Dokumentation: MQTT-Anwendung mit Docker

Einführung

In diesem Projekt habe ich eine MQTT-basierte IoT-Anwendung mit Docker umgesetzt. Die Anforderung war, einen MQTT-Broker, einen Publisher und mehrere Sensoren in einer Docker-Umgebung zu verbinden und zum Laufen zu bringen. Ziel war es, Sensordaten zu erfassen, zu speichern und zu visualisieren.

Systemarchitektur

Meine Implementierung besteht aus folgenden Komponenten:

- **MQTT-Broker (Eclipse Mosquitto)**: Zentraler Nachrichtenverteiler
- **Publisher**: Sendet Testnachrichten an den Broker
- **Temperatursensoren (2)**: Simulieren Temperaturmessungen
- **Feuchtigkeitssensor**: Simuliert Feuchtigkeitsmessungen
- **InfluxDB**: Zeitreihendatenbank zur Speicherung der Sensorwerte
- **InfluxDB-Connector**: Verbindet MQTT mit InfluxDB
- **Grafana**: Visualisiert die gespeicherten Daten

Der Datenfluss ist wie folgt: Die Sensoren senden ihre Messwerte an den MQTT-Broker, der InfluxDB-Connector empfängt diese Nachrichten und schreibt sie in die InfluxDB-Datenbank. Grafana greift auf diese Datenbank zu und stellt die Daten visuell dar.

05C: Umsetzung der Aufgaben

1. Erstellung der Docker-Image-Dateien

Für jede Komponente des Systems habe ich ein eigenes Dockerfile erstellt:

- Für den Broker habe ich ein einfaches Dockerfile auf Basis des offiziellen Eclipse Mosquitto Images erstellt und eine angepasste Konfigurationsdatei hinzugefügt.
- Für die Java-basierten Komponenten (Publisher, Sensoren, InfluxDB-Connector) habe ich ein mehrstufiges Build-Verfahren verwendet:
    - In der ersten Stufe wird Maven genutzt, um die Java-Anwendung zu kompilieren
    - In der zweiten Stufe wird eine schlanke JRE-Umgebung genutzt, um die kompilierte Anwendung auszuführen
- Für Grafana und InfluxDB habe ich die offiziellen Images verwendet und entsprechend konfiguriert

2. Erstellung der Docker-Compose-Datei

In der Docker-Compose-Datei habe ich alle Komponenten definiert und ihre Abhängigkeiten, Netzwerke, Volumes und Umgebungsvariablen konfiguriert. Besonders wichtig war die

