Boosting Research for a Smart and Carbon Neutral Built Environment with Digital Twins – **SmartWins**

Notable case studies highlighting Big Data applications in smart buildings

Nikos Tsalikidis, CERTH

CERTH SmartWins Training Sessions: Day1
23 April 2024
Thessaloniki

Funded by the European Union

This project has received funding from the European Union's Horizon research and innovation programme under grant agreement No 101078997

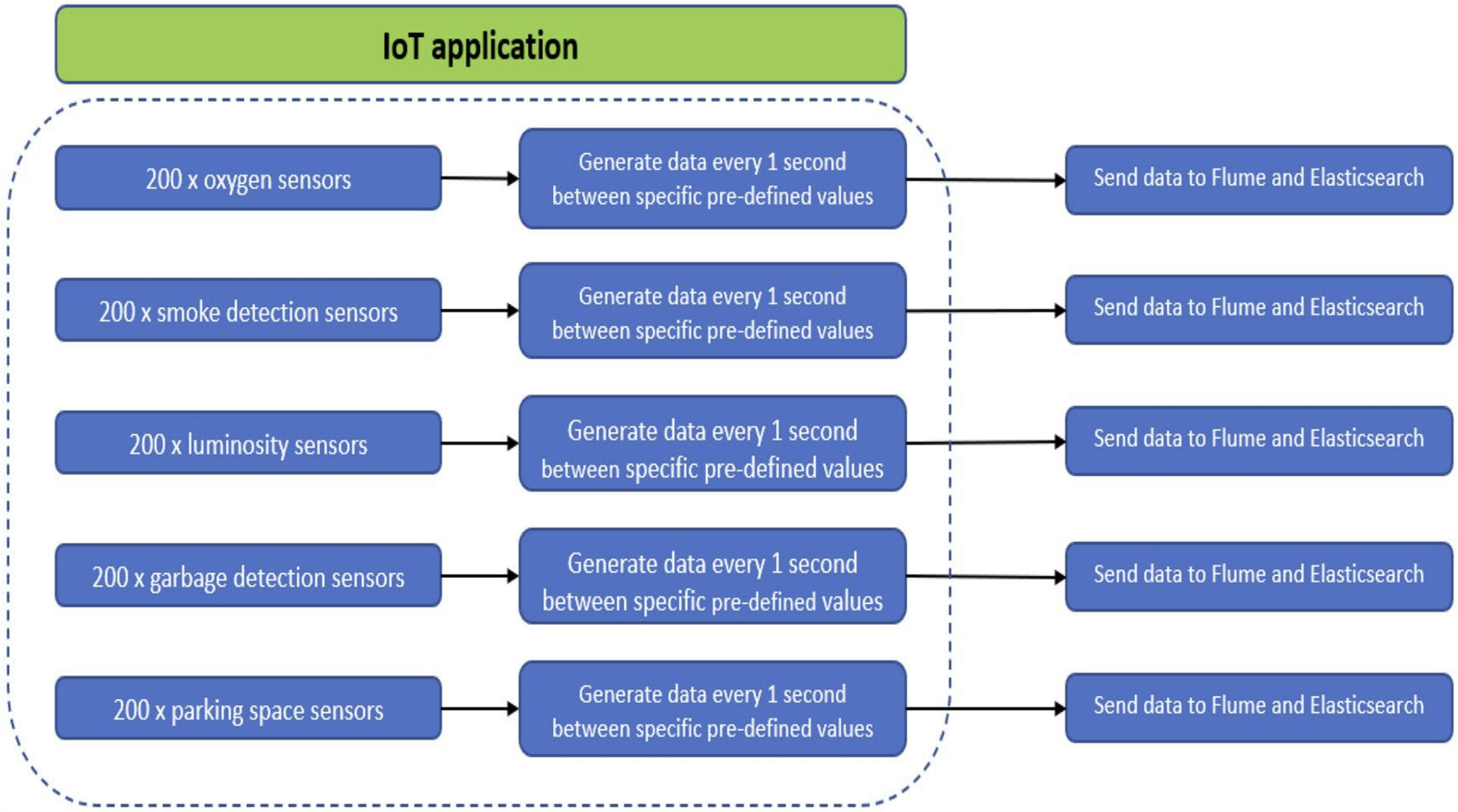
Case study(CS) 1:

