Hochschule Karlsruhe
University of
Applied Sciences

Hochschule Karlsruhe – Data Engineering
WS 2021/2022
DSCB330 – Vorlesung 3 – Pub/Sub

Foto: Quirin Leppert

Themenübersicht

Data Integration

- Data Formats (csv, XML, json)
- Extract, Transform, Load
- Object Relation Mapper (ORM)
- Staging

Data Processing

- Relationale Datenbanken
- nicht-relationale Datenbanken
- Resource Description Framework (RDF)
- Ontologien
- Data Warehouse

Data Modelling

- **Serialisierung**
- **OPC UA**
- **MQTT**
- **Pub/Sub**
- Data pipelines
  - Apache Airflow
  - gRPC

Web-Service Architektur

- Front-End
- Backend for Frontend (BFF)
- Micro Services
- Docker Container

Security

- Security ist wichtig in allen Phasen der Softwareentwicklung und Datenbereitstellung.

Übung

- Erstellung eines Daten-Modells einer prozesstechnischen Anlage
- Statische Daten
- Dynamische Daten
- Auswertung der Daten

Standards für den Datenaustausch Machine-to-Machine (M2M)

MQTT

- Message Queuing Telemetry Transport
- Publish/Subscribe-Architektur
- Übertragung von Telemetriedaten in Form von Nachrichten
- Kommunikation auch bei schlechter Netzwerkverbindung möglich
- Verschlüsselung über TLS möglich
- Standardisierung durch OASIS

OPC UA

- Open Platform Communications Unified Architecture
- plattformunabhängige, service-orientierte Architektur
- Transport von Maschinendaten (Regelgrößen, Messwerten, Parametern, HMI-Beschreibung, …)
- inklusive semantischer Beschreibung
- Integriertes Security-Konzept (UA Security)
- Standardisierung IEC 62541

Quelle: https://de.wikipedia.org/wiki/MQTT; https://mqtt.org/; https://de.wikipedia.org/wiki/OPC_Unified_Architecture; https://opcfoundation.org/

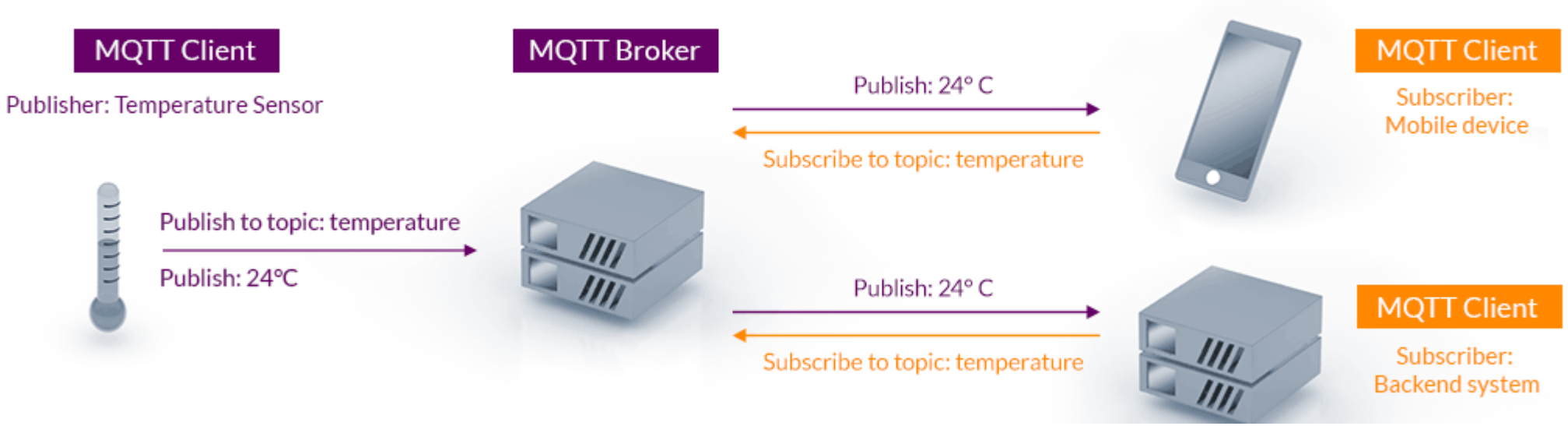
Message Queuing Telemetry Transport (MQTT)

MQTT Architektur

- Server (Broker) dient als Drehscheibe für die gesamte Kommunikation
- Clients melden sich beim Server an
- Für die Datenübertragung veröffentlicht (publish) der Client eine Nachricht mit einem bestimmten Topic.
- Für den Datenempfang meldet (subscribe) sich der Client für ein oder mehrere Topics an.

