Dokumentation der verteilten Systeme – Modul 321

Inhalt

1 Ausgangslage

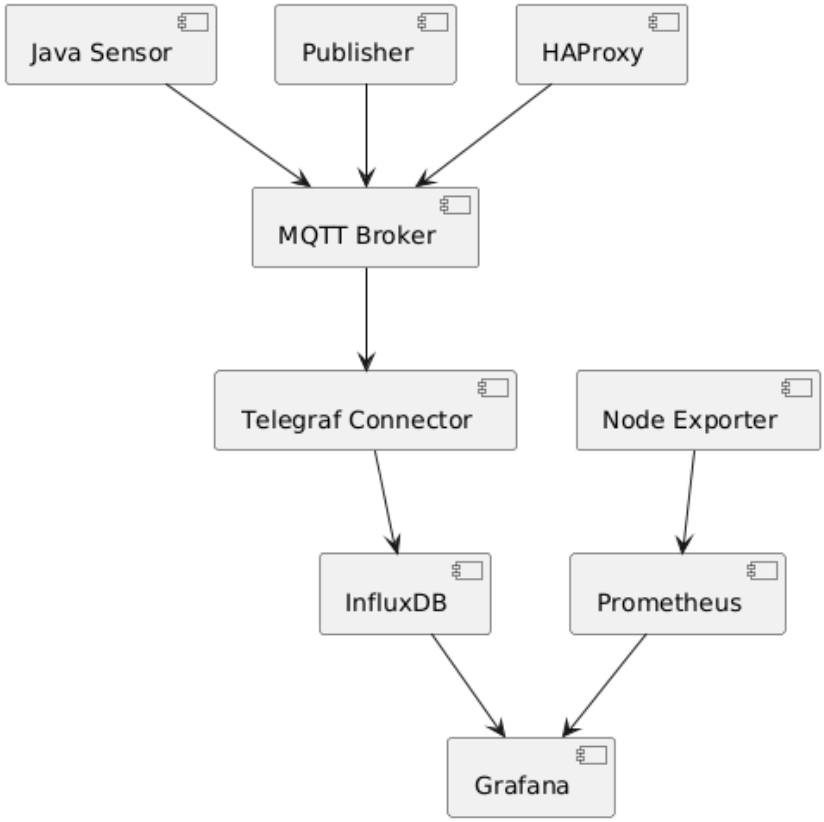
Im Modul 321 wurden mehrere eigenständige Übungen (ZP 1 – 3, SideQuests) umgesetzt. Dabei entstanden diverse Container-basierte Services – darunter MQTT-basierte Sensor-Pipelines, ein Load-Balancer sowie ein Prometheus-Monitoring. Diese Dokumentation führt alle Einzelsysteme zusammen und zeigt, wie sie als verteiltes Gesamtsystem zusammenspielen.

2 Systeme & Dienste

Nr.	System / Dienst	Aufgabe
1	**Docker Engine**	Container-Runtime für sämtliche Komponenten
2	**MQTT Broker (Mosquitto)**	Zentrale Nachrichtenvermittlung der Sensor-Daten
3	**Java-Sensoren**	Publizieren Messwerte (Temperatur, Feuchtigkeit …) über MQTT
4	**Publisher**	Erzeugt Test-Nachrichten zur Last-/Funktions-prüfung
5	**InfluxDB**	Zeitreihen-Datenbank für Sensorwerte
6	**Telegraf (MQTT→Influx Connector)**	Konsumiert MQTT-Topics & schreibt in InfluxDB
7	**Grafana**	Visualisiert Influx- und Prometheus-Daten
8	**Prometheus**	Monitoring- & Metrik-Storage für Infrastruktur
9	**Node Exporter**	Liefert Host-Metriken an Prometheus
10	**HAProxy**	Verteilt eingehende MQTT-Publishes (Round Robin) auf mehrere Broker

