Realtime Dashboards

Streaming Data und Visualisierung

Christian Bockermann

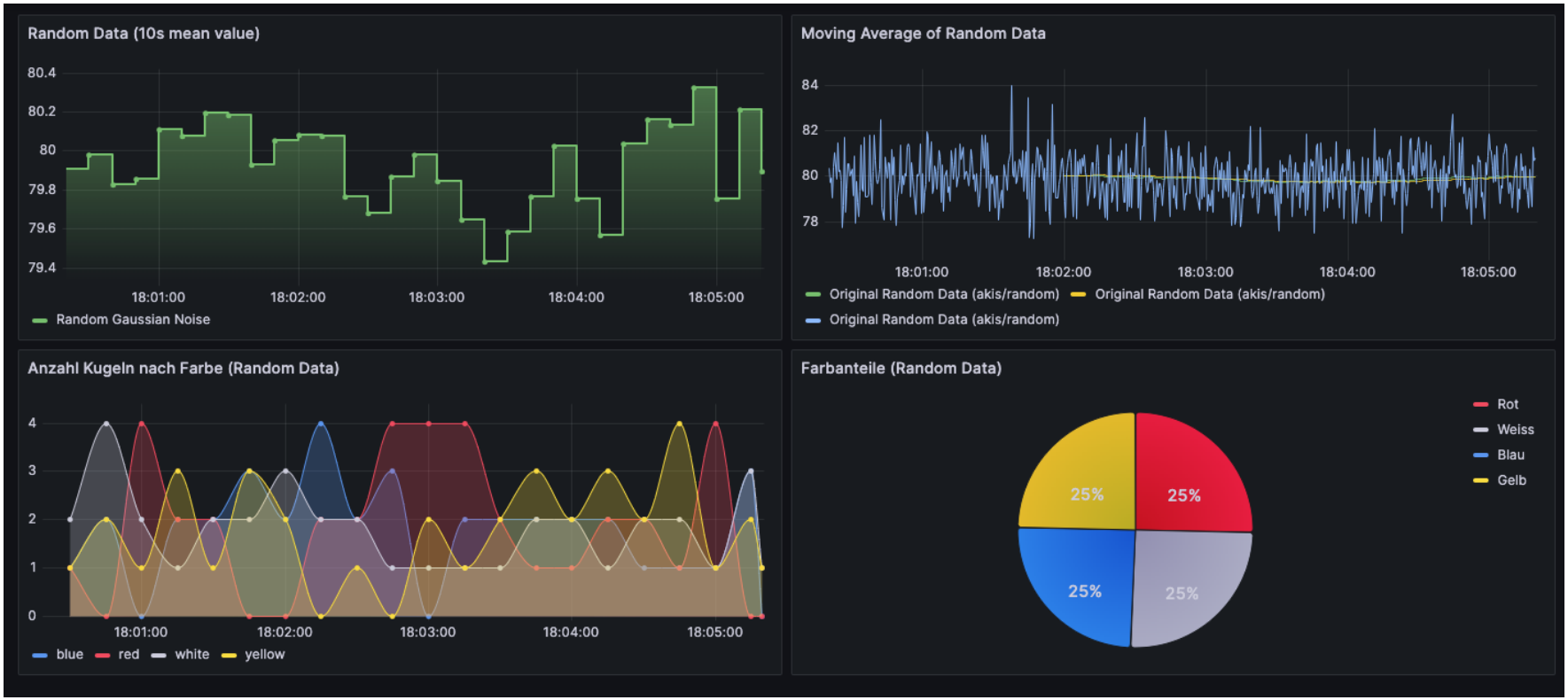
Hochschule Bochum

AKIS Sommersemester 2023

Agenda

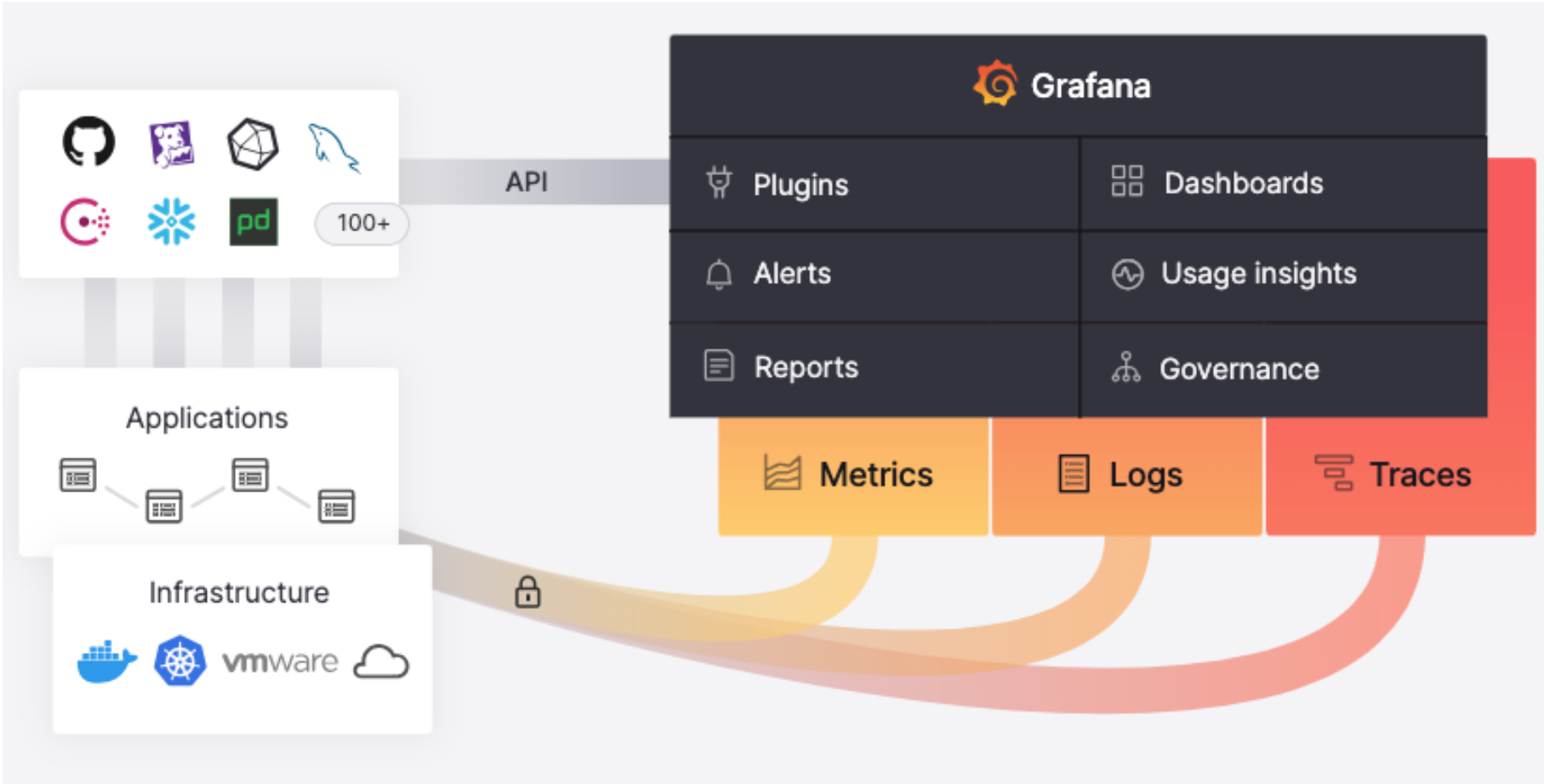
1. Queen of Dashboards: Grafana
2. Groundwork: Data Streaming mit MQTT
3. Zeitreihen – InfluxDB
4. Beispiel: LEGO Fabrik