Reference Architecture Implementation, Sydney Australia

CS1 – Data source

- Sensor data from University of Technology Sydney (UTS) building 11 (Dat available on UTS's web portal)
- Comprised of historical data for two types of sensors (oxygen sensors and gas detection sensors) for one of the floors of building 11.
- To test a smart building application scenario by introducing big data pipelining, storage and analysis tools, the available real-time streaming sensor data were limited.
- **Hence, IoT sensors were virtualized by simulating sensor data.** 5 types of virtual sensors(1000 total sensors):
  - IoT oxygen, Smoke detection, Luminosity sensors, Parking spaces, Garbage detection sensors
  - Assumed to be deployed @200 distinct locations (rooms or levels)
  - Sensors simultaneously generate the data (1 s interval) which are served by ten Flume agents running in parallel.

CS1-Sensor data flow



SmartWins

CS1-Data ingestion & storage

- **Streaming data**: Virtual sensor application pushes to two destinations:
  1. To Flume agents to enable near-real-time ingestion of data into HDFS.
  2. To Elasticsearch to enable near real-time data visualization using Kibana.

  Data stored in Elasticsearch as indexed documents using APIs where Elasticsearch adds a searchable reference to the document in the cluster's index

- **Static data**: once extracted in the .csv format are ingested and stored in HDFS

CS1-Data ingestion & storage -Flume configuration files

```
# sensor100.conf: A single-node Flume configuration

# Name the components on this agent
 a1.sources = r1
 a1.sinks = k1
 a1.channels = c1

# Describe/configure the source
 a1.sources.r1.type = netcat
 a1.sources.r1.bind = 127.0.0.1
 a1.sources.r1.port = 5885

# Describe the sink
 a1.sinks.k1.type = hdfs

# Use a channel which buffers events in memory
 a1.channels.c1.type = memory
 a1.channels.c1.capacity = 1000
 a1.channels.c1.transactionCapacity = 100

#Customizing sink for hdfs
a1.sinks.k1.hdfs.path = hdfs://quickstart.cloudera:8020/user/cloudera/virtualsensor1
a1.sinks.k1.hdfs.filePrefix = netcat
a1.sinks.k1.hdfs.fileType = DataStream

# Bind the source and sink to the channel
 a1.sources.r1.channels = c1
 a1.sinks.k1.channel = c1
```

- Each agent will roll over files after every 30 s, finish writing to it and create a new file in HDFS every 30 s as. tmp file
- 3 key components:
  - ‘Source’ binds to the incoming source of data (binds to a TCP IP)
  - ‘Sink’ binds to the destination where the data need to be stored.
  - ‘Channel’ is a memory channel with a capacity and transaction capacity of 1000 and 100, respectively.
    - Capacity sets the maximum stored events,
    - Transaction capacity determines maximum events exchanged with a source or sink per transaction.

CS1-Data Analysis

- Used PySpark—Spark Python API
- Analyzes sensor incoming data, triggering appropriate controls
- **Example**: If sensor value is <14 (indicating low oxygen levels), the algorithm triggers the HVAC system to turn on.

```python
import time
threshold_oxygen=14
threshold_smoke=19
threshold_luminosity=16
threshold_garbage=20
while(1):
        dataRDD = sc.textFile("/user/cloudera/data/netcat.*")
        for line in dataRDD.collect():
                id,value,postcode=line.split(",")
                id=int(id)
                value=int(value)
                postcode=int(postcode)
                if(id<201):
                        if(value<threshold_oxygen):
                                print ("HVAC system " ,id, " turned ON")
                        else:
                                print("Oxygen level at ",id," OKAY")
                if(200<id<401):
                        if(value>threshold_smoke):
                                print ("Fire Alarm " ,id-200, " turned ON")
                        else:
                                print("No fire at " , id-200)
                if(400<id<601):
                        if(value>0):
                                print ("Parking " ,id-400, " is occupied")
                        else:
                                print("Parking " ,id-400, " is empty")
                if(600<id<801):
                        if(value<threshold_luminosity):
                                print ("Lights at " ,id-600, " turned ON")
                        else:
                                print("Luminosity level at" ,(id-600), " OKAY")

                if(id>800):
                        if(value>threshold_garbage):
                                print ("Garbage at " ,id-800, " is Full")
                        else:
                                print("Garbage at " ,id-800, " has space")

        time.sleep(10)
```