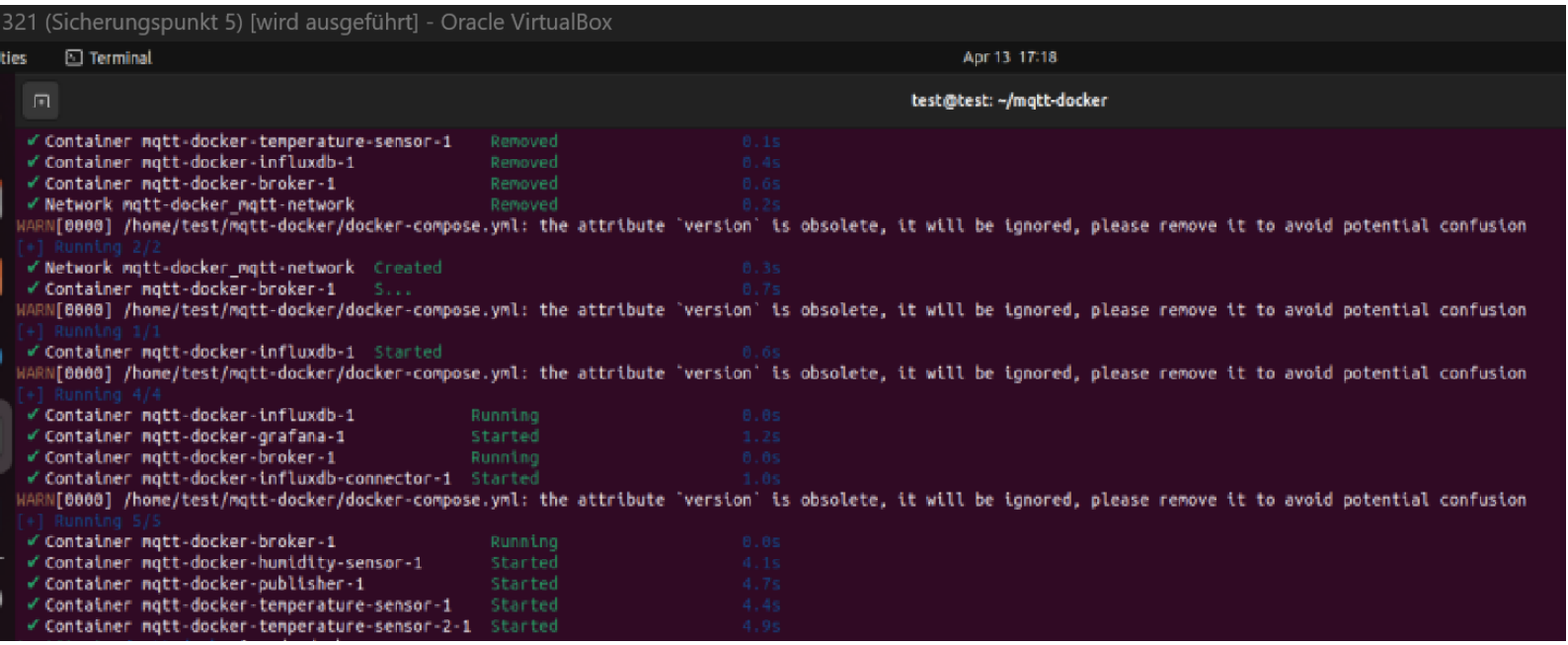
Konfiguration der Kommunikation zwischen den Containern. Zum Beispiel musste ich sicherstellen, dass die Sensoren den Broker unter dem richtigen Hostnamen erreichen können.

3. Ausführung des Docker-Compose-Befehls

Docker-Stack bauen und starten:

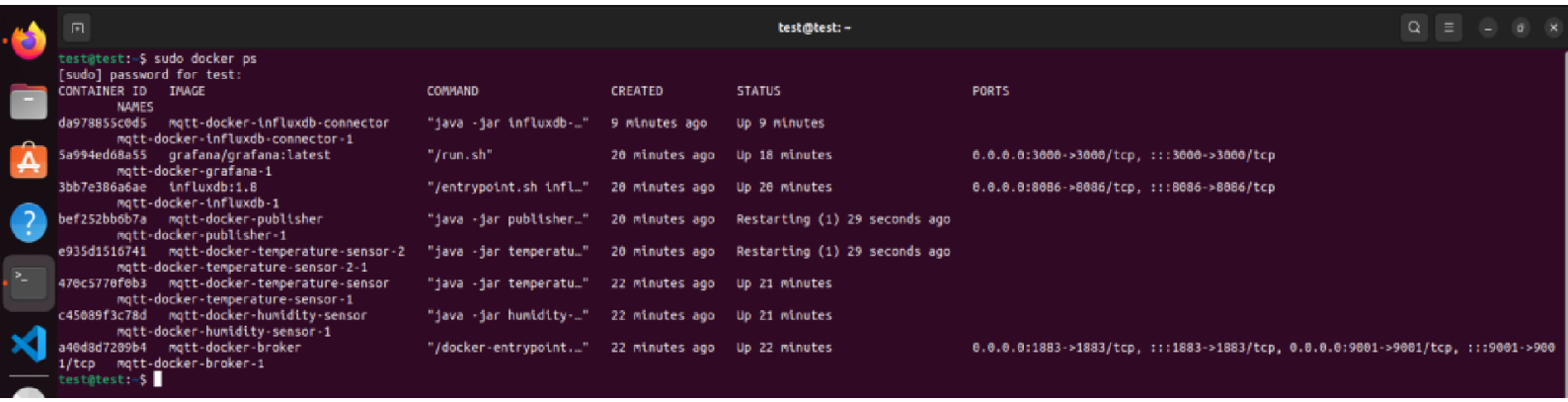
```
# Docker-Compose ausführen

docker-compose build

docker-compose up -d



# Status der Container überprüfen

docker ps
```