Daten-Dashboards mit Grafana



Grafana

- Open Source Web-Anwendung
- Erstellung von Dashboards (Menge von Diagrammen)
- Veröffentlicht 2014
- Implementierung in Go
- Automatischer Reload der Daten, Aktualisierung
- API + Plugins zur Anbindung von Datenquellen


Grafana
Plugins
Dashboards
Alerts
Usage insights
Reports
Governance
API
100+
Applications
Infrastructure
vmware
Metrics
Logs
Traces

Tools um Grafana

- Einsatz häufig mit Tools wie z.B. Prometheus
- Prometheus ist ein Metric-Sammler für IT Infrastruktur

- Promtail ist ein Log-File Collector Agent
- Mit Promtail werden Logs an Grafana Loki gepusht

- Grafana Loki ist ein Log-File Store
- Loki sammelt und indiziert Logs für Grafana

Data Sources

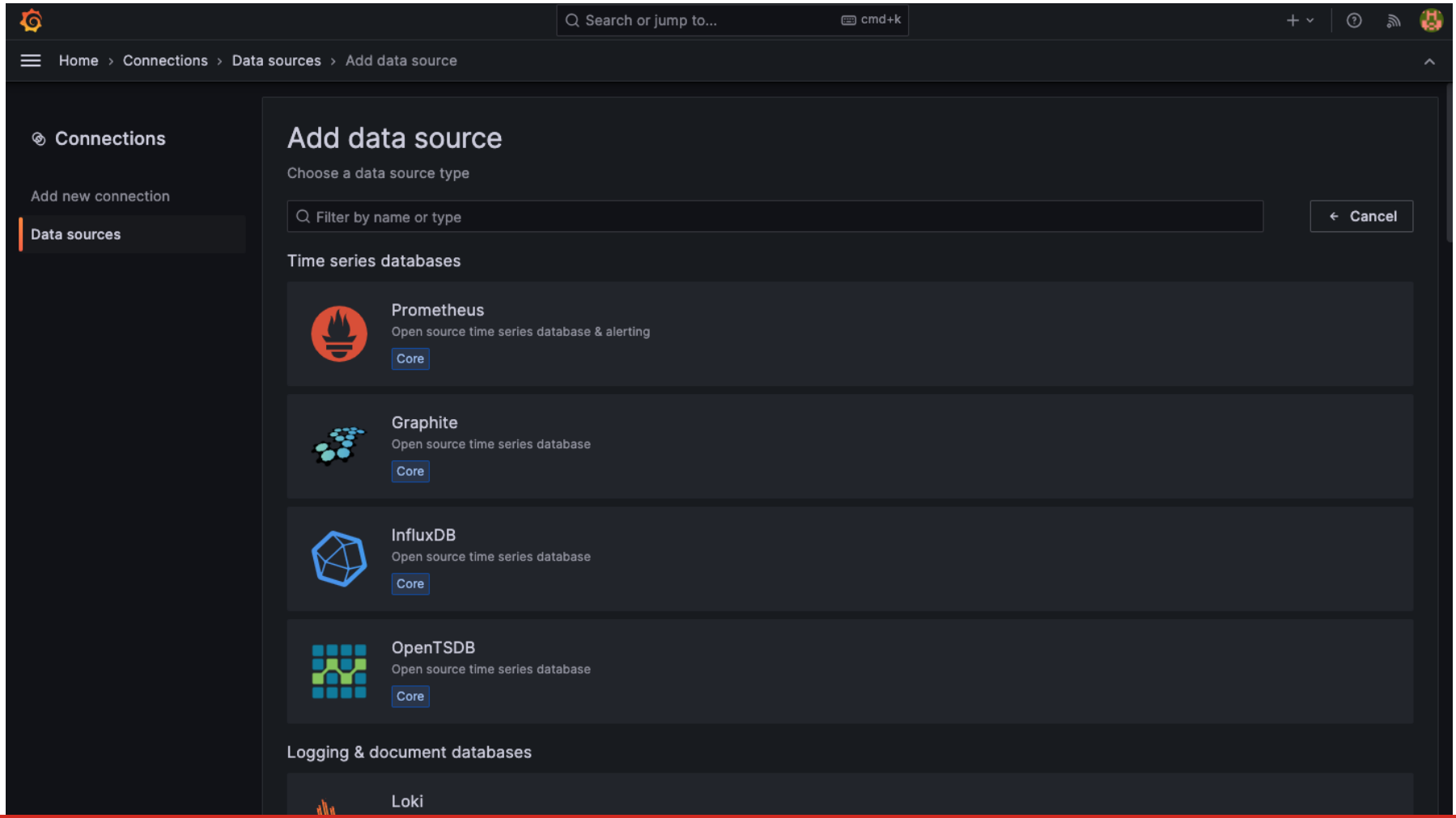
Datenquellen

- Grafana visualisiert Daten ans verschiedenen Quellen
- Über Plugins, etc. lassen sich unterschiedliches Quellen anbinden

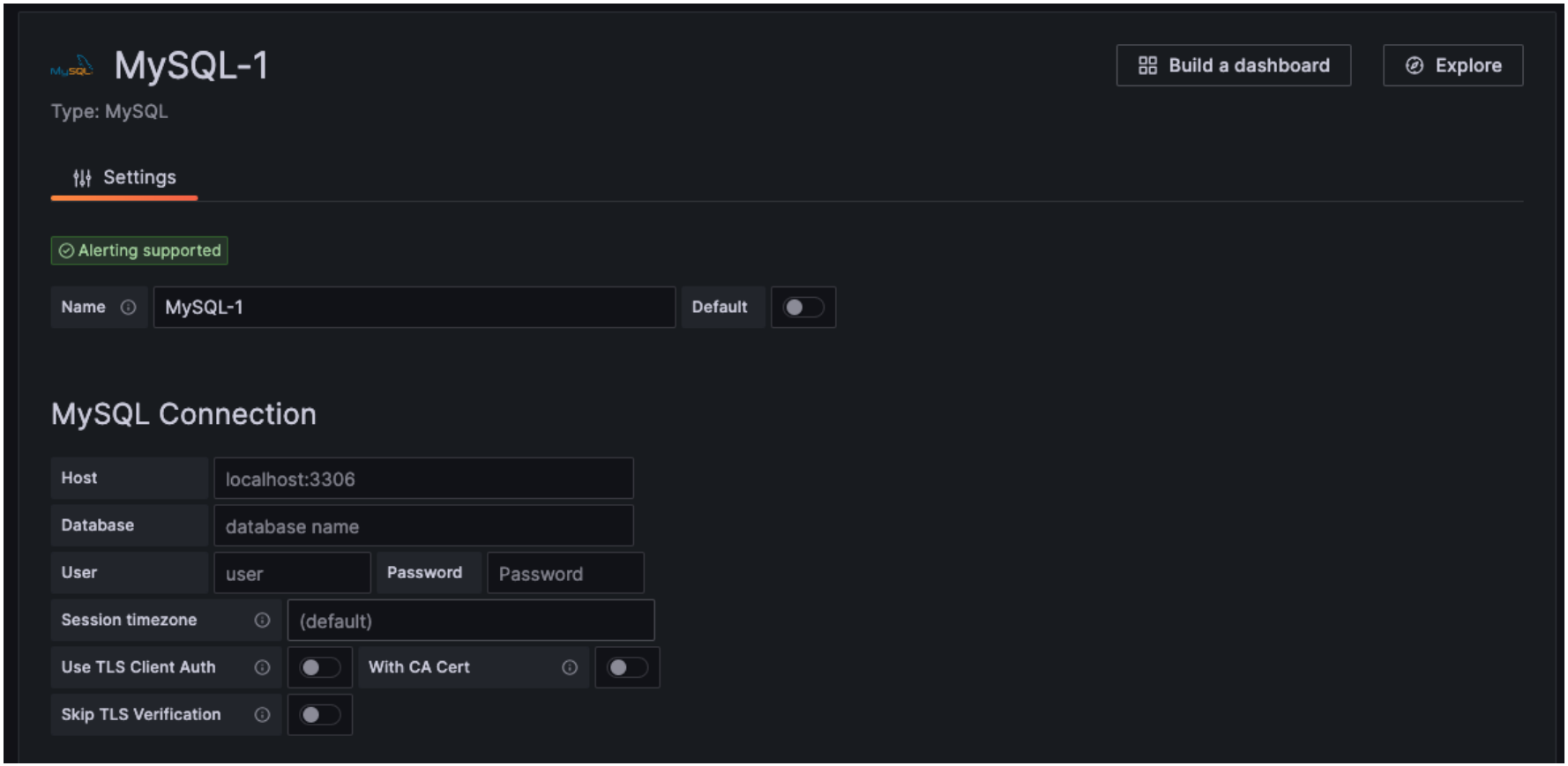
Beispiele:

- SQL Datenbanken (SQL Server, MySQL, ...)
- ElasticSearch
- JIRA, SAP HANA
- Snowflake, MongoDB, Splunk
- MQTT

Datenquellen



Beispiel: MySQL Datenbank als Datenquelle

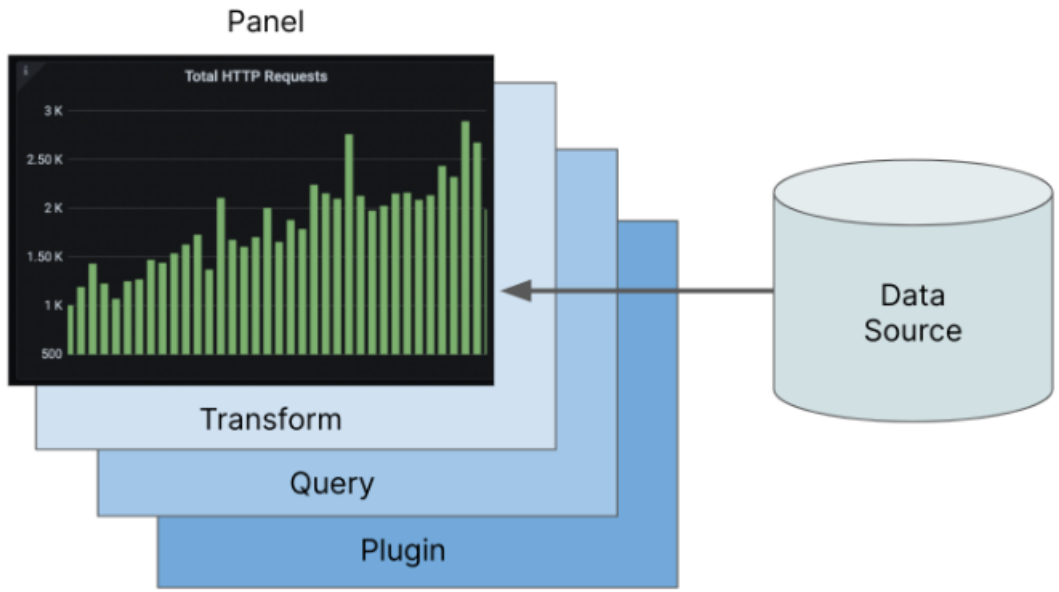


Grafana Dashboard

- Menge von Visualisierungen
- Interaktive Erstellung/Konfiguration im Browser
- Jede Visualisierung ist mit einer oder mehreren Datenquellen verknüpft
- Definition eines Dashboards als JSON-Datei

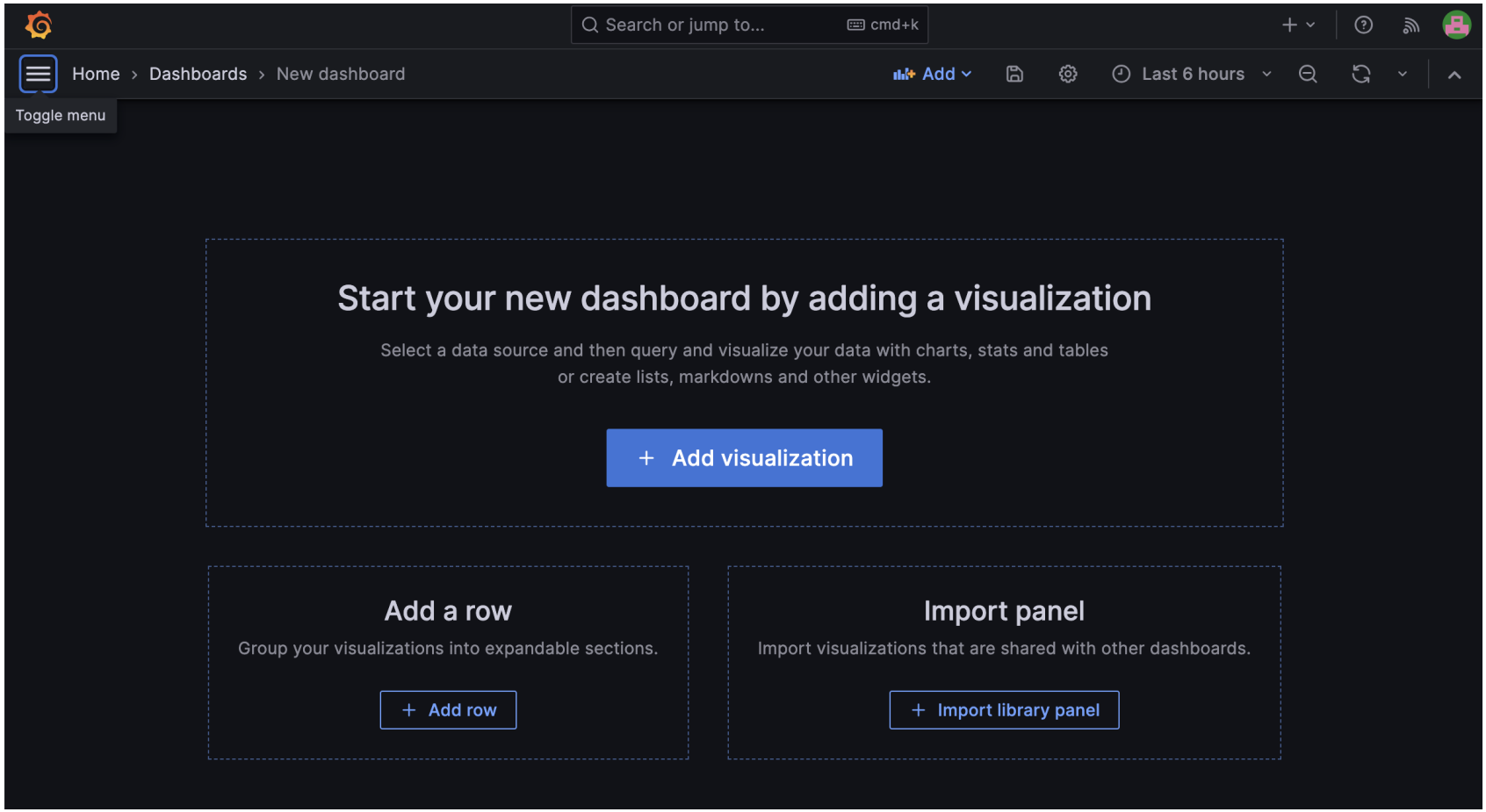
Grafana Dashboard System

- Visualisierung ist ein Panel
- Über Plugin mit der Datenquelle verbunden



Grafik von https://grafana.com/docs/grafana/latest/fundamentals/dashboards-overview/