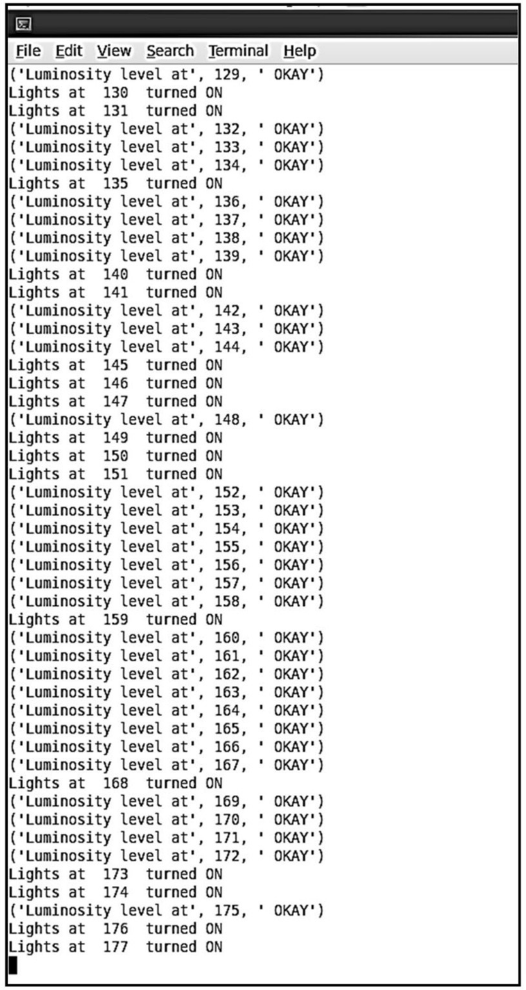
File Edit View Search Terminal Help

```
('Luminosity level at', 129, ' OKAY')
Lights at  130  turned ON
Lights at  131  turned ON
('Luminosity level at', 132, ' OKAY')
('Luminosity level at', 133, ' OKAY')
('Luminosity level at', 134, ' OKAY')
Lights at  135  turned ON
('Luminosity level at', 136, ' OKAY')
('Luminosity level at', 137, ' OKAY')
('Luminosity level at', 138, ' OKAY')
('Luminosity level at', 139, ' OKAY')
Lights at  140  turned ON
Lights at  141  turned ON
('Luminosity level at', 142, ' OKAY')
('Luminosity level at', 143, ' OKAY')
('Luminosity level at', 144, ' OKAY')
Lights at  145  turned ON
Lights at  146  turned ON
Lights at  147  turned ON
('Luminosity level at', 148, ' OKAY')
Lights at  149  turned ON
Lights at  150  turned ON
Lights at  151  turned ON
('Luminosity level at', 152, ' OKAY')
('Luminosity level at', 153, ' OKAY')
('Luminosity level at', 154, ' OKAY')
('Luminosity level at', 155, ' OKAY')
('Luminosity level at', 156, ' OKAY')
('Luminosity level at', 157, ' OKAY')
('Luminosity level at', 158, ' OKAY')
Lights at  159  turned ON
('Luminosity level at', 160, ' OKAY')
('Luminosity level at', 161, ' OKAY')
('Luminosity level at', 162, ' OKAY')
('Luminosity level at', 163, ' OKAY')
('Luminosity level at', 164, ' OKAY')
('Luminosity level at', 165, ' OKAY')
('Luminosity level at', 166, ' OKAY')
('Luminosity level at', 167, ' OKAY')
Lights at  168  turned ON
('Luminosity level at', 169, ' OKAY')
('Luminosity level at', 170, ' OKAY')
('Luminosity level at', 171, ' OKAY')
('Luminosity level at', 172, ' OKAY')
Lights at  173  turned ON
Lights at  174  turned ON
('Luminosity level at', 175, ' OKAY')
Lights at  176  turned ON
Lights at  177  turned ON
```

