

# MQTT Smart Meter Integration

## 1. Ziel & Kontext - Warum wird das Feature benötigt?

- **Was soll erreicht werden:** Smart Meter (Stromzähler) senden Messwerte über MQTT an das ZEV-Backend, wo sie automatisch persistiert und verarbeitet werden.
- **Warum machen wir das:**
  - Automatisierte Datenerfassung statt manueller CSV-Uploads
  - Echtzeit-Überwachung von Verbrauch und Einspeisung
  - Grundlage für Live-Dashboards und automatische Abrechnungen
- **Aktueller Stand:** Messwerte werden manuell per CSV-Upload erfasst ( `MesswerteUploadComponent` ). Dies ist fehleranfällig und zeitverzögert.

## 2. Funktionale Anforderungen (FR)

### FR-1: MQTT Broker (Mosquitto)

1. Mosquitto wird als Docker-Service im Stack bereitgestellt
2. Broker läuft auf Port 1883 (MQTT) und 9001 (WebSocket)
3. Authentifizierung über Username/Password
4. ACLs beschränken Zugriff pro Mandant (Organization)

### FR-2: Topic-Struktur

```
zev/{organizationId}/{einheitId}/messwert
```

Segment	Beschreibung	Beispiel
zev	Root-Namespace	zev
{organizationId}	Keycloak Organization UUID	550e8400-e29b-41d4-a716-446655440000
{einheitId}	Einheit-ID aus der Datenbank	123
messwert	Message-Typ	messwert

Beispiel-Topic:

```
zev/550e8400-e29b-41d4-a716-446655440000/123/messwert
```

### FR-3: Payload-Format (JSON)

```
{
  "timestamp": "2025-12-24T14:30:00Z",
  "verbrauch": 1.25,
  "einspeisung": 0.0,
  "zaehlerstandVerbrauch": 12345.67,
  "zaehlerstandEinspeisung": 5678.90
}
```

Feld	Typ	Pflicht	Beschreibung
timestamp	ISO 8601 DateTime	Ja	Zeitpunkt der Messung
verbrauch	Decimal (kWh)	Ja	Verbrauch seit letzter Messung
einspeisung	Decimal (kWh)	Ja	Einspeisung seit letzter Messung
zaehlerstandVerbrauch	Decimal (kWh)	Nein	Absoluter Zählerstand Verbrauch
zaehlerstandEinspeisung	Decimal (kWh)	Nein	Absoluter Zählerstand Einspeisung

#### FR-4: Backend MQTT Subscriber

1. Spring Boot Service subscribed auf `zev/+/+/messwert` (Wildcard)
2. Bei eingehender Nachricht:
  - Parse Topic: Extrahiere `organizationId` und `einheitId`
  - Validiere JSON-Payload
  - Prüfe ob Einheit existiert und zur Organization gehört
  - Speichere Rohdaten in Tabelle `zaehler_rohdaten`
3. Bei Validierungsfehlern: Logge Warnung, verwirfe Nachricht
4. Duplikat-Erkennung: Gleicher Timestamp + Einheit = Update statt Insert