Security (optional)

- Unverschlüsselte Kommunikation über Port 1883
- Verschlüsselte Kommunikation (Transport Layer Security (TLS)) über Port 8883
- im Prinzip beliebiger, freier Port nutzbar
- Verifizierung der Echtheit des Servers über ein Zertifikat
- Client-Login über User/Passwort



Quelle: https://mqtt.org/

MQTT – Standards

MQTT 5.0

+ Kommunikation über TCP/IP
+ aktueller Standard Version 5
+ https://mqtt.org/mqtt-specification/

MQTT-SN

+ Nutzung anderer Netzwerkprotokolle als TCP/IP
- ZigBee, LoRaWAN, Narrowband IoT (NB-IoT)
+ SN steht für Sensor Networks
+ Geeignet für low power, embedded devices
+ aktueller Standard Version 1.2
+ Geringerer Overhead bei der Übertragung als bei MQTT

Quelle: https://mqtt.org/

MQTT – CONNECT / DISCONNECT

Verbindung mit dem Server (CONNECT)

+ Client Identifier (notwendig)
  - eindeutig bezüglich des Servers
+ Will Message (optional)
  - Nachricht, die der Server nach einem Verbindungsabbruch des Clients verschickt
+ User Name (optional)
+ Password (optional)
+ Nach erfolgreicher Verbindung werden zwischen Server und Client regelmäßig Pakete zur Überprüfung einer vorhandenen Verbindung ausgetauscht (Keep Alive).
  - **MQTT-SN** also features a keep-alive procedure that allows devices to go to sleep when they are not needed and receive any information waiting for them when they wake up.
+ Verbindungsabbruch bei Überschreitung des Keep Alive-Intervalls
+ Aktiv durch den Server (z.B. bei Übermittlung eines ungültigen Passworts)

Trennung vom Server

+ Abmeldung durch den Client (DISCONNECT)

Quelle: https://mqtt.org/; https://www.u-blox.com/en/blogs/insights/mqtt-sn

MQTT – Übertragung einer Nachricht

Übertragung einer Nachricht (Application message)

+ Der Vorgang des Sendens wird publishing genannt.
+ Verschiedene Clients können zum gleichen Topic Inhalte senden.
+ Alle Clients, die ein Topic subscribed haben, empfangen die Nachricht.
+ Eine Nachricht besteht aus
  - Payload data,
  - Quality of Service (QoS),
  - Collection of Properties,
  - Topic Name
+ Frei definierbare Struktur der Topics. Als Trennzeichen der Struktur dient ein Schrägstrich „/“.
  - z.B. Deutschland/Karlsruhe/HKA/E003/Temperatur
+ Der Inhalt einer Nachricht wird „payload“ genannt.

+ Für die Nachrichtenübertragung können drei Qualitäts-Level festgelegt werden:
  - „At most once“: Die Nachricht wird höchstens einmal übertragen, d.h. evtl. wird die Nachricht nicht verschickt.
  - „At least once“: Die Nachricht wird mindestens einmal, evtl. auch mehrmals verschickt.
  - „Exactly once“: Nachrichten werden genau einmal übertragen.
+ Maximale Länge einer Nachricht: ca. 260 MB
+ Maximale Topic-Länge: 65536 byte
+ Übertragung von Strings (UTF-8) oder beliebiger binärer Daten möglich
+ Retain Flag: Der Server speichert die letzte Nachricht zu einem Topic. Diese Nachricht wird auch an Clients übertragen, die das Topic subscriben, nachdem die Nachricht verschickt worden ist.