SmartWins

CS1-Data Analysis- Test case: detection of luminosity

Test case	Detection of low luminosity level and autonomously activating smart lights
Context	Luminosity levels fall below the human luminous comfort levels at a specified location in the smart building
Problem	Low luminosity in the smart building may go unnoticed, causing discomfort and posing safety hazards for residents.
Solution	Luminosity level falls and is detected by the IBDMA architecture. The management is notified, and the smart lights are autonomously activated within two minutes
Test metrics	60 min, number of records in data: 60, detection measure: The terminal displayed the message saying the Lights turned ON when the value of the luminosity sensor fell below the threshold
Description	Luminosity sensor detects low levels, sends data to HDFS via Flume. System activates smart lights and notifies building management within two minutes. Smart lights are turned off autonomously when luminosity reaches acceptable levels.
Consequences/ improved performance metrics	The residents enjoy comfortable luminous levels. The smart building management are notified within two minutes if levels fall below the threshold levels



SmartWins

Case study(CS) 2:

Big Building Data (BBData) v2.0,

Fribourg, Switzerland

CS2-Context and location

- Big Building Data (BBData) is an ingestion, processing and sharing system able to scale up to the Big Data expectations of Smart Building environments
- An applied research contribution of HEIA-FR * intended to develop a scalable cloud platform and tools for storing and processing Smart Living Lab's building data
- Fed by 2000 sensors located at the Halle Blue of the BlueFactory site.
- BBData has been used in production

for more than a year

*School of Engineering and Architecture of Fribourg

SmartWins

BBData v1.0

- Sensors generate data (1), which are encoded into JSON records (2) that include a timestamp, a security token and the virtual object ID to which the sensor belongs.
- JSON records sent through a RESTful JavaEE application (3) running on a GlassFish server, the input API, which validates the token and stores the raw record into a a Kafka topic1 (4) before returning a response.
- At the core of BBData run different Flink streaming applications, all connected through Kafka topics.
  - 1st Topic: takes the raw record and "augments" it with metadata (unit, data type, …) stored into a MySQL database (5).
  - 2nd Topic: stores this augmented measure into a Cassandra NoSQL database (6). Optionally, other "processors" can read from Kafka and do side-processing, such as computing live aggregations.
- Users interact with BBData through another RESTful application, the output API (7), to manage the virtual objects, access the data and control the access rights in a fine-grained manner.
- The whole system runs on Hadoop [7].

BBData v2.0

2nd Version of BBData keeps the same concepts w/ simplified architecture and new technologies.

1. Removed dependency on Hadoop
2. APIs federated into a single executable jar, written in Spring Boot/Kotlin(https://spring.io/projects/spring-boot/) → run on Apache server built in the application
   - Simplifies code, maintainability, benefits from active SpringBoot community.
   - Moved part of the processing to the input API itself. Raw measures are instead saved synchronously by the API

Optimizing BBData v2.0: Cassandra, Kafka, and Caching Strategies

The input API now needs to pull metadata from MySQL database and write to Cassandra.

MySQL bottleneck in token validation and metadata retrieval→ Adopt Cassandra and Kafka for optimized performance

Optimizing Input :

- In-memory Key/Value Cache: Choose between internal (e.g., HashMap) or external (Redis Cache).
- Entries store objectId: token, mapping to metadata set (or null for invalid tokens).
- Eviction Mechanism: Triggered on token deletion or object state change.