Neues Dashboard erstellen



Beispiel: Tankpreise

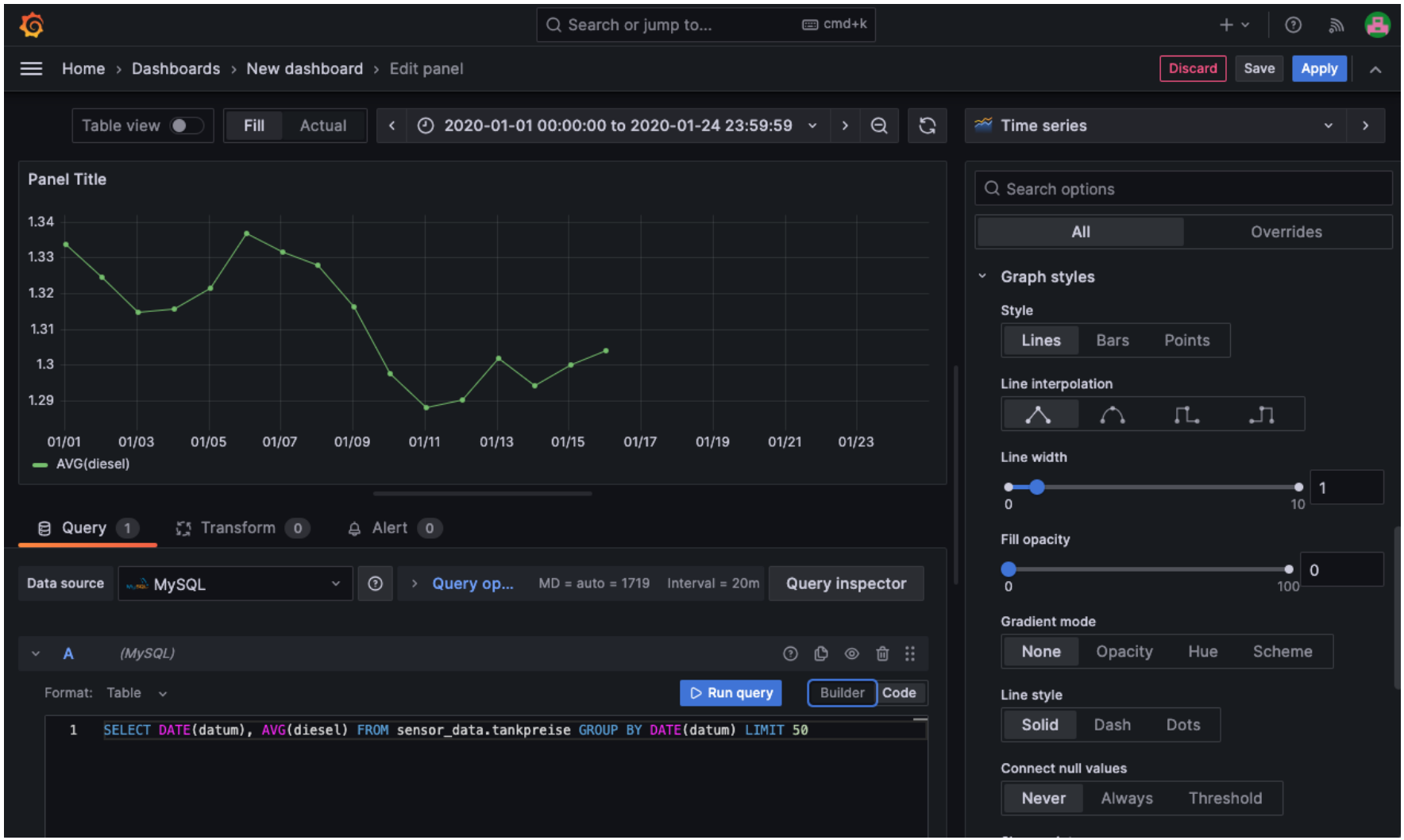
	date	diesel	e5	e10
2020-01-01	02:03:03+01	1.289	1.409	1.369
2020-01-01	04:36:04+01	1.309	1.429	1.409
2020-01-01	04:36:04+01	1.319	1.439	1.399
2020-01-01	04:36:04+01	1.319	1.439	1.419
2020-01-01	04:36:04+01	1.319	1.439	1.419

Tagesdurchschnitt Diesel-Preis

```
SELECT DATE(date),AVG(diesel)
  FROM tankpreise GROUP BY DATE(date)
```

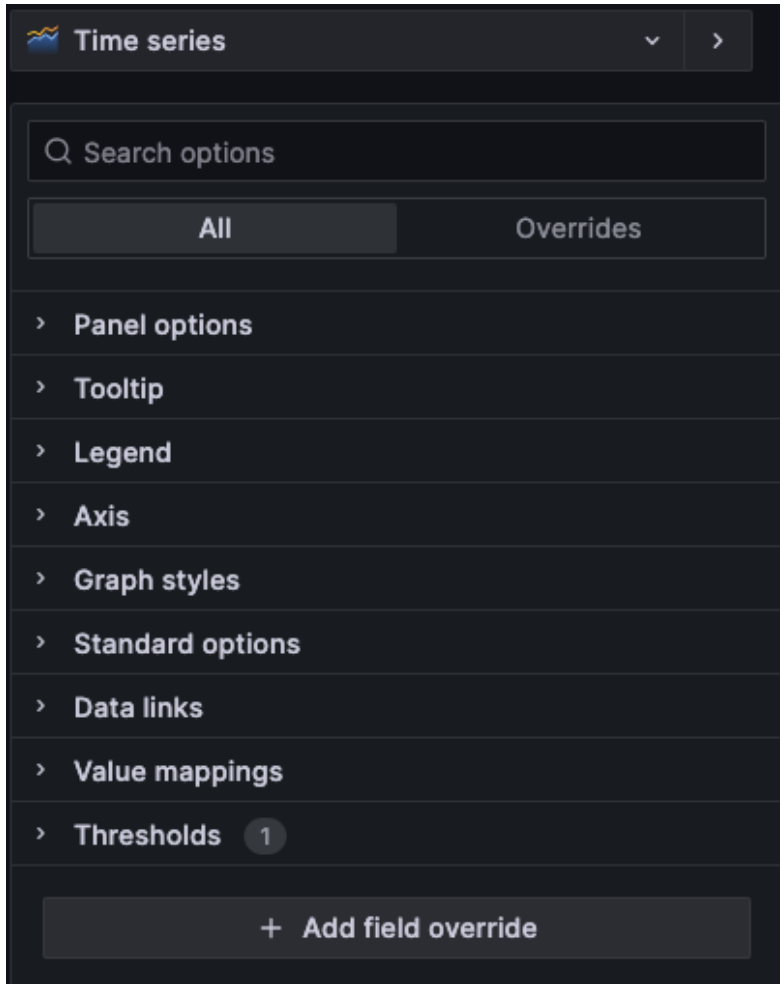
DATE(datum)	AVG(diesel)
2020-01-01	1.333745076586427
2020-01-02	1.3245752988047679
2020-01-03	1.3149396771452728
2020-01-04	1.3156230366492003
2020-01-05	1.3216437098255163
2020-01-06	1.3367058383233381
2020-01-07	1.331616418966719
2020-01-08	1.3280095029239645
2020-01-09	1.3163894230769149