#### FR-5: Rohdaten-Persistierung

MQTT-Nachrichten werden zunächst in einer separaten Rohdaten-Tabelle gespeichert:

```
CREATE TABLE zaehler_rohdaten (
  id BIGSERIAL PRIMARY KEY,
  einheit_id BIGINT NOT NULL REFERENCES einheit(id),
  organization_id UUID NOT NULL,
  timestamp TIMESTAMP WITH TIME ZONE NOT NULL,
  verbrauch DECIMAL(12,4) NOT NULL,
  einspeisung DECIMAL(12,4) NOT NULL,
  zaehlerstand_verbrauch DECIMAL(12,4),
  zaehlerstand_einspeisung DECIMAL(12,4),
  empfangen_am TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  verarbeitet BOOLEAN DEFAULT FALSE,
  verarbeitet_am TIMESTAMP WITH TIME ZONE,

  CONSTRAINT uk_zaehler_rohdaten UNIQUE (einheit_id, timestamp)
);

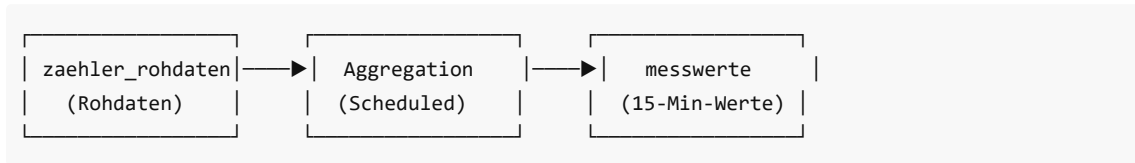
CREATE INDEX idx_zaehler_rohdaten_unverarbeitet
ON zaehler_rohdaten(verarbeitet, timestamp)
WHERE verarbeitet = FALSE;
```

Spalte	Beschreibung
empfangen_am	Zeitpunkt des MQTT-Empfangs
verarbeitet	Flag ob bereits in messwerte übernommen

verarbeitet_am	Zeitpunkt der Übernahme
----------------	-------------------------

## FR-6: Scheduled Aggregation (15-Minuten-Job)

Ein Scheduled Job läuft alle 15 Minuten (:00, :15, :30, :45) und überträgt Rohdaten in die `messwerte`-Tabelle:



### Ablauf:

- Job startet um :00, :15, :30, :45 (Cron: `0 0,15,30,45 * * * *`)
- Ermittle das abgeschlossene 15-Minuten-Intervall (z.B. bei :15 → Intervall 00:00-00:15)
- Für jede Einheit mit unverarbeiteten Rohdaten im Intervall:
  - Aggregiere** alle Rohdaten des Intervalls:
    - `verbrauch` = Summe aller Verbrauchswerte
    - `einspeisung` = Summe aller Einspeisewerte
  - Bestimme Zeitstempel:** Ende des Intervalls (z.B. 00:15:00)
  - Speichere** in `messwerte`-Tabelle mit `quelle = 'MQTT'`
- Markiere verarbeitete Rohdaten: `verarbeitet = TRUE`, `verarbeitet_am = NOW()`

### Beispiel:

Rohdaten (zaehler_rohdaten)		
14:01:00	verbrauch: 0.3	einspeisung: 0.0
14:05:00	verbrauch: 0.4	einspeisung: 0.0
14:10:00	verbrauch: 0.5	einspeisung: 0.1

↓ Aggregation um 14:15 ↓

messwerte		
14:15:00	verbrauch: 1.2	einspeisung: 0.1

### Erweiterung messwerte-Tabelle:

```

ALTER TABLE messwerte ADD COLUMN IF NOT EXISTS quelle VARCHAR(20) DEFAULT 'CSV';
-- Mögliche Werte: 'CSV', 'MQTT', 'API'
  
```

## FR-7: Monitoring & Status

- Prometheus-Metriken für MQTT-Empfang:

- `zev_mqtt_messages_received_total` (Counter)
- `zev_mqtt_messages_processed_total` (Counter)
- `zev_mqtt_messages_failed_total` (Counter)
- `zev_mqtt_last_message_timestamp` (Gauge)

## 2. Prometheus-Metriken für Aggregation:

- `zev_aggregation_runs_total` (Counter)
- `zev_aggregation_records_processed_total` (Counter)
- `zev_aggregation_duration_seconds` (Histogram)
- `zev_aggregation_last_run_timestamp` (Gauge)

## 3. Health-Check Endpoints:

- `/actuator/health/mqtt` zeigt Verbindungsstatus zum Broker
- `/actuator/health/aggregation` zeigt Status des letzten Aggregation-Laufs