Bei der Ausführung von `docker-compose up -d` bin ich auf einige Probleme gestossen:



- Die Ports 1883 (MQTT), 8086 (InfluxDB) und 3000 (Grafana) waren bereits auf meinem System belegt
- Es gab Probleme mit den Java-JAR-Dateien aufgrund von Signaturproblemen

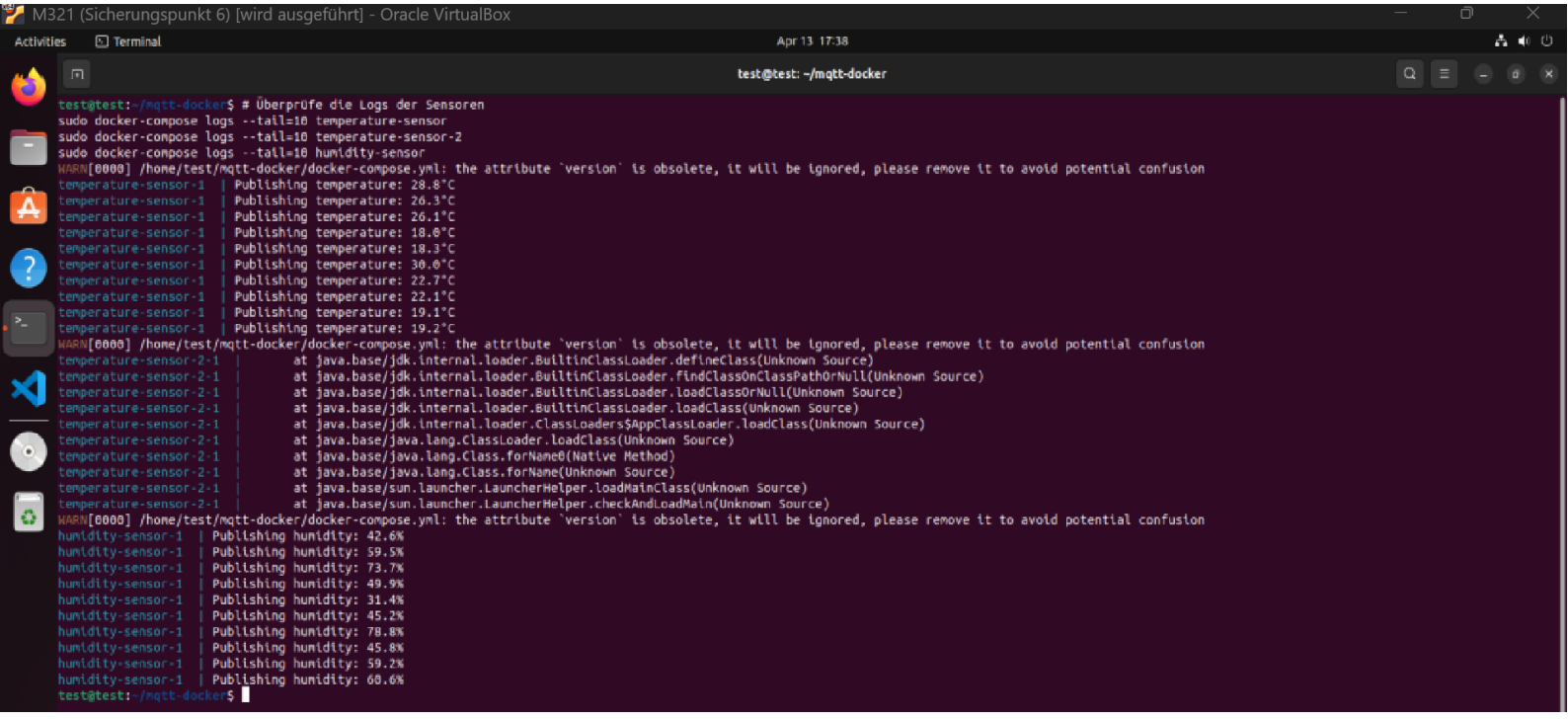
Nach der Lösung dieser Probleme konnte ich alle Container erfolgreich starten. Im Screenshot ist zu sehen, dass alle Container gestartet wurden und laufen.