Quelle: https://mqtt.org/

MQTT – Implementierungen

Mosquitto

- \+ Client und Server
- \- https://mosquitto.org/download/
- \- Dokumentation https://mosquitto.org/man/mosquitto-8.html

HiveMQ MQTT

- \+ Client und Server
- \- https://www.hivemq.com/
- \+ Public Broker
- \- broker.hivemq.com

Pahoo

- \+ Client für python (API)
- \- https://www.eclipse.org/paho/index.php?page=clients/python/index.php
- \- Paket paho-mqtt
- \+ Public Broker
- \- mqtt.eclipse.org

MQTTnet

- \+ Client und Server
- \+ API für .NET
- \+ https://github.com/chkr1011/MQTTnet/wiki
- \+ https://github.com/chkr1011/MQTTnet

Quelle: https://mqtt.org/; https://www.hivemq.com/blog/how-to-get-started-with-mqtt/

MQTT – Publish / Subscribe

Publish a message

```
import paho.mqtt.client as mqtt
client = mqtt.Client()

client.connect("localhost",1883,60)

client.publish("topic/test", "Hello world!")

client.disconnect()
```

+ Übung 09-MQTT

Subscribe to a topic

```
def on_connect(client, userdata, flags, rc):
  print("Connected with result code "+str(rc))

  client.subscribe("topic/test")
def on_message(client, userdata, msg):
  if msg.payload.decode() == "Hello world!":
    print("Yes!")

    client.disconnect()

client = mqtt.Client()

client.connect("localhost",1883,60)
client.on_connect=on_connect

client.on_message=on_message
client.loop_forever()
```

Quelle: https://mqtt.org/; https://www.hivemq.com/blog/how-to-get-started-with-mqtt/

Übungsaufgabe 5

Publish/Subscribe binary data mit MQTT

- Ein Sensor misst jede Sekunde 1024 Beschleunigungswerte. Die möglichen Messwerte reichen von -32768 bis 32767 (int16). Die Samplerate beträgt 10000 Hz. Der Wert 32767 entspricht einer Beschleunigung von 16 g (Ortsfaktor), der Wert 0 entspricht einer Beschleunigung von 0 g.
- Der Sensor misst ein sinusförmiges Signal mit einer Amplitude von 10 g und einer Frequenz von 775 Hz. Das sinusförmige Signal ist leicht verrauscht (Addition einer Pseudo-Zufallszahl mit maximalen Werten von ±0,01 g).
- Implementieren Sie einen MQTT-Client, der den Zeitstempel der Messung und die Messwerte über MQTT jede Sekunde an das Topic sens1/binary als binäres Format veröffentlicht.
- Implementieren Sie einen MQTT-Client, der das Topic sens1/binary empfängt.
- Speichern Sie in einer SQL-Tabelle (1) den Zeitstempel der Messung, den Zeitstempel des Empfangs der Nachricht im datetime2-Format und die Messwerte im Binärformat.
- Speichern Sie in einer weiteren SQL-Tabelle (2) jeden einzelnen Messwert mit den beiden Zeitstempeln und einem Identifier für das Datenpaket.
- Führen Sie den Vorgang für ca. 10 min aus.
- (1) Lesen Sie Datenpakete im Binärformat aus der Tabelle sortiert nach der Zeit der Erzeugung aus und decodieren Sie die Daten. Beschränken Sie den Zeitraum der auszulesenden Messwerte auf die dritte Minute.
- (2) Lesen Sie die Daten aus der Tabelle sortiert nach der Zeit der Erzeugung gruppiert nach dem Identifier für das Datenpaket aus. Beschränken Sie den Zeitraum der auszulesenden Messwerte auf die dritte Minute.
- Vergleichen Sie die Ausführungszeit beider Methoden.

Advanced Message Queuing Protocol (AMQP)

Standardisierung

- Standardisierung durch OASIS
- ISO/IEC 19464
- Vision: To become the standard protocol for interoperability between all messaging middleware