Visualisierung für Tankspreise (Durchschnitt Diesel-Preis)

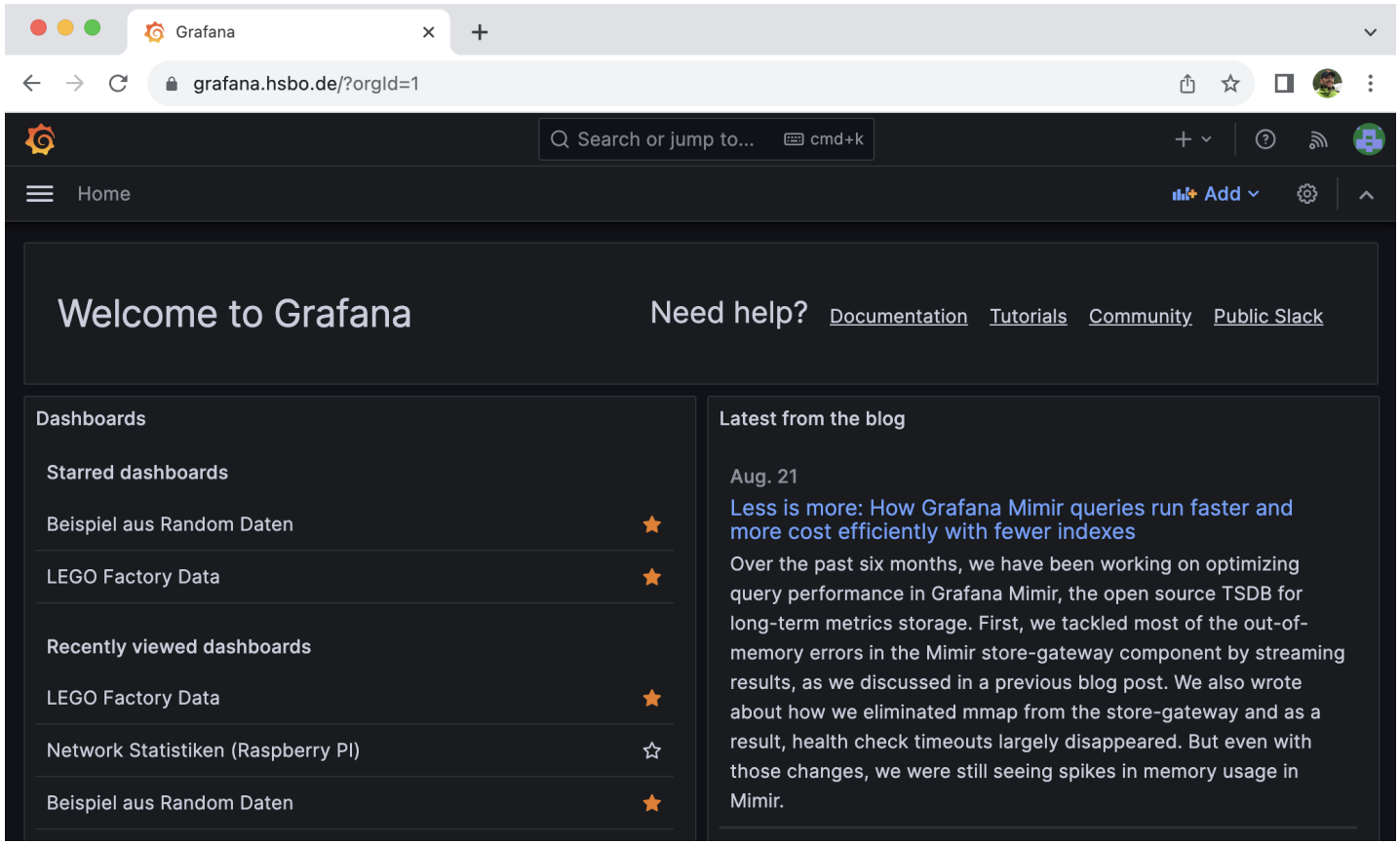


Optionen für Visualisierung

- Jede Visualisierung hat Darstellungsoptionen
- Farben, Linien, Legende,...



Grafana: https://grafana.hsbo.de



Benutzername: `akis` Kennwort: `summerschool2023`

Was wollen wir eigentlich visualisieren?

Was wird visualisiert?

Was wollen wir eigentlich visualisieren?

- Realtime Daten - was ist das genau?
- Wo entstehen die Daten?
- Was müssen wir tun, damit Realtime auch Sinn macht?

Was wird visualisiert?

Was wollen wir eigentlich visualisieren?

- Realtime Daten - was ist das genau?
- Wo entstehen die Daten?
- Was müssen wir tun, damit Realtime auch Sinn macht?

Zwei Fragen:

1. Wie läuft die Verarbeitung bis zur Visualisierung?
2. Wie speichern wir Zeitreihen-Daten am besten?

Groundwork: Data Streaming mit MQTT

Kontinuierliche Datenströme

- Viele Sensoren liefern kontinuierliche Meßwerte
- Zeitreihe = Sammlung gleich strukturierter Meßwerte

Kontinuierliche Datenströme

- Viele Sensoren liefern kontinuierliche Meßwerte
- Zeitreihe = Sammlung gleich strukturierter Meßwerte

Messaging Systeme als Grundlage

- Systeme kommunizieren über Nachrichten
- Nachrichten werden über Kanäle (topic) geschickt
- Systeme können Kanäle abbonieren

Beispiel: Analyse von Studierenden-Zahlen


```
{id:252,date:'2022-12-06 09:47',room:'AW-01-39',value:+1}
```

Beispiel: Analyse von Studierenden-Zahlen


```
{id:252,date:'2022-12-06 09:47',room:'AW-01-39',value:+1}
```

Ziele:

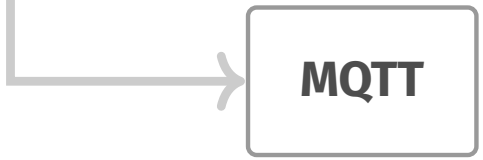
- Echtzeit-Darstellung der Raumbelegung
- Möglichkeit zur Analyse historischer Daten

Beispiel: Analyse von Studierenden-Zahlen


```
{id:252,date:'2022-12-06 09:47',room:'AW-01-39',value:+1}
{id:253,date:'2022-12-06 09:48',room:'AW-01-39',value:-1}
{id:254,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:255,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:256,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:258,date:'2022-12-06 09:52',room:'AW-01-39',value:+1}
```

