometheus den Blackbox Exporter unter `blackbox-exporter:9115` im Cluster erreichen.

ConfigMap & Secret

> Enthält die zentrale Prometheus-Konfiguration (z. B. scrape-Intervalle, Targets) sowie optionale vertrauliche Informationen.

ConfigMap

configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: m347-prometheus
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: prometheus
        static_configs:
          - targets: ['localhost:9090']
```

```
    - job_name: grafana
      static_configs:
        - targets:
            - grafana-service.m347-grafana.svc.cluster.local:80

    - job_name: wordpress-http
      metrics_path: /probe
      params:
        module: [http_2xx]
      static_configs:
        - targets:
            - wordpress-service.m347-wordpress.svc.cluster.local
      relabel_configs:
        - source_labels: [__address__]
          target_label: __param_target
        - source_labels: [__param_target]
          target_label: instance
        - target_label: __address__
          replacement: blackbox-exporter.m347-prometheus.svc.cluster.local:9115

    - job_name: redmine-http
      metrics_path: /probe
      params:
        module: [http_2xx]
      static_configs:
        - targets:
            - redmine-service.m347-redmine.svc.cluster.local
      relabel_configs:
        - source_labels: [__address__]
          target_label: __param_target
        - source_labels: [__param_target]
          target_label: instance
        - target_label: __address__
          replacement: blackbox-exporter.m347-prometheus.svc.cluster.local:9115

    - job_name: mediawiki-http
      metrics_path: /probe
      params:
        module: [http_2xx]
      static_configs:
        - targets:
            - mediawiki-service.m347-mediawiki.svc.cluster.local
      relabel_configs:
        - source_labels: [__address__]
          target_label: __param_target
        - source_labels: [__param_target]
          target_label: instance
        - target_label: __address__
          replacement: blackbox-exporter.m347-prometheus.svc.cluster.local:9115
```

Erklärung der Konfiguration

- **scrape_interval: 15s**
  Alle 15 Sekunden werden Targets abgefragt. Kleinere Zeitintervalle ergeben besseres Monitoring, dafür grössere Serverbelastung. Umgekehrt könnte man bei einem grösseres Zeitintervall kleinere Ausfälle übersehen.

- **Job: prometheus**
  Überwacht Prometheus selbst auf `localhost:9090`. Dies ist nicht unbedingt notwendig, da falls Prometheus nicht erreichbar ist man dies gar nicht nachschauen kann über Prometheus selbst und falls er "down" ist, keine Daten sammeln kann. es ist aber eine Default-Einstellung und wurde so beibehalten.

- **Job: grafana**
  Fragt Grafana direkt über den Cluster-Service `grafana-service` ab.

- **Jobs: wordpress-http, redmine-http, mediawiki-http**
  HTTP-Checks auf WordPress, Redmine und MediaWiki über den Blackbox Exporter.

- **metrics_path: /probe & module: http_2xx**
  Nutzen den Blackbox Exporter für HTTP-Checks; Erfolg nur bei HTTP-Status 2xx.

Secret

secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: prometheus-secret
  namespace: m347-prometheus
type: Opaque
stringData:
  admin-user: admin
  admin-password: password
```

Erklärung der Konfiguration

- **Kind: Secret**
  Speichert sensible Daten wie Benutzernamen und Passwörter.

- **stringData**
  Akzeptiert Klartext, muss also nicht Base64-kodiert sein. Es wird dann intern umkodiert.

Ingress / Externer Zugriff

> Regelt den externen Zugriff auf Prometheus über den Hostnamen mithilfe eines Ingress Controllers.

Die Datei `prometheus-ingress.yaml` definiert, unter welchem Hostnamen (`prometheus.m347.ch`) Prometheus von ausserhalb des Clusters erreichbar ist.
Sie verweist auf den zentralen Ingress Controller und sorgt für die Weiterleitung eingehender Anfragen an den

zugehörigen Service von Prometheus.
Da das zugrundeliegende Ingress-System für alle Anwendungen identisch ist, wird die übergeordnete Konfiguration des Ingress Controllers inklusive Routingprinzipien und Klassendefinition zentral in der Konfigurationsdatei des Ingress Controllers dokumentiert.

Besonderheiten & Herausforderungen

Die Implementierung von Prometheus als Monitoring-Lösung im Kubernetes-Cluster erforderte eine präzise Konfiguration der Scrape-Jobs, um sowohl interne Metriken als auch End-to-End-Checks über den Blackbox Exporter abzudecken. Eine Herausforderung war dabei die zuverlässige Integration des Blackbox Exporters und die exakte Definition der relabel_configs, damit Prometheus korrekt auf externe Services zugreifen konnte. Darüber hinaus musste die persistente Speicherung der Metriken sichergestellt werden, um Langzeitdaten auch bei Neustarts zu erhalten und eine kontinuierliche Überwachung zu gewährleisten.