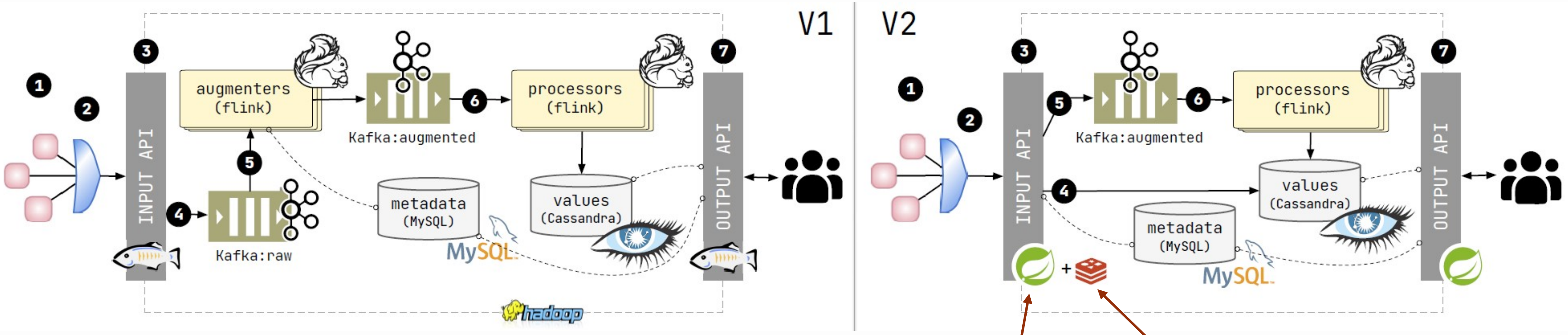
Asynchronous Updates:

- Configurable thread pool for asynchronous statistics updates
- Ensures timely and non-blocking processing of read/write counters.
- Seamless Transition: Switches to synchronous updates if the thread pool is saturated.

Benefits:

- Faster Token Validation and Metadata Retrieval:
  - Reduced latency with in-memory caching.
  - Immediate access to metadata for valid tokens.
- Efficient Data Processing:
  - Cassandra and Kafka streamline data flow, mitigating MySQL bottlenecks.
  - In-memory cache minimizes redundant database queries.
- Real-time Statistics Updates:
  - Asynchronous updates ensure continuous tracking of read/write counters.
  - Smooth transition to synchronous updates maintains accuracy under load.

BBData V1.0 and V2.0



Spring Boot/Kotlin

Spring Data Redis

Case study(CS) 3:

R2M offices Digital Twin, Pavia Italy

CS3 - Context

- Case study: An architecture for defining a DT integrated with Big Data technologies, IoT, and data retrieved from multiple sources.
- End-goal: A 3D model of the offices of an Italian company → designed w/ different digital layers showing information extracted from collected data.
- Motivation: Company seeks to enhance space utilization by implementing a system that monitors temperature, humidity, and air quality→commitment to sustainability and user well-being, focusing on efficient energy management.

Additional tools utilized in CS3

Spaceti (https://landing.spaceti.com/)

- Provides tools for facility optimization, tenant satisfaction, and efficiency
- Smart building platform uses sensors and IoT for real-time data
- MLalgorithms for actionable insights
- Mobile applications for personalized building experiences

OpenWeatherData (https://openweathermap.org/)

- Open and easy-to-use platform for weather-related data. Provides current conditions, forecasts, historical data
- Accessible through API for developers, researchers, and businesses
- Enables innovation and collaboration in weather data usage across industries

Matterport(https://matterport.com/)

