Konfiguration - Grafana

- **Verwendung**: Grafana wird als Dashboard- und Visualisierungslösung innerhalb des Kubernetes-Clusters betrieben. Die Konfiguration erfolgt über eigenständige YAML-Dateien, die das Deployment, Netzwerkzugriff, persistente Speicherung und Provisioning von Dashboards und Datenquellen definieren.
- **Einsatzgrund**: Grafana ist eine weit verbreitete Open-Source-Plattform für die Visualisierung von Zeitreihenmetriken. Es ermöglicht die flexible Erstellung von Dashboards, die Darstellung von Messwerten aus Prometheus und anderen Datenquellen und bietet leistungsstarke Funktionen zur Analyse und Darstellung von Monitoring-Daten.
- **Rolle im System**: Grafana dient als Frontend zur Darstellung und Auswertung der von Prometheus gesammelten Metriken. Es bietet Dashboards zur Überwachung der Anwendungslandschaft (z. B. WordPress, Redmine, MediaWiki) und ermöglicht eine schnelle Diagnose von Problemen. Die Anwendung ist über Ingress extern erreichbar und speichert Dashboards und Konfigurationen persistent.

Ressourcen - Anwendung

Im Folgenden sind alle YAML-Dateien aufgeführt, die zur Bereitstellung und Konfiguration von **Grafana** benötigt werden. Sie definieren unter anderem das Deployment, Netzwerkzugang, persistente Datenspeicherung und die automatische Anbindung an **Prometheus**.

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
Data Provisioning	`data-provisioning.yaml`	Konfiguriert automatisch Prometheus als Datenquelle
Ingress	`grafana-ingress.yaml`	Zugriff via Hostname

Files

Deployment

> Startet den Grafana-Pod mit allen nötigen Konfigurationen, Volumes und Umgebungsvariablen. Verlinkt u. a. PVC, ConfigMap und Secret.

deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
  namespace: m347-grafana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
        - name: grafana
          image: grafana/grafana:latest
          ports:
            - containerPort: 3000
          volumeMounts:
            - name: grafana-storage
              mountPath: /var/lib/grafana
            - name: grafana-datasource
              mountPath: /etc/grafana/provisioning/datasources
          env:
            - name: GF_SECURITY_ADMIN_USER
              valueFrom:
                secretKeyRef:
                  name: grafana-secret
                  key: GF_SECURITY_ADMIN_USER
            - name: GF_SECURITY_ADMIN_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: grafana-secret
                  key: GF_SECURITY_ADMIN_PASSWORD
      volumes:
        - name: grafana-storage
          persistentVolumeClaim:
            claimName: grafana-pvc
        - name: grafana-datasource
          configMap:
            name: grafana-config
```

Erklärung der Konfiguration

- **image: grafana/grafana:latest**
  Verwendet das offizielle Grafana-Image in der neuesten Version.

- **volumeMounts**
  Sagt wo im Container das Volume eingebunden wird. Nutzt den Namen aus `volumes` und gibt einen Pfad an.
  - **grafana-storage:** Bindet den persistenten Speicher unter `/var/lib/grafana`, z.B. für Dashboards oder User-Daten.
  - **grafana-datasource:** Bindet die Datenquellen-Konfiguration aus der ConfigMap unter `/etc/grafana/provisioning/datasources`.
- **env**
  - **GF_SECURITY_ADMIN_USER & GF_SECURITY_ADMIN_PASSWORD:**
    Lesen Admin-Benutzername und Passwort aus dem Secret `grafana-secret` und setzen sie als Umgebungsvariablen für Grafana.
- **volumes**
  Definiert welche Volumes im Pod verfügbar sind. Es beschreibt die Quelle des Volumes, also woher die Daten kommen.
  - **PersistentVolumeClaim:** Speichert Grafana-Daten dauerhaft.
  - **ConfigMap:** Enthält Konfigurationen für automatisch provisionierte Datenquellen.

Service

> Stellt einen internen Kubernetes-Service zur Verfügung, über den Grafana im Cluster erreichbar ist.

service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: grafana-service
  namespace: m347-grafana
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 3000
  selector:
    app: grafana
```

Erklärung der Konfiguration

- **type: ClusterIP**
  Erstellt einen internen Service, der nur innerhalb des Kubernetes-Clusters erreichbar ist.
- **selector: app: grafana**
  Verbindet den Service mit dem Grafana-Pod über das Label `app: grafana`.