3 Architektur (UML)

3.1 Komponenten-Diagramm

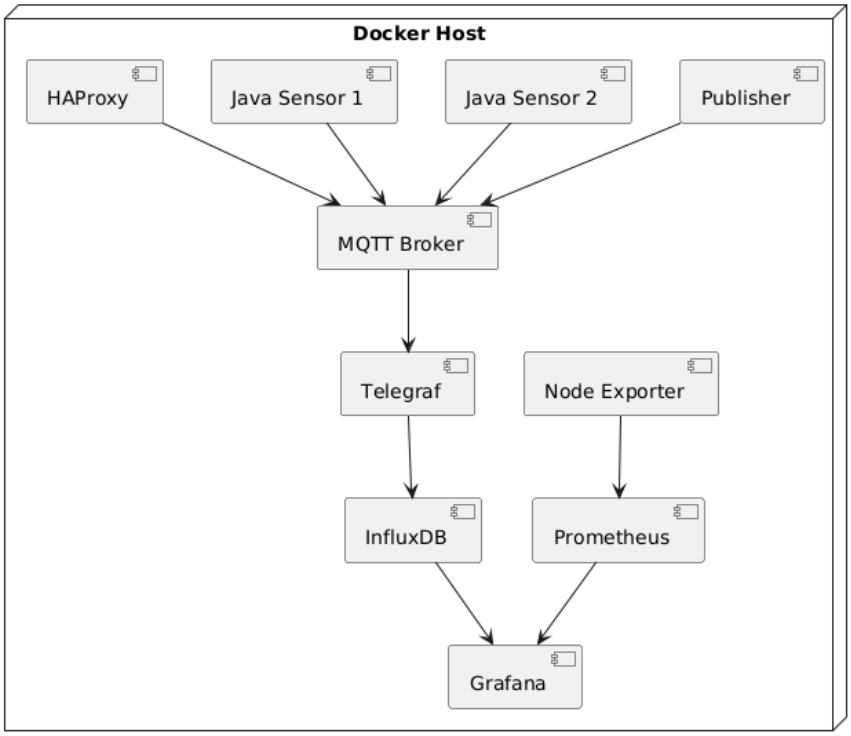


Das Komponenten-Diagramm veranschaulicht die logische Architektur des verteilten Systems. Es zeigt die Hauptkomponenten und deren Beziehungen zueinander:

- Java Sensor, Publisher und HAProxy kommunizieren mit dem MQTT Broker als zentrale Nachrichtenvermittlungsinstanz
- Der MQTT Broker leitet Daten an den Telegraf Connector weiter, der diese für die InfluxDB aufbereitet
- Parallel dazu liefert der Node Exporter Systemmetriken an Prometheus
- Sowohl InfluxDB (für Sensordaten) als auch Prometheus (für Systemmetriken) speisen ihre Daten in Grafana ein, das als zentrale Visualisierungsplattform dient

Diese Struktur ermöglicht eine klare Trennung zwischen Datenerfassung, -verarbeitung und -visualisierung.

3.2 Deployment-Diagramm



Das Deployment-Diagramm stellt die physische Verteilung der Komponenten dar und zeigt, wie sie in der Docker-Umgebung installiert sind:

- Alle Komponenten laufen innerhalb eines Docker Hosts
- Mehrere Java Sensor-Instanzen (1 und 2) sowie Publisher und HAProxy kommunizieren mit dem MQTT Broker
- Die Datenverarbeitungskette vom MQTT Broker über Telegraf nach InfluxDB bleibt erhalten
- Ebenso bleibt die Metrikkette vom Node Exporter über Prometheus zu Grafana bestehen