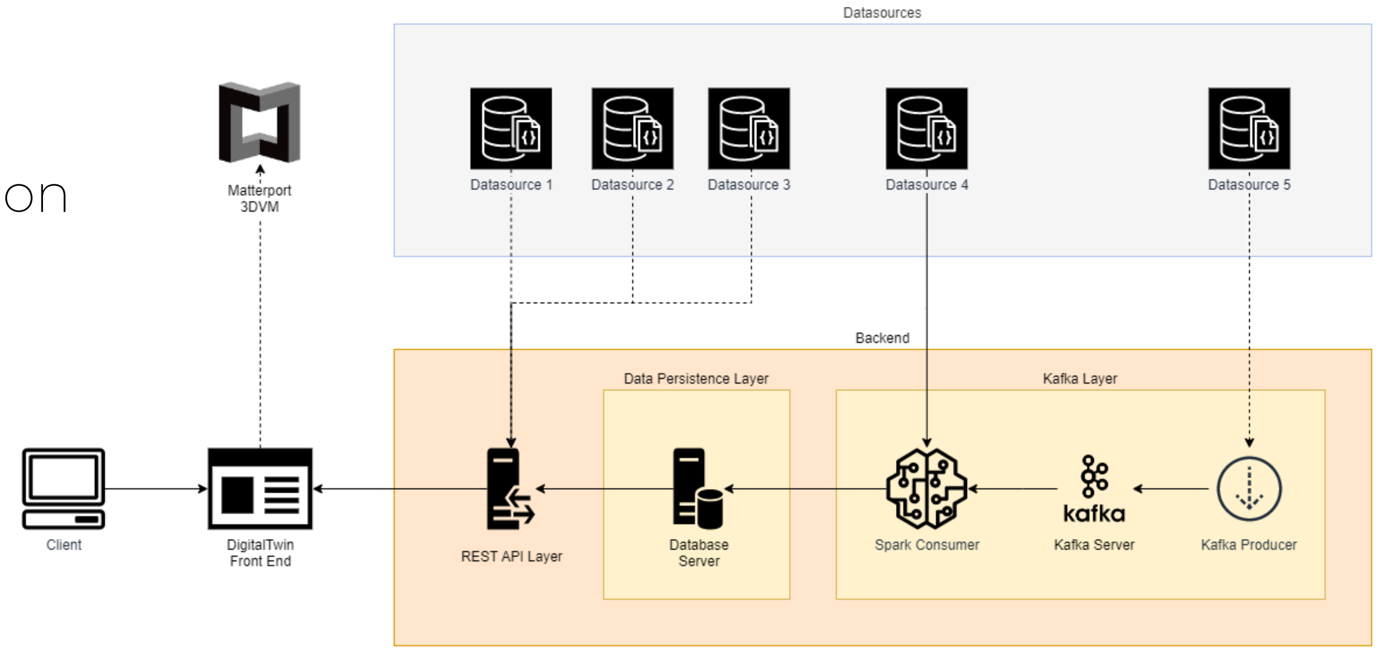
- 3D camera provider and platform for creating immersive/ interactive 3D
- Captures every detail of physical spaces
- Features include measurement tools, virtual reality compatibility, and annotations
- Widely used by real estate agents, architects, and interior designers

CS3- Architectural design

- Front-End: User accesses through the browser (Client)
- Front-End calls the Matterport 3DVM which provides the 3D model with all the metadata.
- Backend: includes the REST API Layer which allows the connection with the data sources
- Support for

REST API Layer:

1. Persistent Layer : stores information
2. Kafka layer: fetches information from data sources in modular way.



CS3 - Backend Architecture Overview

1. **REST API Layer (Express.js):**
    1. Developed using Express.js (https://expressjs.com/)
    2. Unified interface for data interaction.
    3. Exposes endpoints to DigitalTwin FrontEnd.
    4. Empowered to retrieve data from third-party services and query data persistence layer.
2. **Persistent Layer (PostgreSQL):**
    1. Stores raw tweets with metadata.
    2. Stores computed results from ML algorithms applied by the Kafka Consumer.
    3. Associated with data manipulated in Apache Kafka.

SmartWins

CS3 - Kafka Layer (in detail)

Kafka Producer:

- Queries Twitter API for company-related tweets.
- Encodes and sends tweets to Kafka Server for further processing.