Implementierungen

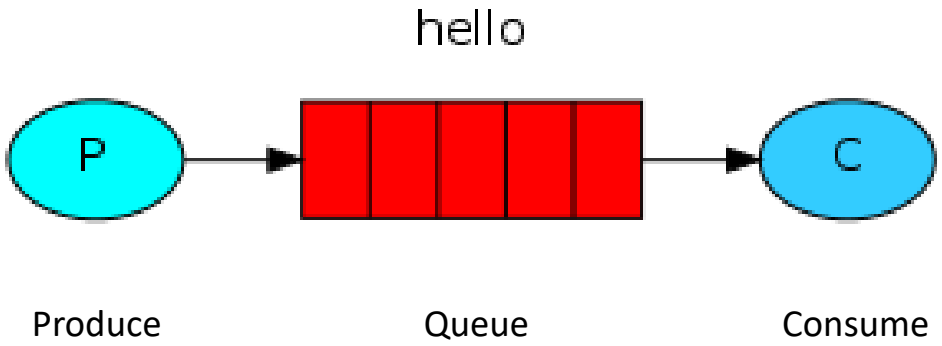
- SwiftMQ
- Microsoft Windows Azure Service Bus
- RabbitMQ …

Einsatz

- Event bus e.g. between Microservices
- Exchange of messages
- Streaming of data

RabbitMQ

- Supports messaging protocols like AMQP 0.9.1/1.0, MQTT…

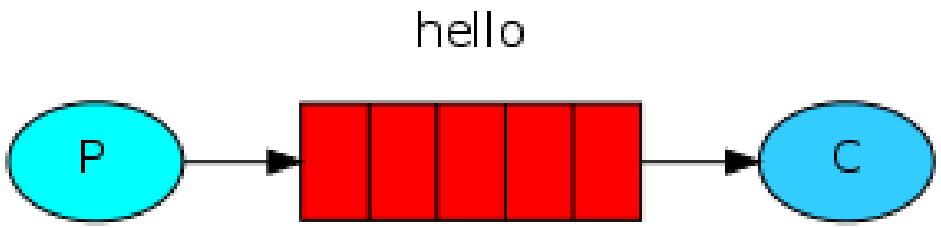


Quelle: https://de.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol; https://www.amqp.org/; https://de.wikipedia.org/wiki/RabbitMQ

Advanced Message Queuing Protocol (AMQP)

Send/receive

+ Übung 10-AMPQ-RabbitMQ
+ Docker image für den RabbitMQ-Broker
- docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.9-management



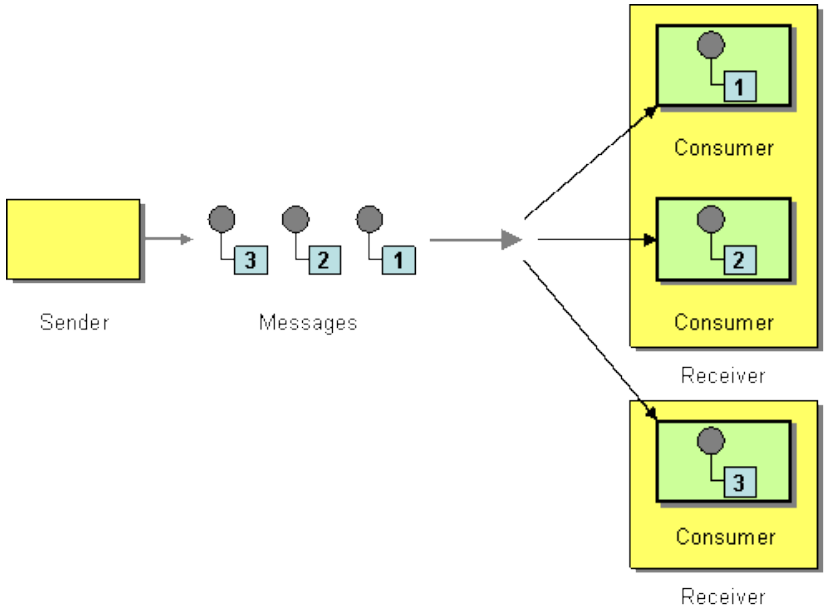
Quelle: https://www.rabbitmq.com/download.html; https://www.rabbitmq.com/getstarted.html

Advanced Message Queuing Protocol (AMQP)

Work Queues

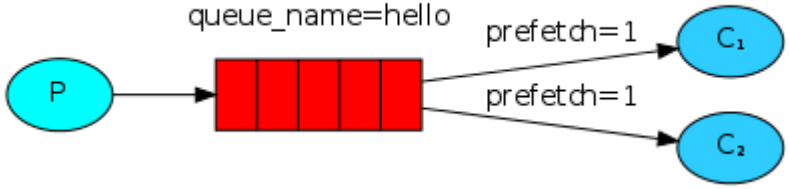
+ Verteilung zeitintensiver Tasks auf verschiedene Worker

Competing Consumers Pattern



+ Die Queue wird geleert, indem an den nächsten verfügbaren Empfänger eine Nachricht geschickt wird (round-robin). Als Standardverhalten wird eine Nachricht nach dem Verschicken gelöscht.
+ Optional: Acknowledgement durch den Empfänger. Sendet der Empfänger kein Acknowledgement (z.B. durch einen Verbindungsabbruch), so sendet der Server die Nachricht an einen anderen Empfänger.

