Data Economy

Student: Martin Johannes Brucker

Student Number:: Mj,Brucker@stud.dhsh.de

E-Mail: Martin.Brucker@student.fh-kiel.de

Prof. Dr. Michael Prange

Deadline 17.05.2024

Die Abgabe ist

x ein erster Versuch

☐ ein Wiederholungsversuch

1 Abstract

This document provides an in-depth overview of a sample setup involving the FIWARE Orion Context Broker, QuantumLeap REST Service, and Crate time series database. Our focus will be on creating a real-time parking spot simulator and performing real-life weather querying using the OpenWeather API. The setup adheres to the NGSI-v2 API standard and serves as a robust proof of concept for future development.

FIWARE Orion Context Broker: This component acts as a central hub for managing context information. It allows us to handle real-time data from various sources efficiently.
QuantumLeap REST Service: QuantumLeap is responsible for storing historical context data. It enables us to analyze trends, patterns, and historical events related to parking spots and weather conditions.
Crate Time Series Database: CrateDB is our choice for storing time-series data. Its scalability and performance make it ideal for handling large volumes of real-time data.
Implementation Details:
We'll set up the architecture using Docker containers orchestrated by a docker-compose configuration. The .yml files define the services, networks, and volumes required for seamless deployment.
The real-time parking spot simulator will generate random parking spot data such as availability, occupancy, and cappacity. We'll simulate scenarios such as parking spot arrivals and departures.
For weather querying, we'll integrate the OpenWeather API. By fetching real-time weather data (temperature, humidity, precipitation, etc.), we can correlate it with parking spot usage patterns.

Benefits:
Rapid Deployment: The Docker-based approach ensures quick setup and replication across different environments.
Scalability: As the system grows, we can easily scale the architecture horizontally by adding more containers.
Interoperability: The NGSI-v2 API standard allows seamless integration with other FIWARE components and third-party services.

This ecosystem serves as a foundation for further development, whether on local servers or in the cloud. Feel free to customize and extend it according to your specific use case! 😊

2 FIWARE and NGSI-v2/LD

The Internet of Things (IoT) is rapidly expanding, connecting a vast array of devices and sensors that generate a continuous stream of data. Managing this data effectively is crucial for unlocking the full potential of IoT applications. FIWARE and its Next Generation Service Interface (NGSI) address this challenge by providing a standardized approach to context management.

FIWARE is an open-source initiative focused on fostering the development of smart solutions in various domains, including smart cities, industries, and agriculture. It offers a comprehensive set of open APIs and software components that enable developers to build interoperable and scalable applications.

A key component of FIWARE is the Context Broker, which acts as a central repository for storing and managing context information. This information can include sensor data, device status, location information, and any other relevant data points that describe the state of a system or environment.

NGSI, or Next Generation Service Interface, defines a set of APIs that enable applications to interact with the FIWARE Context Broker. These APIs provide a standardized way to:

Register data providers: Devices, sensors, and other data sources can register with the Context Broker to publish their context information.

Manage context entities: Entities represent real-world objects or concepts within the context information model. NGSI allows creation, retrieval, update, and deletion of entities.

Manage context attributes: Entities have attributes that represent specific properties of interest. NGSI enables management of attributes and their associated values.

Subscribe to context updates: Applications can subscribe to receive notifications when context information changes.

Source: NGSI-LD How to - Fiware-DataModels

Schema.org - Schema.org

NGSI-LD FAQ - Fiware-DataModels

3 REST Quantumleap and Crate time series database

The ever-growing realm of the Internet of Things (IoT) generates a constant stream of time-series data – sensor readings, device statuses, and environmental measurements, to name a few. Effectively storing, querying, and analyzing this data is essential for unlocking the true potential of IoT applications. FIWARE QuantumLeap steps in as a bridge between NGSI-based context management and time series databases like CrateDB.

QuantumLeap is a FIWARE component that acts as a RESTful interface for storing and retrieving time series data associated with NGSI entities. It essentially translates NGSI context information into a format suitable for time series databases, enabling developers to leverage the power of specialized databases for temporal data management.

NGSI-compatible: QuantumLeap seamlessly integrates with NGSI v2 APIs. Developers can use familiar NGSI operations to manage entities and their associated time series data.

CrateDB is a distributed SQL database specifically designed for handling time series data. It offers several advantages for managing large volumes of temporal data:

Scalability: CrateDB can scale horizontally by adding more nodes to the cluster, enabling it to handle ever-increasing data volumes efficiently.