Persistente Daten (PVC)

> Verknüpft Grafana mit einem Persistent Volume, um Daten wie gespeicherte Dashboards und User-Einstellungen dauerhaft zu sichern.

pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: grafana-pvc
  namespace: m347-grafana
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

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 für die dauerhafte Speicherung von Grafana-Daten an. Kubernetes benötigt die Form GibiByte (Gi) anstelle von Gigabyte (GiB) um zu funktionieren.

ConfigMap & Secret

> Stellt Konfigurationswerte und Zugangsdaten bereit, auf die Grafana beim Start zugreift.

ConfigMap

configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-config
  namespace: m347-grafana
data:
  grafana.ini: |
    [server]
    root_url = http://localhost:3000
    [security]
    admin_user = ${GF_SECURITY_ADMIN_USER}
    admin_password = ${GF_SECURITY_ADMIN_PASSWORD}
```

Erklärung der Konfiguration

- **kind: ConfigMap**
  Enthält Konfigurationsdateien für Grafana, hier speziell die `grafana.ini`.

- **grafana.ini**

  - **[server] root_url:**
    Legt die Basis-URL fest, unter der Grafana erreichbar ist (hier lokal auf Port 3000).
  - **[security] admin_user & admin_password:**
    Verwenden Umgebungsvariablen (`GF_SECURITY_ADMIN_USER`, `GF_SECURITY_ADMIN_PASSWORD`) für den Admin-Login.

Secret

secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: grafana-secret
  namespace: m347-grafana
type: Opaque
stringData:
  GF_SECURITY_ADMIN_USER: admin
  GF_SECURITY_ADMIN_PASSWORD: password
```

Erklärung der Konfiguration

- **kind: Secret**
  Speichert sensible Daten wie den Admin-Benutzernamen und das Passwort für Grafana.

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

  - **GF_SECURITY_ADMIN_USER:** Benutzername für den Admin-Login (hier „admin“).
  - **GF_SECURITY_ADMIN_PASSWORD:** Passwort für den Admin-Login (hier „password“).

Data Provisioning

> Definiert in YAML die automatische Anbindung von Prometheus als Datenquelle. Wird beim Start von Grafana erkannt und geladen.

data-provisioning.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-datasource
  namespace: m347-grafana
  labels:
    grafana_datasource: "1"
```

```
data:
  datasource.yaml: |
    apiVersion: 1
    datasources:
      - name: Prometheus
        type: prometheus
        url: http://prometheus-service.m347-prometheus.svc.cluster.local:80
        access: proxy
        isDefault: true
```

Erklärung der Konfiguration

- **labels: grafana_datasource: "1"**
  Markiert die ConfigMap als Datenquellen-Provisionierung für Grafana.
- **datasource.yaml**
  - **name: Prometheus:** Legt „Prometheus“ als Datenquelle fest.
  - **type: prometheus:** Datenquelle ist vom Typ Prometheus.
  - **isDefault: true:** Macht diese Datenquelle zur Standardquelle in Grafana.

Ingress / Externer Zugriff

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

Die Datei `grafana-ingress.yaml` definiert, unter welchem Hostnamen (`grafana.m347.ch`) Grafana 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 Grafana.
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 Nutzung von Grafana innerhalb des Kubernetes-Clusters erforderte eine genaue Konfiguration der Provisionierung von Datenquellen und Dashboards, um die automatische Einbindung von Prometheus als Datenquelle sicherzustellen. Eine wesentliche Herausforderung bestand darin, die persistente Speicherung von Benutzerkonfigurationen und Dashboards konsistent bereitzustellen und die Wiederherstellbarkeit bei Pod-Neustarts zu gewährleisten. Ausserdem stellte die Einrichtung der Zugriffsrechte über Secrets für den Admin-Benutzer eine wichtige Aufgabe dar, um die Sicherheit der Anwendung zu gewährleisten und unautorisierten Zugriff zu verhindern.