

Prometheus

Erstellt von Jonathan Al Kass & Johannes Olzem

Datum: 19.12.2025
Klasse: FI244
Dozent: Eray Sönmez

Inhaltsverzeichnis

1. Einleitung.....	1
2. Systemanforderungen.....	1
3. Unterschiede zu relationalen Datenbanken.....	2
4. Software- & Anwendungsbeispiel.....	2
5. Schnittstellen (APIs) für Programmiersprachen.....	4
5.1. Beispielabfrage.....	4
5.2. Antwort.....	4
5.3. Management API.....	4
6. Administration (Tools).....	5
7. Monitoring (Tools).....	5
8. Backup & Recovery.....	5
9. Vor- und Nachteile der Datenbanktechnologie.....	6
10. Fazit.....	6
11. Persönliche Meinung.....	6
Anhang.....	7

1. Einleitung

Moderne IT-Systeme erzeugen viele Messdaten. Diese Daten müssen gespeichert und ausgewertet werden. Dafür wird eine geeignete Datenbanktechnologie benötigt. Diese Dokumentation beschäftigt sich mit dem Monitoring-System Prometheus.

Prometheus sammelt Messwerte aus Anwendungen und Systemen. Die Daten werden in einer Zeitreihendatenbank gespeichert. So kann der Zustand eines Systems über einen längeren Zeitraum beobachtet werden. Prometheus wird häufig zur Überwachung von IT-Infrastrukturen eingesetzt.

Ziel dieser Dokumentation ist es, die Funktionsweise von Prometheus verständlich zu erklären. Außerdem werden die wichtigsten Vorteile des Systems vorgestellt. Der Leser soll ein grundlegendes Verständnis für die Technologie erhalten und ihren Einsatz im Monitoring nachvollziehen können.

2. Systemanforderungen

Der Prometheus Server besteht aus mehreren Bestandteilen:

1. Time Series Database (TSDB)
Hier werden die Zeitreihen-Daten gespeichert
2. HTTP-Server
Prometheus bringt seinen eigenen HTTP Server mit Weboberfläche auf welcher man z.B. Daten abfragen und Endpunkte konfigurieren kann.
3. Retrieval Komponente
Hier werden Metrik-Daten eingesammelt und in der TSDB gespeichert

Dieser kann entweder auf dem System selbst oder in einem Container (z.B. Docker) installiert werden.

Von den Entwicklern sind keine spezifischen Systemanforderungen angegeben.

3. Unterschiede zu relationalen Datenbanken

Prometheus	Relationale Datenbanken
Für Monitoring & Observability	Zur persistenten Datenspeicherung
Speichert Metriken	Speichert Geschäftsdaten
Zeitreihen Datenmodell	Tabellen mit Zeilen & Spalten
Keine Joins, keine Fremdschlüssel	Normalisierung, Relationen & Joins
PromQL	SQL
Optimiert für viel Schreiben & regelmäßiges lesen	Ausgewogenes Lesen & Schreiben
Pull: Prometheus fragt Daten ab	Push: Applikation schreibt Daten
Daten werden regelmäßig überschrieben	Daten bleiben bis zum löschen bestehen
Keine Transaktionen	ACID-Prinzip
Kleine Inkonsistenzen akzeptabel	Hohe Datenintegrität

4. Software- & Anwendungsbeispiel

Eine Beispielhafte Anwendung von Prometheus könnte die folgende Situation darstellen:

In einem heimischen Netzwerk hat ein Anwender ein Netzwerk bestehend aus einem Client (10.42.0.12) und einem Heimserver System (10.42.0.13). Auf diesem System läuft die Prometheus Datenbank, sowie Grafana zur Visualisierung und PiHole (ein DNS-Werbeblocker).

Prometheus sammelt hierbei Metriken zum eigenen Host mit dem Programm "node-exporter", Daten zu PiHole mit dem Programm "pihole-exporter" und Netzwerkdaten/ -statistiken zum Client und zu sich selbst über das Simple Network Management Protocol mit dem Programm "snmp_exporter". Diese Daten werden dann in Grafana visualisiert und in einzelne Dashboards aufgeteilt.

Um diese Services einfach zu verwalten werden diese mithilfe einer docker-compose.yml Datei dokumentiert und dann mit dem Kommando `docker-compose up -d` gestartet. Die verwendete Datei befindet sich im Anhang.