# 3. Akzeptanzkriterien

## MQTT-Empfang:

- ☐ Mosquitto-Container startet mit `docker-compose up`
- ☐ Backend verbindet sich automatisch zum MQTT-Broker
- ☐ Gültige MQTT-Nachrichten werden in `zaehler_rohdaten` gespeichert
- ☐ Ungültige Nachrichten werden geloggt aber nicht gespeichert
- ☐ Nachrichten für nicht-existierende Einheiten werden abgelehnt
- ☐ Organization-ID im Topic muss zur Einheit passen (Multi-Tenancy)

## 15-Minuten-Aggregation:

- ☐ Scheduled Job läuft um :00, :15, :30, :45
- ☐ Rohdaten werden korrekt zu 15-Minuten-Werten aggregiert
- ☐ Aggregierte Werte erscheinen in `messwerte` mit `quelle = 'MQTT'`
- ☐ Rohdaten werden nach Verarbeitung als `verarbeitet = TRUE` markiert
- ☐ Bereits verarbeitete Rohdaten werden nicht erneut aggregiert

## Monitoring:

- ☐ Prometheus-Metriken sind unter `/actuator/prometheus` verfügbar
- ☐ Health-Check zeigt MQTT-Verbindungsstatus

# 4. Nicht-funktionale Anforderungen (NFR)

## NFR-1: Performance

- Verarbeitung einer Nachricht < 100ms
- System muss mindestens 100 Nachrichten/Sekunde verarbeiten können
- Nachrichten werden asynchron verarbeitet (non-blocking)

## NFR-2: Sicherheit

- MQTT-Broker erfordert Authentifizierung (Username/Password)
- ACLs beschränken Topics pro Mandant
- Keine Passwörter im Code - Environment-Variablen verwenden
- TLS-Verschlüsselung für Produktionsumgebung (Port 8883)

## NFR-3: Zuverlässigkeit

- QoS Level 1 (At least once delivery)

- Persistent Sessions für Reconnect nach Verbindungsabbruch
- Retry-Logik bei Datenbankfehlern

#### NFR-4: Kompatibilität

- Bestehende CSV-Upload-Funktionalität bleibt erhalten
- Messwerte aus beiden Quellen werden gleich behandelt
- Keine Breaking Changes an der `messwerte` -Tabelle

## 5. Edge Cases & Fehlerbehandlung

#### MQTT-Empfang:

Szenario	Verhalten
Ungültiges JSON	Warnung loggen, Nachricht verwerfen
Unbekannte Einheit-ID	Warnung loggen, Nachricht verwerfen
Organization-ID stimmt nicht	Fehler loggen (Security-Concern), Nachricht verwerfen
Timestamp in der Zukunft	Warnung loggen, Nachricht trotzdem speichern
Duplikat (gleicher Timestamp)	Bestehenden Eintrag aktualisieren
Broker nicht erreichbar	Reconnect mit Exponential Backoff
Datenbank nicht erreichbar	Nachricht in Memory-Queue, Retry
Negative Werte	Warnung loggen, Nachricht verwerfen

#### 15-Minuten-Aggregation:

Szenario	Verhalten
Keine Rohdaten im Intervall	Kein Eintrag in messwerte (kein Nullwert)
Nur eine Rohdaten-Zeile	Wert 1:1 übernehmen
Rohdaten über Intervallgrenze	Jedes Intervall separat aggregieren
Job-Ausfall (z.B. Neustart)	Nächster Job verarbeitet alle unverarbeiteten Daten
Bereits existierender messwerte-Eintrag	Update statt Insert (Upsert)
Rohdaten mit <code>verarbeitet=TRUE</code>	Werden ignoriert

## 6. Abgrenzung / Out of Scope

- **Nicht** umgesetzt in dieser Phase:
  - Frontend-Anzeige von Live-Daten (WebSocket-Bridge)
  - Automatische Geräte-Registrierung (Einheiten müssen existieren)
  - Bidirektionale Kommunikation (Befehle an Zähler senden)
  - Komplexe ACL-Verwaltung über UI
  - TLS-Zertifikatsverwaltung (manuell konfigurieren)