Beispiel: Analyse von Studierenden-Zahlen

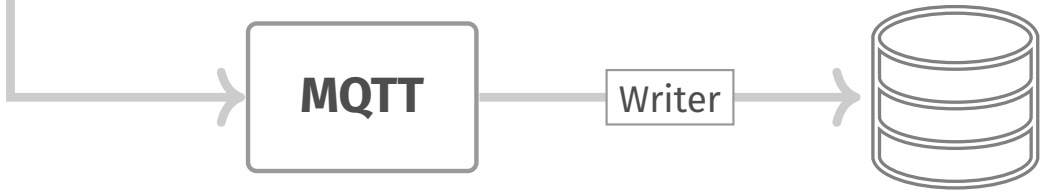

```
{id:252,date:'2022-12-06 09:47',room:'AW-01-39',value:+1}
{id:253,date:'2022-12-06 09:48',room:'AW-01-39',value:-1}
{id:254,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:255,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:256,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:258,date:'2022-12-06 09:52',room:'AW-01-39',value:+1}
```



Broker

Beispiel: Analyse von Studierenden-Zahlen


```
{id:252,date:'2022-12-06 09:47',room:'AW-01-39',value:+1}
{id:253,date:'2022-12-06 09:48',room:'AW-01-39',value:-1}
{id:254,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:255,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:256,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:258,date:'2022-12-06 09:52',room:'AW-01-39',value:+1}
```

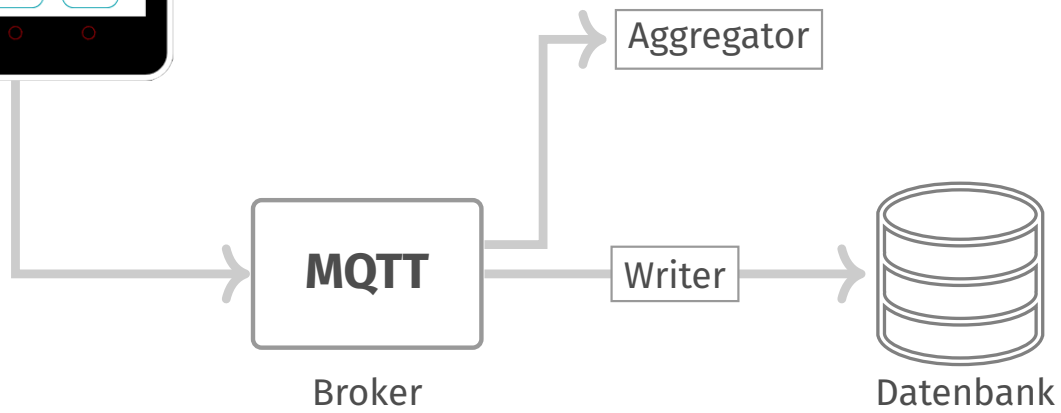


Broker

Datenbank
(= historischer Speicher)

Beispiel: Analyse von Studierenden-Zahlen

Raum	Anzahl
AW 01-39	11
AW 1-33	15
AW 5-24	6



Nachrichtenverarbeitung im Aggregator

```
def on_message(client, userdata, msg):
    print(f"message from ‘{msg.topic}‘ topic")
    global counts
    data = loads(msg.payload.decode())
    room = data['room']

    if room not in counts:
        counts[room] = 1
    else:
        counts['room'] = counts['room'] + 1
```

Kanal/Topic abonnieren und verarbeiten

```
# Topic, auf dem Messwerte publiziert werden
topic = 'counts'

mqtt.subscribe(topic)
mqtt.on_message = on_message

mqtt.loop_forever()
```

Anderer Broker: Apache Kafka

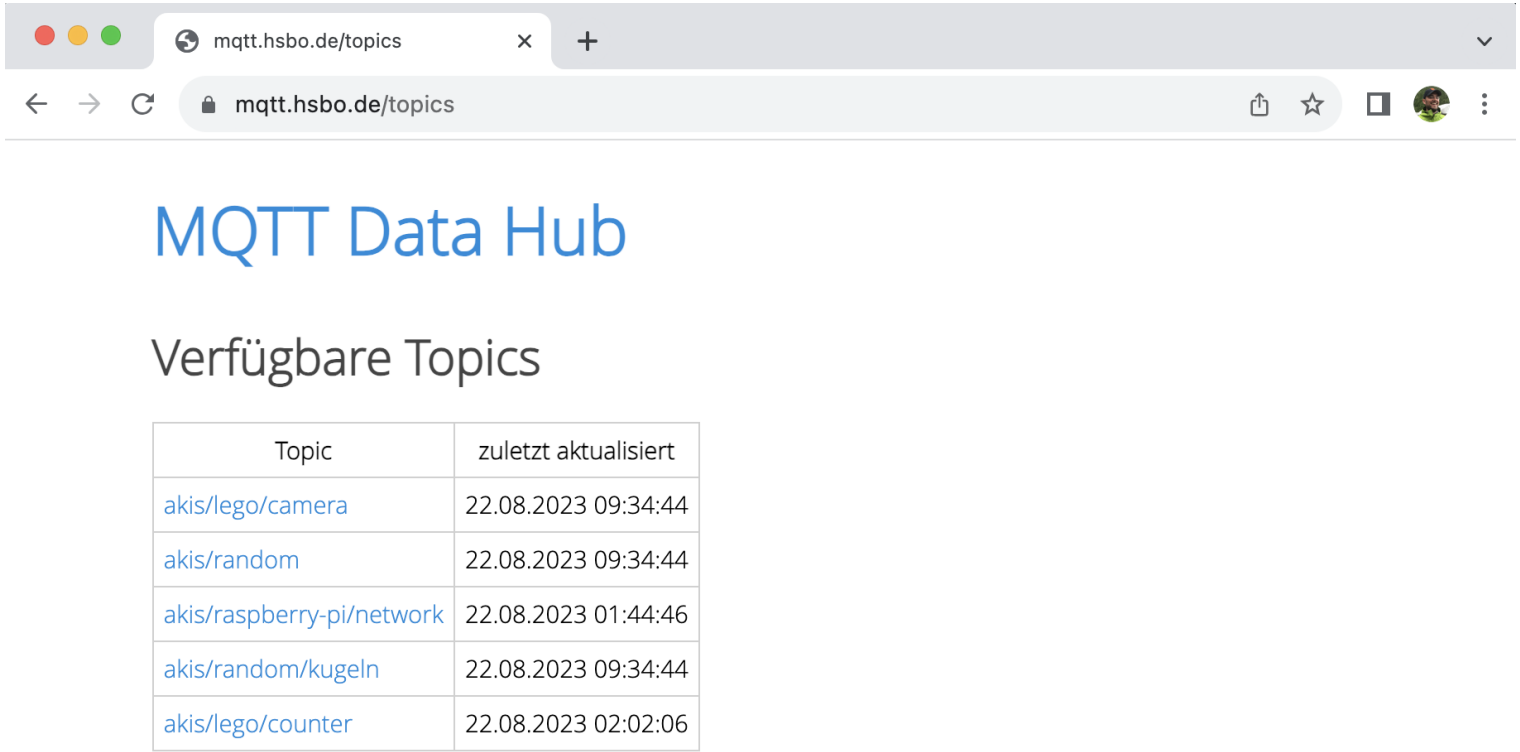
- Hoch-Skalierbarer Message Broker
- Clusterfähig (hochverfügbar)
- Entwickelt bei LinkedIn, Open Source

Anderer Broker: Apache Kafka

- Hoch-Skalierbarer Message Broker
- Clusterfähig (hochverfügbar)
- Entwickelt bei LinkedIn, Open Source

Nachrichten im Topic werden
temporär gespeichert (z.B. 1 Monat)

Broker: https://mqtt.hsbo.de



Benutzername: `akis` Kennwort: `summerschool2023`

Zeitreihen – InfluxDB

InfluxDB – eine TimeSeries Datenbank

- Optimiert auf Speichern/Verarbeiten von Zeitreihen
- OpenSource, hoch skalierbar
- Entwickelt von der Firma InfluxData
- Implementiert in Go

Warum nicht SQL?