Um die Metriken nun in Prometheus einzubinden wird der Server mit der `prometheus.yml` wie folgt konfiguriert:

```
scrape_configs:
  - job_name: 'prometheus'           # Metriken über Prometheus
    static_configs:
      - targets: ['127.0.0.1:9090']

  - job_name: 'pihole'              # Metriken über PiHole
    static_configs:
      - targets: ['127.0.0.1:9617']

  - job_name: 'node-exporter'        # Metriken zum System
    static_configs:
      - targets: ['127.0.0.1:9100']

  - job_name: 'snmp'
    static_configs:
      - targets:                       # Metriken über Server/Client
        - 10.42.0.12
        - 127.0.0.1
    metrics_path: /snmp
    params:
      auth: [public_v2]              # Nutze SNMP community "public"
      module: [if_mib]
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: 127.0.0.1:9116 # Endpunkt des SNMP-Exporters
```

Um den SNMP-Exporter zu konfigurieren wird die offizielle snmp.yml Datei der Entwickler verwendet ([hier](#) zu finden).

Nachdem die Docker Container mit den neuen Konfigurationsdateien neu gestartet wurden, findet sich unter <http://localhost:9090/> die Weboberfläche von Prometheus und unter <http://localhost:3000/> die Weboberfläche von Grafana. Hier können nun die einzelnen Dashboards (z.B. mithilfe von Templates) eingerichtet werden.

5. Schnittstellen (APIs) für Programmiersprachen

Die Querying API findet sich unter dem Endpunkt `/api/v1`. Eine Beispielabfrage wäre: `GET /api/v1/query?query=go_threads`. Die API Antwortet im JSON Format mit den Keys `"status"`, welche entweder `"success"` oder `"error"` sein kann und `"data"` welche den auszugebenden Datentyp und die abgefragten Daten enthält, sofern die Abfrage korrekt war.

5.1. Beispielabfrage

```
GET /api/v1/query?query=go_threads{instance="localhost:9090"}
```

Gibt die Anzahl der zugeteilten Threads aus.

5.2. Antwort

```
{
  "status": "success",          # "success" oder "error"
  "data": {
    "resultType": "vector",    # Datentyp
    "result": [
      {
        "metric": {            # Daten der abgefragten Metrik
          "__name__": "go_threads",
          "instance": "localhost:9090",
          "job": "prometheus"
        },
        "value": [
          1765871138.435,      # Zeitpunkt des Datenpunkts
          "26"                 # Wert
        ]
      }
    ]
  }
}
```

5.3. Management API

Die Management API hat lediglich 4 Endpunkte:

- `GET /-/healthy` welcher immer 200 ausgibt solange Prometheus online ist
- `GET /-/ready` welcher 200 ausgibt, sobald Prometheus bereit für Abfragen ist
- `PUT /-/reload` Ist standardmäßig deaktiviert. Lädt Konfigurationen neu.
- `PUT /-/quit` Ist standardmäßig deaktiviert. Führt Server ordnungsgemäß herunter.

6. Administration (Tools)

Die Konfiguration der Prometheus Installation erfolgt mit YML. Die komplette bestehende Konfiguration lässt sich in der Weboberfläche unter "Status > Configuration" anzeigen. Sie liegt (im Docker Container) unter `/prometheus/prometheus.yml`. Hier werden sowohl globale Konfigurationsoptionen festgelegt, wie z.B. `scrape_interval` als auch die Targets (→ Datenquellen). Die Konfiguration um sich selbst einzubinden sieht wie folgt aus:

```
scrape_configs:
  - job_name: 'prometheus'                # Name des Targets
    static_configs:
      - targets: ['127.0.0.1:9090'] # Host des Targets
```

Um z.B. die Aufbewahrungszeit der Daten auf 5 Tage einzustellen nutzt man folgendes:

```
storage:
  tsdb:
    retention:
      time: 5d
```

7. Monitoring (Tools)

Da Prometheus hauptsächlich für Metriken und Monitoringdaten ausgelegt ist, kann es sich auch selbst monitoren. Dafür steht unter `/metrics` ein eigenes Target bereit, welches sich wie in „Administration (Tools)“ beschrieben einbinden lässt. Hier finden sich Metriken wie z.B. wie viele HTTP-Requests gesendet wurden, wie viele Threads verfügbar sind oder wie viel Arbeitsspeicher von Prometheus gebraucht wird.

8. Backup & Recovery

Um ein Backup der Prometheus Datenbank zu erstellen werden Snapshots empfohlen. Ein Snapshot lässt sich per API Call am Endpunkt `POST /api/v1/admin/tsdb/snapshot` erstellen und man erhält einen Dateipfad zurück unter welchem der Snapshot gespeichert wurde.

Diese Admin API muss freigeschaltet werden, indem Prometheus mit der Option `--web.enable-admin-api` ausgeführt wird.