Real-time Ingestion: CrateDB is optimized for real-time data ingestion, making it ideal for capturing and storing sensor data streams from IoT devices.

SQL Queries: CrateDB allows querying time series data using familiar SQL syntax, making it accessible to developers with existing SQL knowledge.

Working Together: QuantumLeap and CrateDB in Action

Here's a scenario where QuantumLeap and CrateDB work together:

Sensor Data Generation: An IoT sensor attached to a piece of machinery continuously transmits readings on temperature, vibration, and power consumption.

Data Submission via NGSI: The sensor data is formatted according to NGSI v2 specifications and sent to the FIWARE Context Broker.

QuantumLeap Intercepts: QuantumLeap acts as an intermediary, receiving the NGSI data stream.

Data Transformation: QuantumLeap transforms the NGSI data into a format compatible with CrateDB, potentially extracting timestamps and relevant attributes.

Storage in CrateDB: The transformed data is then stored in CrateDB, creating a time series record for each sensor reading.

NGSI-based Queries: Developers can use NGSI queries to retrieve historical sensor data from CrateDB through QuantumLeap. For example, they could query for temperature readings from the past hour or vibration data exceeding a certain threshold within a specific timeframe.

This combined approach allows developers to leverage the benefits of NGSI for context management and CrateDB's strengths in time series data handling.

4 Docker-Compose

Docker Compose is a tool for defining and running multi-container applications. It allows you to specify the services (containers) that make up your application and how they interact with each other.

In the context of FIWARE, you can use Docker Compose to run Orion (Context Broker), QuantumLeap, and CrateDB all within separate containers. Here's a breakdown:

Containers: Each service (Orion, QuantumLeap, CrateDB) runs in its own isolated container, ensuring they don't interfere with each other or the host system.

Docker Compose File: This file defines the configuration for each container, including the image to use (e.g., fiware/orion for Orion), ports to expose, environment variables, and dependencies between services (e.g., QuantumLeap relying on CrateDB).

Benefits: Docker Compose simplifies deployment and management. You can start/stop all services with a single command and easily scale your application by adding more containers.

Here's what it looks like in practice:

Create a docker-compose.yml file: This file defines the configuration for Orion, QuantumLeap, and CrateDB containers, including their images, ports, and any necessary environment variables.

Run docker-compose up -d: This command instructs Docker Compose to build the images (if needed) and start all the defined services in detached mode (background).

Enjoy a running FIWARE stack: Orion, QuantumLeap, and CrateDB are now operational within their respective containers, working together to manage context information and time series data.

5 Parking Spot Simulator

The Parking Spot Simulator is a Python application designed to interact with the Orion Context Broker to manage parking spot data. Here's an abstract explanation of its structure:

Context Broker Connection:

The program establishes a connection to the Orion Context Broker, a key component for managing context information in smart applications.

Entity Management:

The simulator defines the structure and attributes of parking spot entities.

It checks if the specified entity type (in this case, parking spots) exists in the Context Broker and creates it if necessary.

Simulation of Parking Spots:

The program is designed to simulate ten parking spots.

It can dynamically create these parking spot entities in the Context Broker, although this part is optional and can be commented out or included based on requirements.

Status Updates:

The core functionality of the simulator is to continuously update the status of these parking spots.

It randomly determines whether each parking spot is "free" or "occupied" and updates the Context Broker accordingly.

Continuous Operation:

The application runs indefinitely, simulating real-time changes in parking spot statuses, which are communicated to the Context Broker.

In essence, this simulator acts as an interface between a simulated physical environment (the parking spots) and the Orion Context Broker, facilitating real-time updates and management of parking spot data.

6 Weather Collector

The Weather Data Integration System is a Python application designed to fetch weather data and communicate it to the Orion Context Broker. Here's an abstract explanation of its structure:

Context Broker Connection:

The program establishes a connection to the Orion Context Broker to manage and update weather-related context information.

External Data Fetching:

It retrieves weather data from an external API (OpenWeatherMap) using specified city ID and API key.

The data includes key weather metrics such as temperature and humidity.

Data Transformation and Transmission:

The fetched weather data is formatted into a payload compatible with the Orion Context Broker.

This payload includes the current temperature and humidity, labeled and typed according to the Context Broker's requirements.