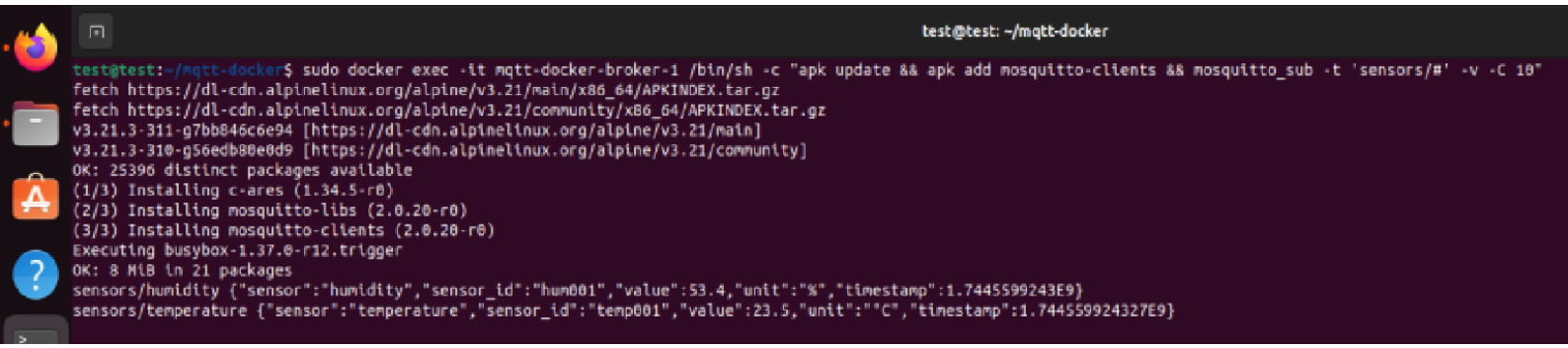
4. Funktionsüberprüfung der Komponenten

Ich habe mehrere Tests durchgeführt, um die korrekte Funktion aller Komponenten zu überprüfen:

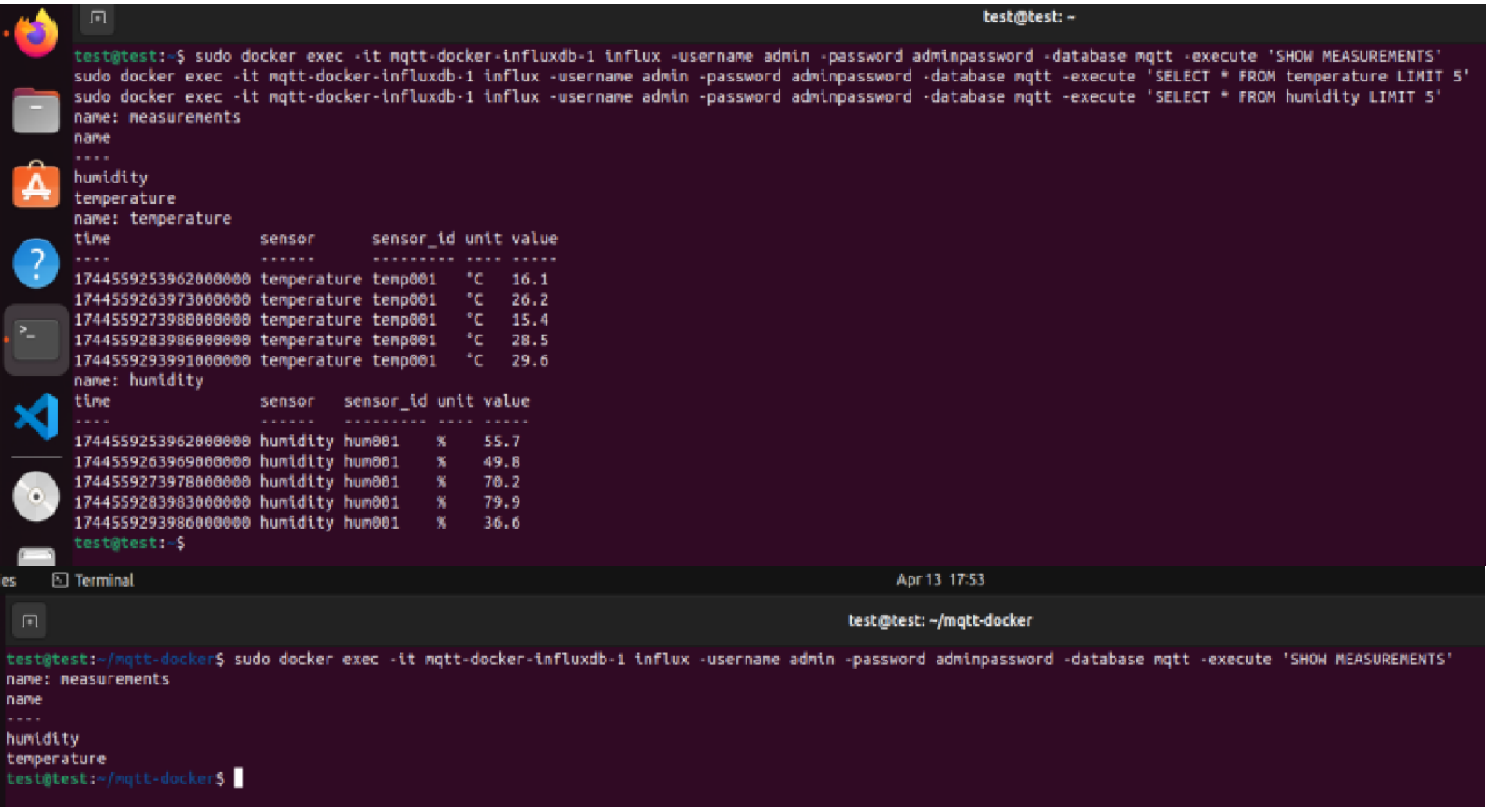
1. **Container-Status**: Mit `sudo docker ps` habe ich überprüft, dass alle Container laufen
2. **Sensordaten**: Mit `sudo docker-compose logs --tail=10 temperature-sensor` und entsprechenden Befehlen für die anderen Sensoren habe ich verifiziert, dass die Sensoren Daten generieren und senden.