- SQL Datenbanken auf Relationen optimiert
- Daten enthalten Verbindungen zu anderen Objekten
- Fremdschlüssel-Beziehungen müssen aufgelöst werden

- TimeSeries-Datenbanken auf Zeitreihen optimiert
- Zeitreihen enthalten i.d.R. keine komplexen Referenzen
- Referenzen werden als kompakte Meta-Daten gespeichert

Konzept von InfluxDB

- InfluxDB speichert Daten in Buckets
- Ein Bucket kann mehrere *measurements* enthalten
- Measurements sind “gleichartige” Zeitreihen
- Alle Datenpunkte in einem measurement haben die gleichen Felder

Konzept von InfluxDB

Ein Point besteht aus *tag key*, *tag values*, *field* Namen, Wert und Zeitstempel.
Eine Series ist eine Menge von Points mit gleichem Wert für *measurement*, *tag keys* und *tag values*.

_time	_measurement	city	country	_field	_value
2022-01-01T12:00:00Z	weather	London	UK	temperature	12.0
2022-02-01T12:00:00Z	weather	London	UK	temperature	12.1
2022-03-01T12:00:00Z	weather	London	UK	temperature	11.5
2022-04-01T12:00:00Z	weather	London	UK	temperature	5.9

Point

Series

_time	_measurement	city	country	_field	_value
2022-01-01T12:00:00Z	weather	Cologne	DE	temperature	13.2
2022-02-01T12:00:00Z	weather	Cologne	DE	temperature	11.5
2022-03-01T12:00:00Z	weather	Cologne	DE	temperature	10.2
2022-04-01T12:00:00Z	weather	Cologne	DE	temperature	7.9

Series

Daten ohne Anfang und Ende

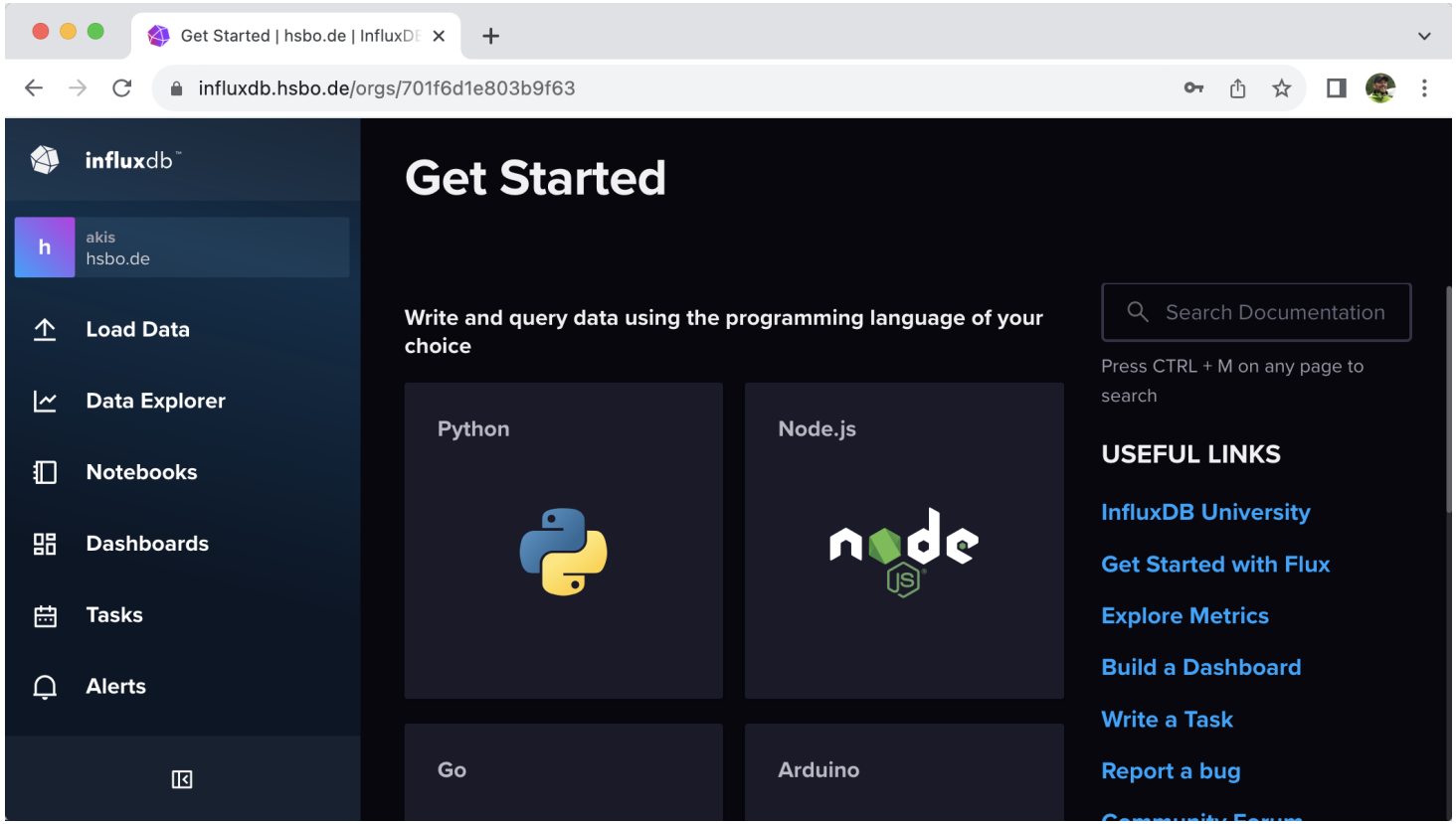
- Zeitreihendaten haben kein Anfang/Ende
- InfluxDB hat funktionale Abfrage-Sprache um gewünschte Werte aus Zeitreihen zu berechnen
- Zur Erinnerung: Bucket enthält mehrere Measurements (=Zeitreihen gleicher Art)

Influx Abfragen

- Abfragen von Daten aus Buckets
- Aneinanderreihung von Filtern (range, filter)

```
from(bucket: "akis_summerschool")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_field"] == "_anzahl" )
  |> filter(fn: (r) => r["_measurement"] == "akis/random/kugeln")
  |> group( columns: ["farbe"] )
  |> aggregateWindow(every: 15s, fn: count)
```