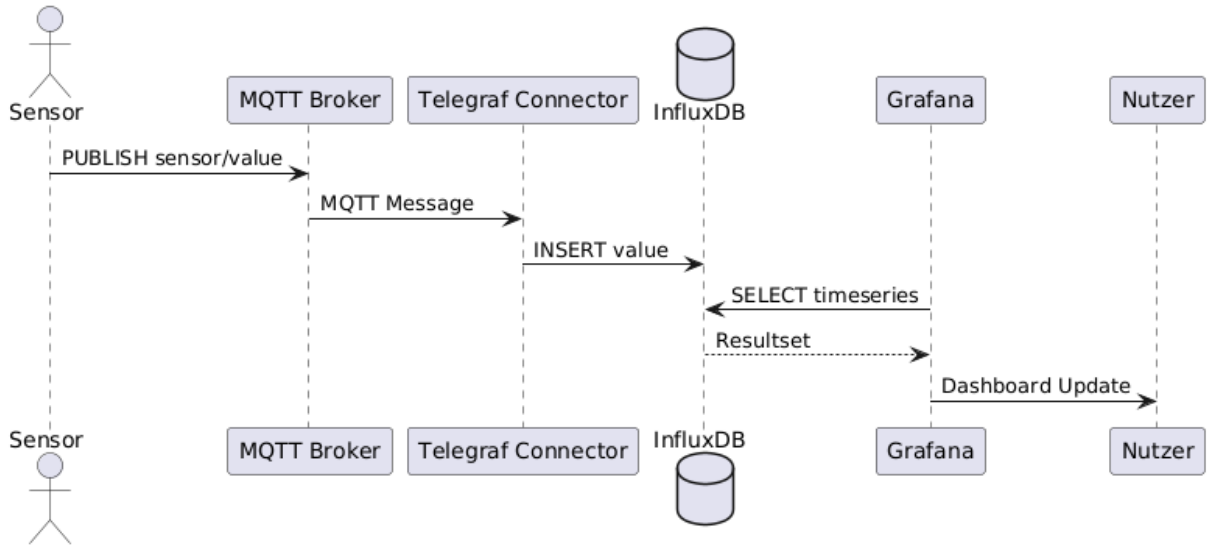
Dieses Diagramm verdeutlicht den containerisierten Ansatz, bei dem alle Dienste auf demselben physischen Host laufen, aber in isolierten Docker-Containern.

4 Detailbeschreibung der Interaktionen

#	Interaktion	Beschreibung
1	**Sensor → Broker**	Jeder Sensor veröffentlicht JSON-Payload (Topic `sensors/<type>`) im 2-Sek-Takt.
2	**Broker → Telegraf**	Telegraf abonniert `sensors/#`, parst Payload & mappt Felder auf Influx-Measurements.
3	**Telegraf → InfluxDB**	Influx-Line-Protocol-Writes – 1 Messung ≈ 1 Datensatz.
4	**InfluxDB → Grafana**	Grafana-Dashboard ruft Zeitreihen ab (Flux/SQL).
5	**Node Exporter → Prometheus**	Host-Metriken (CPU, RAM, Disk) werden jede 15 s gescraped.
6	**Prometheus → Grafana**	Zweites Dashboard zeigt Server-Health-Metriken.
7	**HAProxy → MQTT Broker-Cluster**	Round-Robin verteilt Publishes; Fallback bei Broker-Ausfall.

5 End-to-End Ablauf

Sequenzdiagramm



Das Sequenzdiagramm zeigt den End-to-End-Ablauf der Datenverarbeitung im zeitlichen Verlauf:

1. Der Sensor veröffentlicht Messdaten über PUBLISH an den MQTT Broker
2. Der MQTT Broker leitet die Nachricht an den Telegraf Connector weiter
3. Der Telegraf Connector fügt die Daten per INSERT in die InfluxDB ein
4. Grafana fragt die Zeitreihen mit SELECT aus der InfluxDB ab
5. Die InfluxDB liefert das Resultset zurück an Grafana
6. Grafana aktualisiert das Dashboard für den Nutzer

Dieses Diagramm verdeutlicht den Datenfluss vom Sensor bis zur Benutzeroberfläche und zeigt präzise, wie die verschiedenen Komponenten im zeitlichen Ablauf miteinander interagieren.