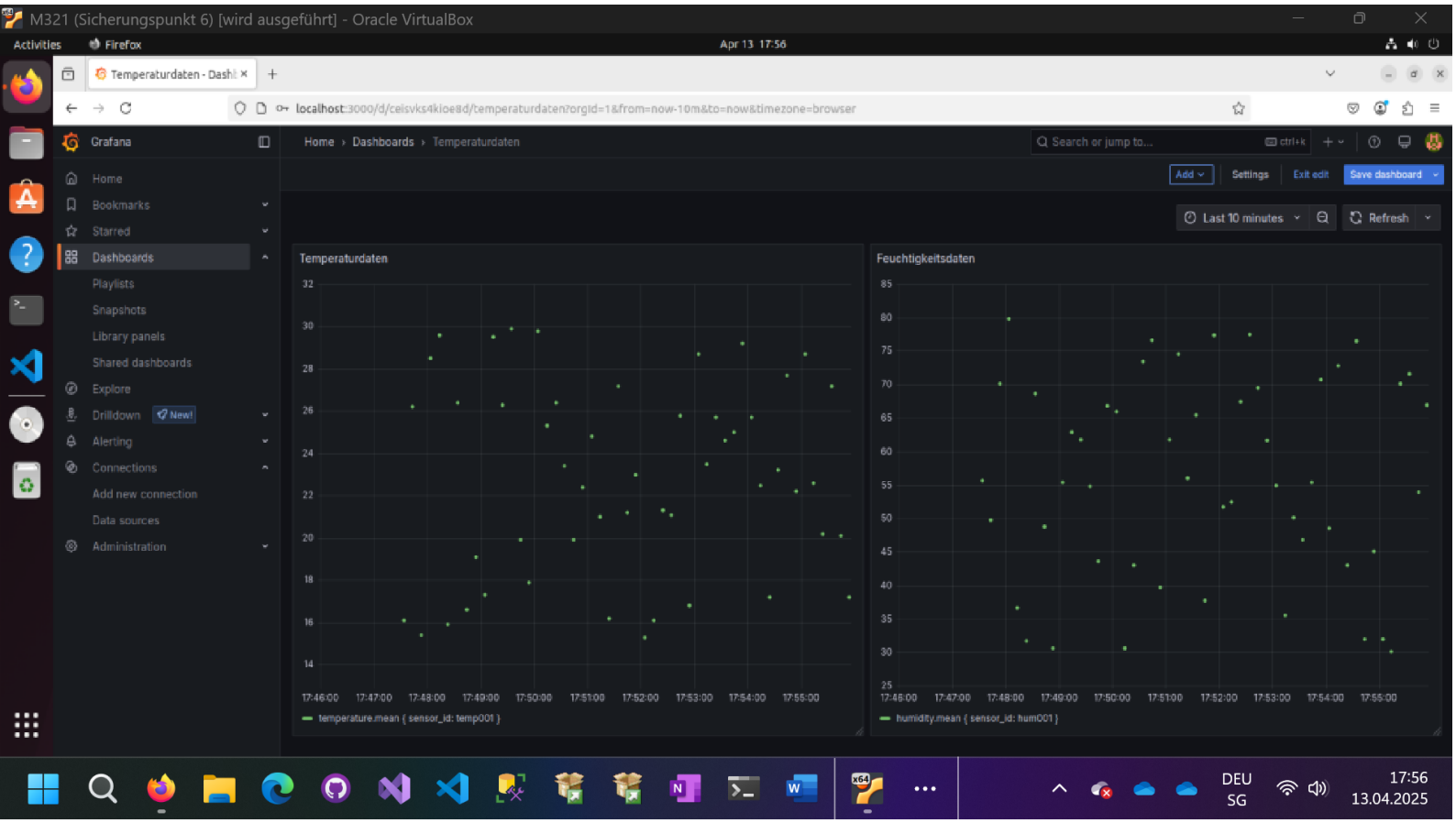
3. **MQTT-Nachrichten**: Mit dem Befehl `sudo docker exec -it mqtt-docker-broker-1 /bin/sh -c "apk update && apk add mosquitto-clients && mosquitto_sub -t 'sensors/#' -v -C 10"` habe ich überprüft, dass die MQTT-Nachrichten korrekt übertragen werden (Image 5)



4. **InfluxDB-Daten**: Mit entsprechenden Befehlen habe ich die Speicherung der Daten in InfluxDB überprüft (Image 2 und 6)



5. **Grafana-Visualisierung**: Ich habe ein Dashboard in Grafana erstellt, das die Temperatur- und Feuchtigkeitsdaten visualisiert (Image 1)



Aufgabe 6A: Erweiterte Tests und Dokumentation

1. Mindestens 2 Sensoren liefern Werte

Wie in Image 1 zu sehen ist, werden Daten von zwei Sensoren angezeigt:

- Ein Temperatursensor (`sensor_id: temp001`)
- Ein Feuchtigkeitssensor (`sensor_id: hum001`)

Die Sensordaten werden kontinuierlich aktualisiert, wie in Image 4 zu sehen ist, das die Logs der Sensoren zeigt.