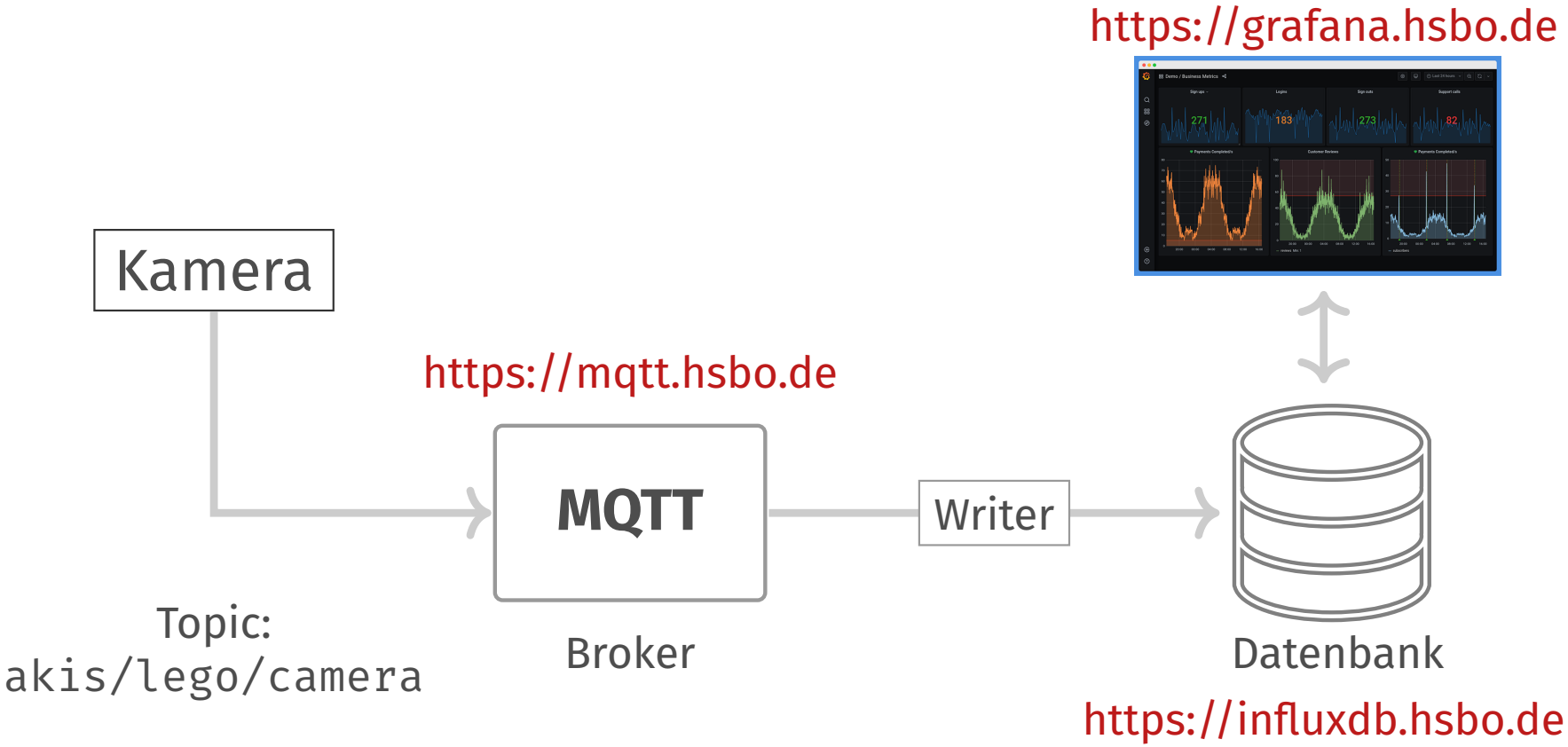
InfluxDB: https://influxdb.hsbo.de



Benutzername: `akis` Kennwort: `summerschool2023`

Beispiel: LEGO Fabrik

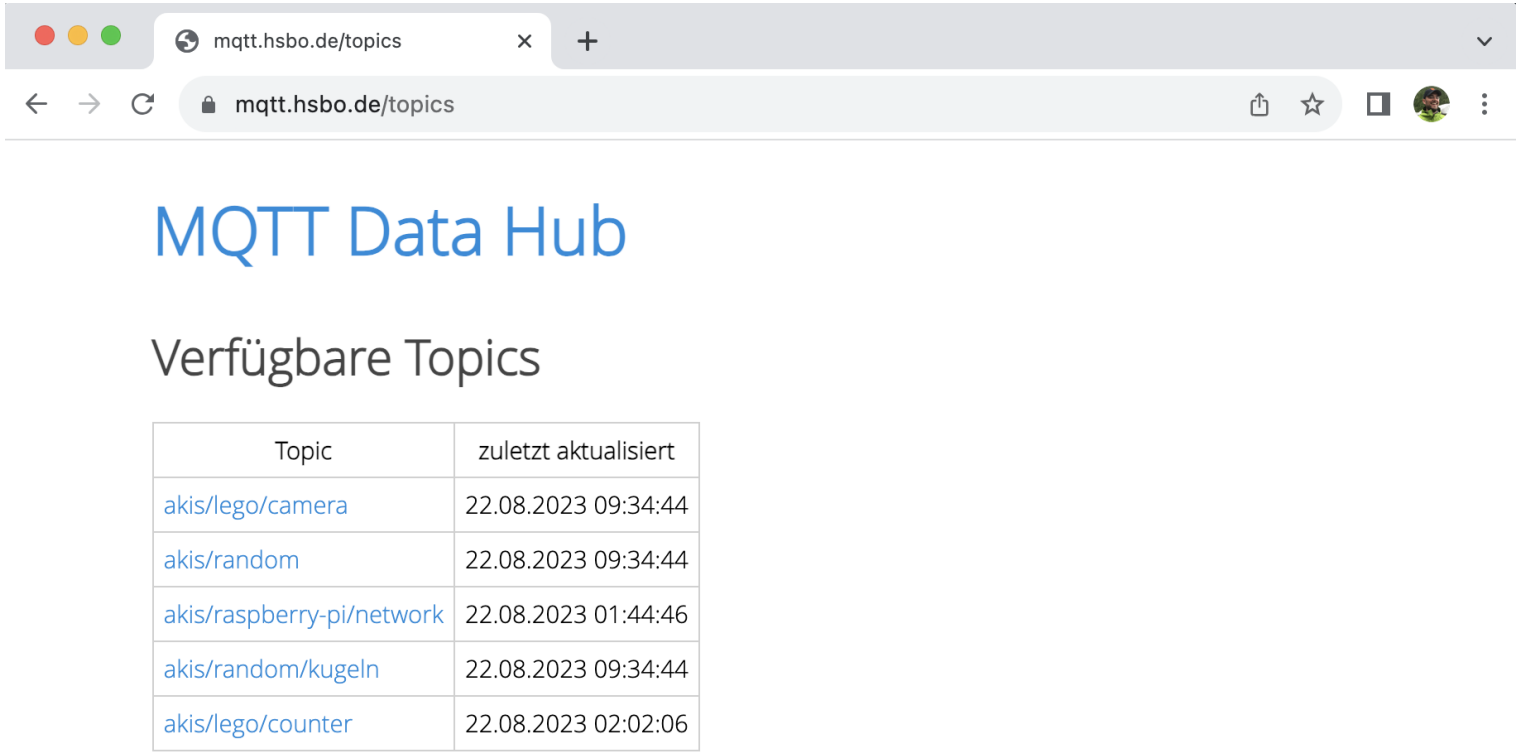
LEGO Fabrik



- Kamera liefert R/G/B Wert + Intensität (JSON-Format):

```
{ '_r':9, '_g':8, '_b':4, '_i': 24 }
```

Broker: https://mqtt.hsbo.de



Benutzername: `akis` Kennwort: `summerschool2023`

Datenstrom der LEGO Fabrik

```
#...

def on_message(client, userdata, msg):
    topic = msg.topic
    body = msg.payload.decode()

    print(f"Received '{body}'")
    print(f"  from topic {topic}")

client = connect_mqtt()
client.subscribe("akis/lego/camera")
client.on_message = on_message

client.loop_forever()
```