## 7. Offene Fragen

- ☐ Welche Smart Meter Hardware soll unterstützt werden? (Payload-Format ggf. anpassen)
- ☐ Soll es ein Admin-UI für MQTT-Konfiguration geben?
- ☐ Benötigen wir eine Dead Letter Queue für fehlgeschlagene Nachrichten?
- ☐ Soll der Broker extern erreichbar sein oder nur im Docker-Netzwerk?
- ☐ Wie lange sollen Rohdaten aufbewahrt werden? (Cleanup-Job?)

## 8. Technische Details

### Docker-Compose Erweiterung

```
mosquitto:
  image: eclipse-mosquitto:2
  container_name: mosquitto
  ports:
    - "1883:1883"
    - "9001:9001"
  volumes:
    - ./mosquitto/config:/mosquitto/config
    - ./mosquitto/data:/mosquitto/data
    - ./mosquitto/log:/mosquitto/log
  networks:
    - zev-network
  restart: unless-stopped
```

### Mosquitto Konfiguration

```
# mosquitto/config/mosquitto.conf
listener 1883
listener 9001
protocol websockets

allow_anonymous false
password_file /mosquitto/config/passwords.txt

acl_file /mosquitto/config/acl.conf
```

### Spring Boot Dependencies

```
<dependency>
  <groupId>org.springframework.integration</groupId>
  <artifactId>spring-integration-mqtt</artifactId>
</dependency>
<dependency>
  <groupId>org.eclipse.paho</groupId>
  <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
```

```
<version>1.2.5</version>
</dependency>
```

## Backend Konfiguration (application.yml)

```
mqtt:
  broker:
    url: tcp://mosquitto:1883
    username: ${MQTT_USERNAME:zev-backend}
    password: ${MQTT_PASSWORD:secret}
  client:
    id: zev-backend-${random.uuid}
  topics:
    messwerte: zev/+/+/messwert
  qos: 1
```

## 9. Testplan

### Unit Tests

- `MqttMessageParserTest` - Payload-Parsing und Validierung
- `MqttTopicParserTest` - Topic-Extraktion (orgId, einheitId)
- `MesswertAggregatorTest` - 15-Minuten-Aggregationslogik
- `IntervalCalculatorTest` - Berechnung der Intervallgrenzen

### Integration Tests

- `MqttSubscriberIT` - MQTT-Empfang mit Testcontainers (Mosquitto + PostgreSQL)
  - Publish Test-Nachricht → Prüfe Eintrag in `zaehler_rohdaten`
- `MesswertAggregationIT` - Scheduled Job Verarbeitung
  - Rohdaten einfügen → Job triggern → Prüfe `messwerte` -Eintrag
  - Prüfe `verarbeitet` -Flag nach Job-Durchlauf
- `MultipleMessagesIT` - Mehrere Nachrichten im gleichen Intervall
  - 3 Rohdaten einfügen → Job triggern → Prüfe Summe in `messwerte`

### Manueller Test

- Test-Publisher Script für Entwicklung:

```
# Einzelne Nachricht
mosquitto_pub -h localhost -p 1883 -u testuser -P testpass \
  -t "zev/org-123/einheit-1/messwert" \
  -m '{"timestamp":"2025-12-24T14:30:00Z","verbrauch":1.5,"einspeisung":0.0}'

# Mehrere Nachrichten simulieren (für Aggregations-Test)
for i in 1 2 3; do
  mosquitto_pub -h localhost -p 1883 -u testuser -P testpass \
    -t "zev/org-123/einheit-1/messwert" \
```

```
-m "{\"timestamp\":\"2025-12-24T14:00${i}:00Z\",\"verbrauch\":0.5,\"einspeisung\":0.0}"
```

done