2. Überwachung des Container-Status

Mit dem Befehl `sudo docker ps` habe ich den Status aller Container überwacht (Image 7). Die Ausgabe zeigt, dass alle relevanten Container laufen:

- mqtt-docker-broker-1
- mqtt-docker-temperature-sensor-1
- mqtt-docker-temperature-sensor-2-1
- mqtt-docker-humidity-sensor-1
- mqtt-docker-influxdb-1
- mqtt-docker-influxdb-connector-1
- mqtt-docker-grafana-1
- mqtt-docker-publisher-1

3. Testplan und Interaktion zwischen Komponenten

Ich habe einen umfassenden Testplan durchgeführt:

a) **Container-Status-Test**: Alle Container laufen (Image 7)

b) **MQTT-Broker-Test**: Der Broker empfängt Nachrichten von den Sensoren (Image 5)

```
sensors/humidity
{"sensor":"humidity","sensor_id":"hum001","value":53.4,"unit":"%","timestam
p":1.74455992439E9}
sensors/temperature
{"sensor":"temperature","sensor_id":"temp001","value":23.5,"unit":"°C","tim
estamp":1.74455992427E9}
```

c) **Sensordaten-Test**: Die Sensoren veröffentlichen kontinuierlich Daten (Image 4)

```
temperature-sensor-1 | Publishing temperature: 28.8°C
humidity-sensor-1 | Publishing humidity: 60.6%
```

d) **InfluxDB-Datenspeicherung-Test**: Die Daten werden korrekt in InfluxDB gespeichert (Image 6)

```
name: temperature
time                sensor       sensor_id unit value
----                ------       --------- ---- -----
```

```
1744559253962000000  temperature  temp001   °C   16.1
```

e) **Grafana-Visualisierung-Test**: Die Daten werden korrekt in Grafana visualisiert (Image 1)

Die Kommunikation zwischen den Komponenten funktioniert wie erwartet. Die Sensoren senden Daten an den MQTT-Broker, der InfluxDB-Connector empfängt diese Nachrichten und speichert sie in InfluxDB, und Grafana visualisiert die Daten.

4. Probleme und Lösungen

Während der Implementierung bin ich auf folgende Probleme gestoen:

1. **Port-Konflikte**: Die Ports 1883, 8086 und 3000 waren bereits auf meinem System belegt. Ich habe die lokalen Dienste (Mosquitto, InfluxDB, Grafana) gestoppt, um die Ports freizugeben.
2. **JAR-Signierungsprobleme**: Bei den Java-Komponenten gab es Probleme mit den Signaturen in den JAR-Dateien:

```
Exception in thread "main" java.lang.SecurityException: Invalid signature
file digest for Manifest main attributes
```

   Dies trat bei den Sensoren und dem InfluxDB-Connector auf. Ich musste die pom.xml-Dateien anpassen, um die Signaturprüfung zu deaktivieren.

3. **Container-Abhängigkeiten**: Die Reihenfolge des Starts der Container war wichtig. Ich musste sicherstellen, dass der Broker und InfluxDB vollständig gestartet sind, bevor die anderen Container gestartet werden.

5. Zusammenfassung und Ausblick

Das implementierte System erfüllt alle Anforderungen der Aufgabenstellung. Die Docker-Containerisierung ermöglicht eine einfache Bereitstellung und Skalierung des Systems. Das System kann wie folgt erweitert werden:

1. **Weitere Sensoren**: Hinzufügen weiterer Sensortypen wie Luftdruck oder Lichtsensoren
2. **Authentifizierung**: Implementierung von Sicherheitsmechanismen für den MQTT-Broker
3. **Alarme**: Konfiguration von Alarmregeln in Grafana für kritische Sensorwerte
4. **Persistente Speicherung**: Konfiguration von Retention-Policies für die langfristige Datenspeicherung

Die Nutzung von Docker hat sich als effektiv erwiesen, um ein verteiltes System mit mehreren Komponenten zu implementieren und zu verwalten. Die Container-Technologie ermöglicht eine klare Trennung der Komponenten und vereinfacht die Bereitstellung.