Kafka Consumer:

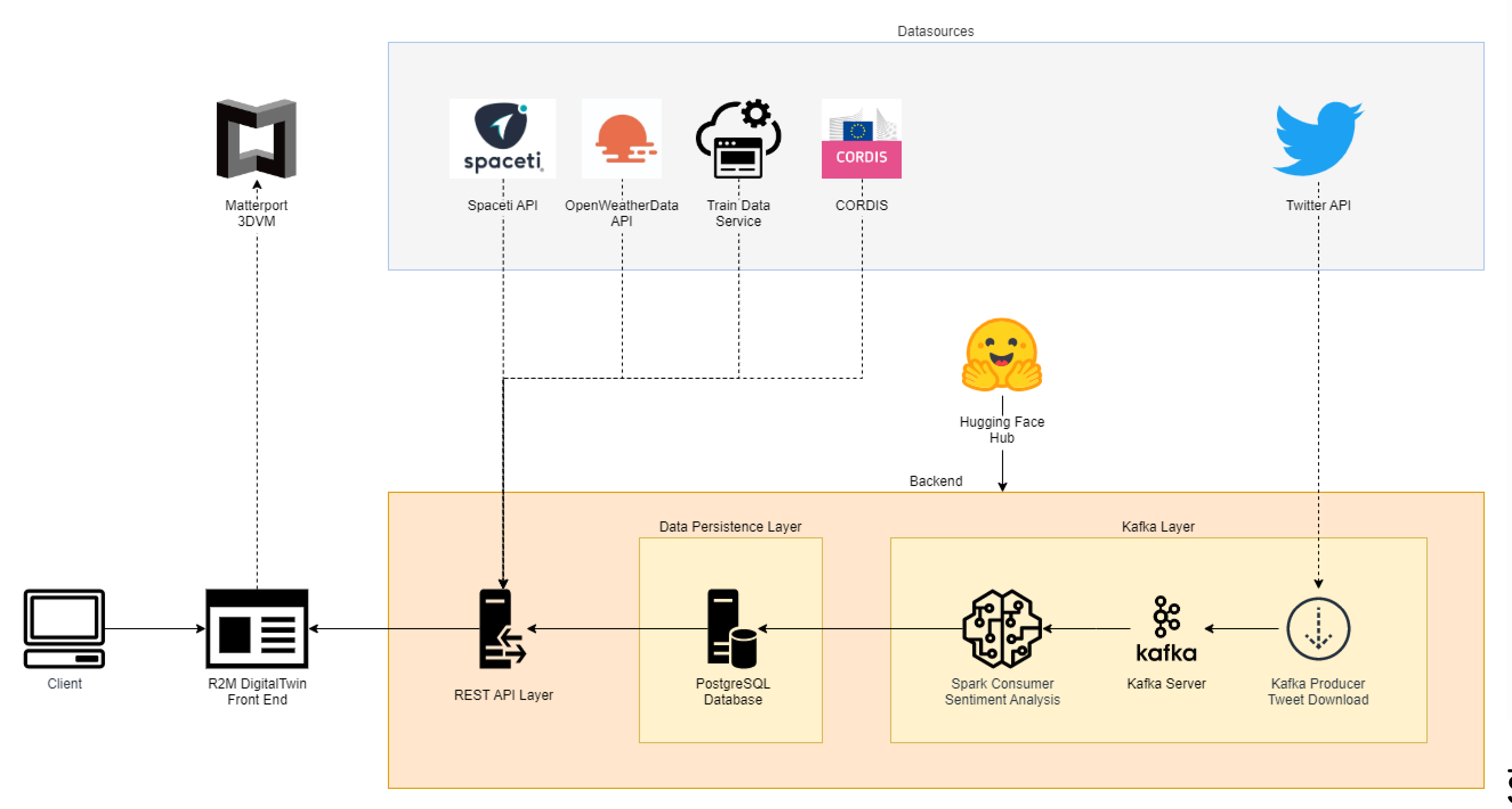
- Spark Streaming app performing sentiment analysis on tweets.
- Labels tweets as positive, neutral, or negative with a reliability score.
- Integrates PostgreSQL for storage and availability to the final application.