Continuous Data Update:

The application runs in an infinite loop, continuously fetching new weather data at regular intervals (every 5 minutes).

Each new set of weather data is sent to the Orion Context Broker, ensuring the context information remains up-to-date.

Execution Block:

The script initiates the continuous data fetching and updating process when executed as the main module (if __name__ == "__main__":).

This structure allows the system to act as a bridge between real-time weather data and the Orion Context Broker, facilitating ongoing updates and integration of external environmental data into a smart application context.

7 Visualisation

The Parking Spot Visualization System is a Jupyter Notebook application designed to fetch and visualize parking spot data from the Orion Context Broker.

Context Broker Connection:

Purpose: Establish a connection to the Orion Context Broker to manage and retrieve parking spot context information.

Implementation: The application uses the Orion Context Broker's REST API to query parking spot entities of a specified type (ParkingSpot).

Data Fetching:

Purpose: Retrieve parking spot data from the Orion Context Broker.

Implementation: The application sends a GET request to the Orion Context Broker, requesting entities of type ParkingSpot.

Data: The response includes key attributes such as the status of each parking spot (e.g., free or occupied).

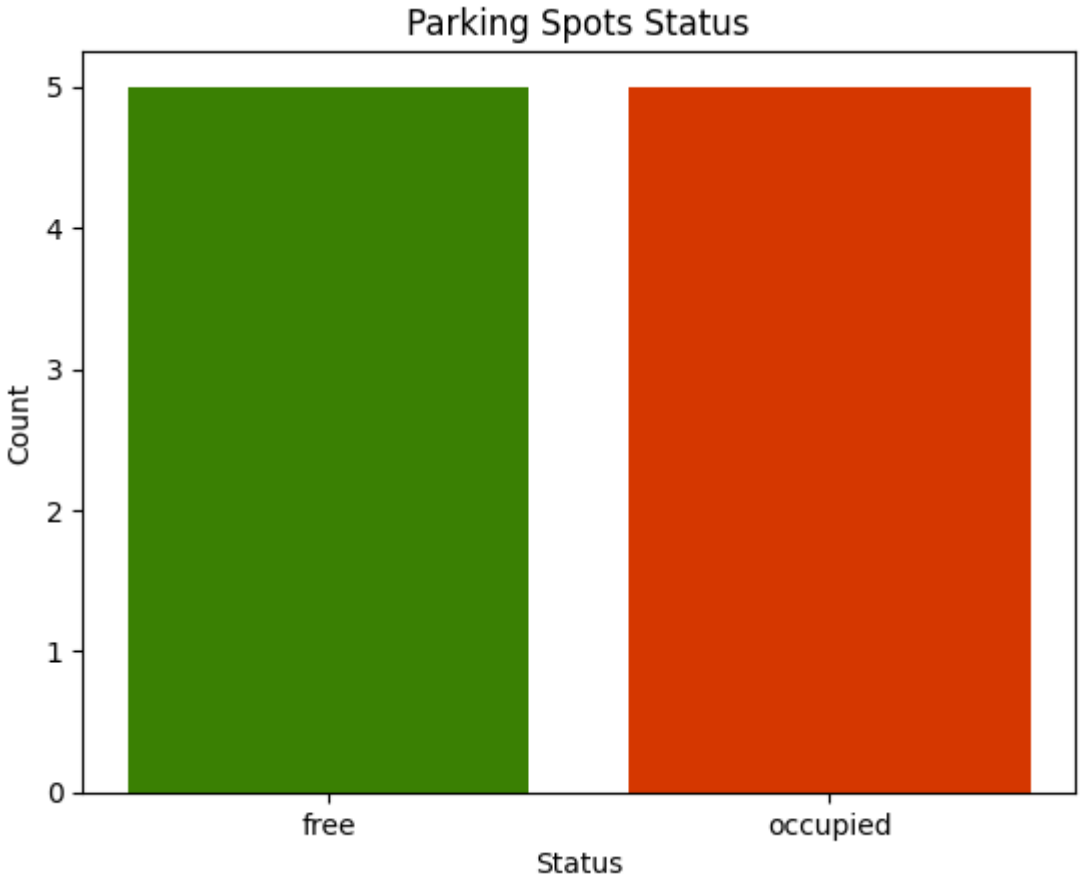
Data Visualization:

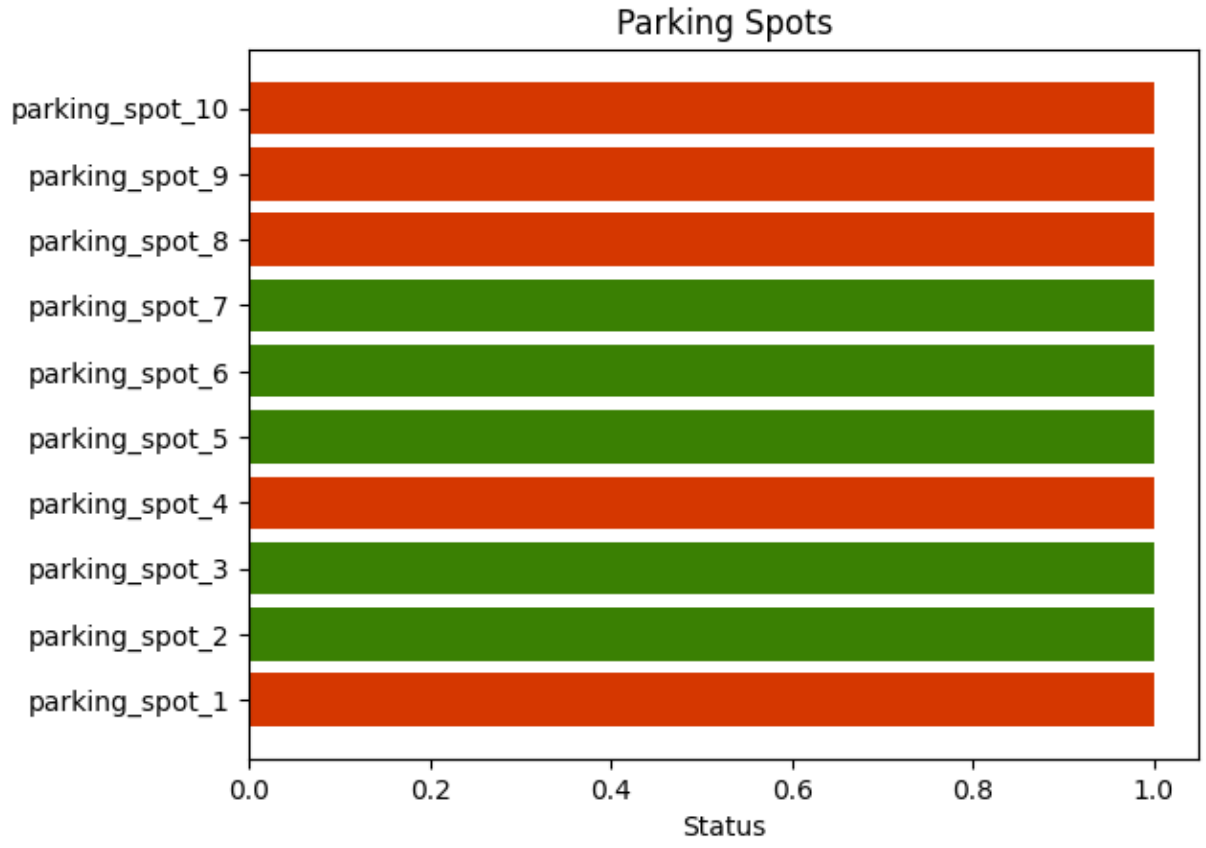
Purpose: Process and visualize the fetched parking spot data.

Implementation: The application uses matplotlib to generate visualizations.

Overall Status Visualization: A bar chart displays the count of free and occupied parking spots.

Individual Spot Status Visualization: A horizontal bar chart shows the status of each individual parking spot.


Parking Spots Status
Count
0
1
2
3
4
5
free
occupied
Status


Parking Spots
parking_spot_10
parking_spot_9
parking_spot_8
parking_spot_7
parking_spot_6
parking_spot_5
parking_spot_4
parking_spot_3
parking_spot_2
parking_spot_1
0.0
0.2
0.4
0.6
0.8
1.0
Status

Installation Guide

Carry out the commands in the following order

1. Inside / docker-compose up -d
2. Inside /parking_simulator docker build -t parking-simulator .
3. Inside /parking_simulator docker run parking-simulator
4. Inside /weather-collector docker build -t weather-collector .
5. Inside /weather-collector docker run weather-collector
6. For visualisation use the .ipynb inside /visualisation