+ Übung 11-AMPQ-RabbitMQ-Worker-Queue
+ Docker image für den RabbitMQ-Broker
- docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.9-management

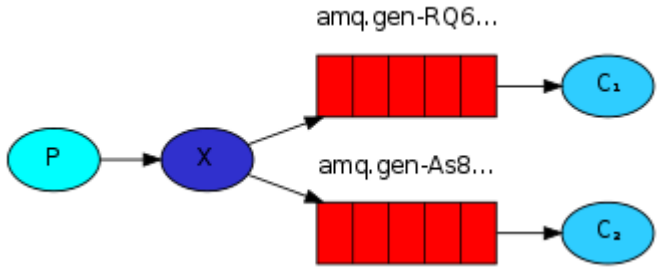


Quelle: https://www.enterpriseintegrationpatterns.com/patterns/messaging/CompetingConsumers.html

Advanced Message Queuing Protocol (AMQP)

Verteilung einer Nachricht auf mehrere Empfänger

+ Publish/subscribe
+ Der Sender (Producer) schickt die Nachricht an einen Exchange. Der Exchange leitet die Nachricht weiter. Der Producer muss nicht wissen, was mit der Nachricht passiert.



+ Die Nachricht kann über einen Exchange vom Typ “fanout” an beliebig viele Queues weitergeleitet werden (broadcast).

+ Übung 12-AMPQ-RabbitMQ-publish-subscribe
+ Docker image für den RabbitMQ-Broker
- docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.9-management

Quelle: https://www.enterpriseintegrationpatterns.com/patterns/messaging/CompetingConsumers.html

Übungsaufgabe 6

Performance-Test

+ MQTT
- Erstellen Sie in python mit MQTT (Paket paho.mqtt.client) einen publisher und einen subscriber. Nutzen Sie z.B. mosquitto als Server.
- Testen Sie, wie viele Nachrichten (bestehend aus einem Zeichen) pro Sekunde maximal übertragen werden können.
+ AMQP
- Erstellen Sie in python mit RabbitMQ (Paket pika) einen publisher und einen subscriber.
- Testen Sie, wie viele Nachrichten (bestehend aus einem Zeichen) pro Sekunde maximal übertragen werden können.

+ MQTT (MQTTnet) (optional)
- Erstellen Sie in .NET 5.0 mit der Bibliothek MQTTnet einen publisher und einen subscriber. Nutzen Sie MQTTnet als Server.
- Testen Sie, wie viele Nachrichten (bestehend aus einem Zeichen) pro Sekunde maximal übertragen werden können.
- https://github.com/chkr1011/MQTTnet/wiki
- https://github.com/chkr1011/MQTTnet

Zusammenfassung

MQTT

+ Einfache, stark skalierbare Kommunikation
+ Publish/Subscribe-Mechanismus zur entkoppelten Kommunikation zwischen verschiedenen Teilnehmern
+ Einschränkungen gegenüber AMQP, z.B. erfolgt immer ein Broadcast an alle Subscriber zu einem Topic
+ „Exactly once“ ermöglicht garantierte Übertragung ohne Berücksichtigung der Verarbeitung

AMQP

+ Komplexer in der Handhabung
+ Größere Flexibilität, z.B. Publish/Subscribe, Competing Consumers Pattern (round-robin) programmierbar
+ Acknowledgement garantiert Übertragung und Verarbeitung eines Tasks durch einen Client

+ Nächste Vorlesung: OPC UA

Quelle: https://www.enterpriseintegrationpatterns.com/patterns/messaging/CompetingConsumers.html

Hochschule Karlsruhe
University of
Applied Sciences

www.h-ka.de