Key Features:

- Daily operations ensuring real-time data processing.
- Sentiment analysis leveraging open-source ML models from Hugging Face.
- Reliable categorization of sentiment with associated scores.

SmartWins

Overall schema of the use case

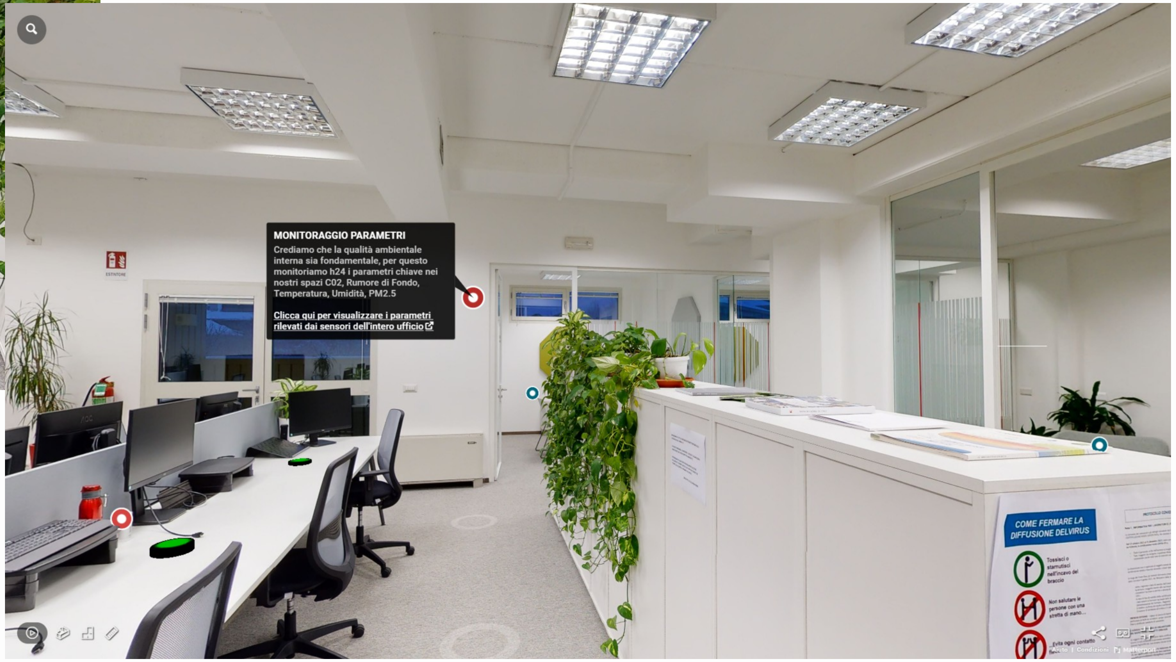


SmarTwins

CS3- Implementation snapshots

Occupation sensor widgets inside of the Digital Twin. At the time of the screenshot, the three seats in front of the respective workstations were free.

Mattertag functionality inside of the Digital Twin.



SmartWins

References

- **CS1**: Bashir, M.R., Gill, A.Q. & Beydoun, G. A Reference Architecture for IoT-Enabled Smart Buildings. SN COMPUT. SCI. 3, 493 (2022). https://doi.org/10.1007/s42979-022-01401-9

- **CS2**: Lucy Linder et al 2021 J. Phys.: Conf. Ser. 2042 012016

- **CS3**: *R. Alonso, R. Locci and D. Reforgiato Recupero, Improving Digital Twin Experience through Big Data, IoT and Social Analysis: anarchitecture and a case study, Heliyon, 9, e24741, doi: https://doi.org/10.1016/j.heliyon.2024.e24741*

Project Partners

kaunas university of technology
ktu 1922

CERTH
CENTRE FOR RESEARCH & TECHNOLOGY HELLAS

POLITECNICO MILANO 1863

Contecht

INNOTROPE

Funded by the European Union

This project has received funding from the European Union's Horizon research and innovation programme under grant agreement No 101078997

SmartWins