Die Inhalte des erstellten Snapshot Ordners können nun auf einem Backupsystem gespeichert werden und zur Wiederherstellung in das `/prometheus` Verzeichnis kopiert werden.

9. Vor- und Nachteile der Datenbanktechnologie

Vorteile	Nachteile
Schnelle (& Automatische) Datenerfassung	Steile Lernkurve bei PromQL
Geringe Latenz bei Abfragen	Konfiguration erfordert Erfahrung
Komplexe Analysen möglich	Hoher Ressourcenverbrauch bei hohen Datenmengen
Integration mit vielen Exportern	Langzeitarchivierung erfordert zusätzliche Komponenten
Horizontal Skalierbar	Keine eingebaute Authentifizierung
Unterstützt große Systemlandschaften	

10. Fazit

Prometheus ist ein modernes und leistungsfähiges Monitoring-System zur Erfassung von Messdaten. Es eignet sich besonders gut zur Überwachung von IT-Systemen mit ein oder mehreren Services. Durch das Zeitreihen-Datenmodell und die gute Zusammenarbeit mit Tools wie Grafana lassen sich Systemzustände übersichtlich darstellen. Auch wenn die Einrichtung und die Abfragesprache etwas komplex sein können, bietet Prometheus insgesamt viele Vorteile für das Monitoring.

11. Persönliche Meinung

Unserer Meinung nach ist Prometheus ein sehr gutes Werkzeug zur Überwachung von IT-Infrastrukturen. Besonders positiv finden wir die automatische Erfassung der Daten und die vielen Erweiterungsmöglichkeiten durch Exporter. Der Einstieg ist zwar nicht ganz einfach, aber nach kurzer Einarbeitung ist das System gut nutzbar und sehr hilfreich.

Nachdem diese Einführung nun im Rahmen der Ausarbeitung dieser Dokumentation erfolgt ist, werden wir Prometheus wahrscheinlich auch in unserem privaten Umfeld einrichten und verwenden.

Anhang

docker-compose.yml

```
# Persistenter Datenspeicher zwischen Containern
volumes:
  prometheus-data:
  grafana-data:

services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    network_mode: host # Nutze Netzwerkkarte des Servers
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-data:/prometheus
    restart: unless-stopped

  pi-hole:
    container_name: pi-hole
    image: pi-hole/pi-hole:latest
    ports:
      - "53:53/tcp"
      - "53:53/udp"
      - "4000:80"
      - "4443:443"
    environment:
      TZ: 'Europe/Berlin'
      FTLCNF_webserver_api_password: 'admin'
      FTLCNF_dns_listeningMode: 'ALL'
    cap_add:
      - NET_ADMIN
      - SYS_TIME
    restart: unless-stopped

  node-exporter:
    image: quay.io/prometheus/node-exporter
    container_name: node_exporter
    command:
      - '--path.rootfs=/host'
    network_mode: host
    pid: host
    restart: unless-stopped
```

```
volumes:
  - '/:/host:ro,rslave'

pihole-exporter:
  container_name: pihole-exporter
  image: ekofr/pihole-exporter
  environment:
    - PIHOLE_HOSTNAME=10.42.0.13
    - PIHOLE_PASSWORD=admin
    - PIHOLE_PORT=4000
    - PORT=9617
  network_mode: host
  restart: unless-stopped

snmp-exporter:
  container_name: snmp-exporter
  image: prom/snmp-exporter:v0.29.0
  volumes:
    - ./snmp.yml:/etc/snmp_exporter/snmp.yml:ro
  command:
    - '--config.file=/etc/snmp_exporter/snmp.yml'
  network_mode: host
  restart: unless-stopped

grafana:
  image: grafana/grafana
  container_name: grafana
  ports:
    - "3000:3000"
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=admin
  volumes:
    - grafana-data:/var/lib/grafana
  restart: unless-stopped
```

prometheus.yml

```
scrape_configs:
  - job_name: 'prometheus'          # Metriken über Prometheus
    static_configs:
      - targets: ['127.0.0.1:9090']

  - job_name: 'pihole'              # Metriken über PiHole
    static_configs:
      - targets: ['127.0.0.1:9617']

  - job_name: 'node-exporter'        # Metriken zum System
    static_configs:
      - targets: ['127.0.0.1:9100']

  - job_name: 'snmp'
    static_configs:
      - targets:                      # Metriken über Server/Client
        - 10.42.0.12
        - 127.0.0.1
    metrics_path: /snmp
    params:
      auth: [public_v2]              # Nutze SNMP community "public"
      module: [if_mib]
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: 127.0.0.1:9116  # Endpunkt des SNMP-Exporters
```