

Systems of Systems (SoS)

M.Sc. In Critical Computing Systems Engineering

ISEP/IPP – 2021/22, 2nd semester

Assignment 2:

FIWARE

Pedro Santos

A solid orange horizontal bar spanning the width of the slide, located at the bottom.

Overview of FIWARE



FIWARE

- Why do we need FIWARE?
 - Gathering, publishing, processing and analyzing private and open data at large scale
- What is it?
 - A curated framework of Open Source Platform components to accelerate the development of Smart Solutions
 - Advanced OpenStack-based cloud + rich library of Generic Enablers (GEs)
- Big push by European Commission towards promoting openness and sharing of data
- Built around the concept of CONTEXT!

Context

Boiler

- Manufacturer
- Last revision
- Product id
- Temperature
- Actions



Users

- Name-Surname
- Birthday
- Location
- ToDo list



Street Devices

- Location
- Observations
- Commands



Public Bus T.System

- Location
- Arrival time

APPs / Services / Data Scientist

NGSI API

Context Broker



City

- OpenData
- Users Input

Previously: Silos or Verticals



Higher Efficiency

- Automatization

Intelligence₁

Intelligence₂

Intelligence_N



Higher IT Business

- Common suppliers

Connectivity₁

Connectivity₂

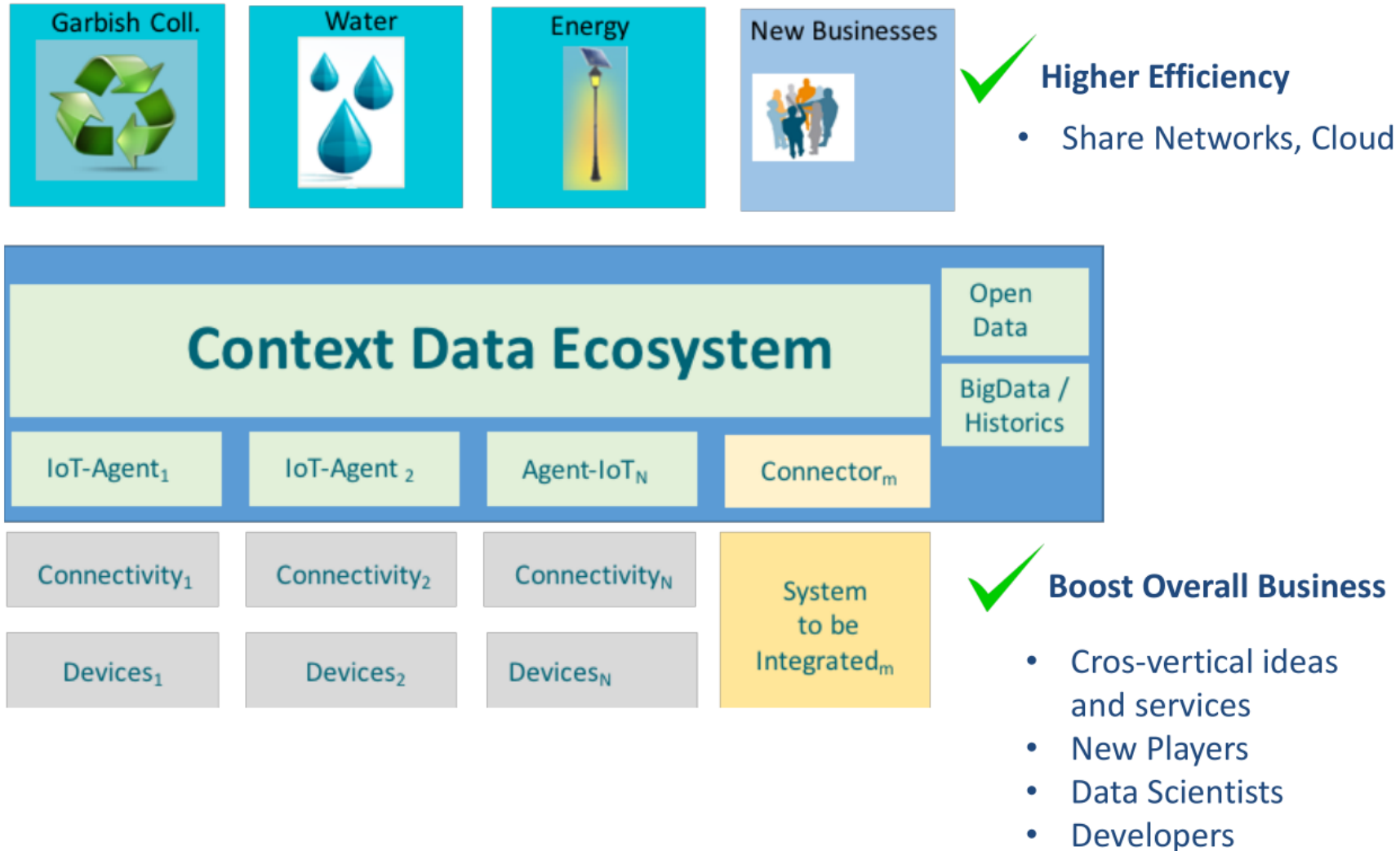
Connectivity_N

Devices₁

Devices₂

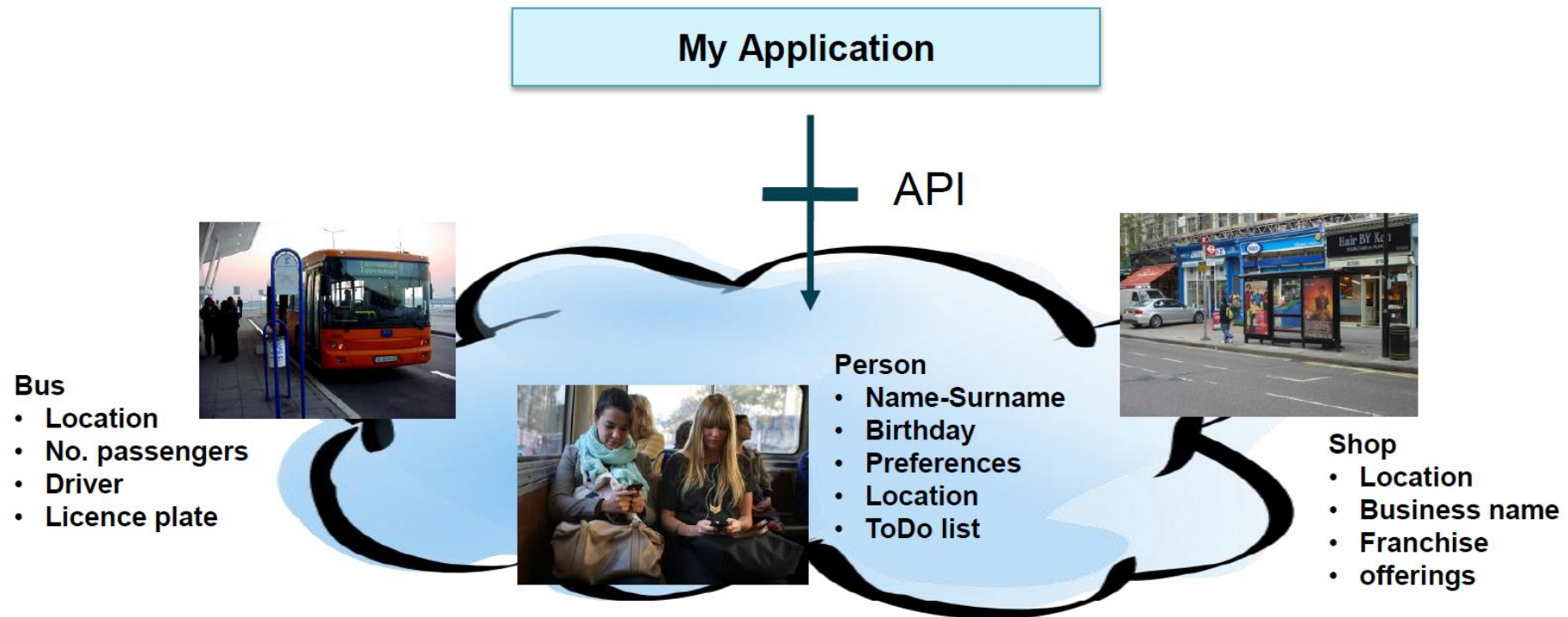
Devices_N

FIWARE: Growth Engine for Local Ecosystems



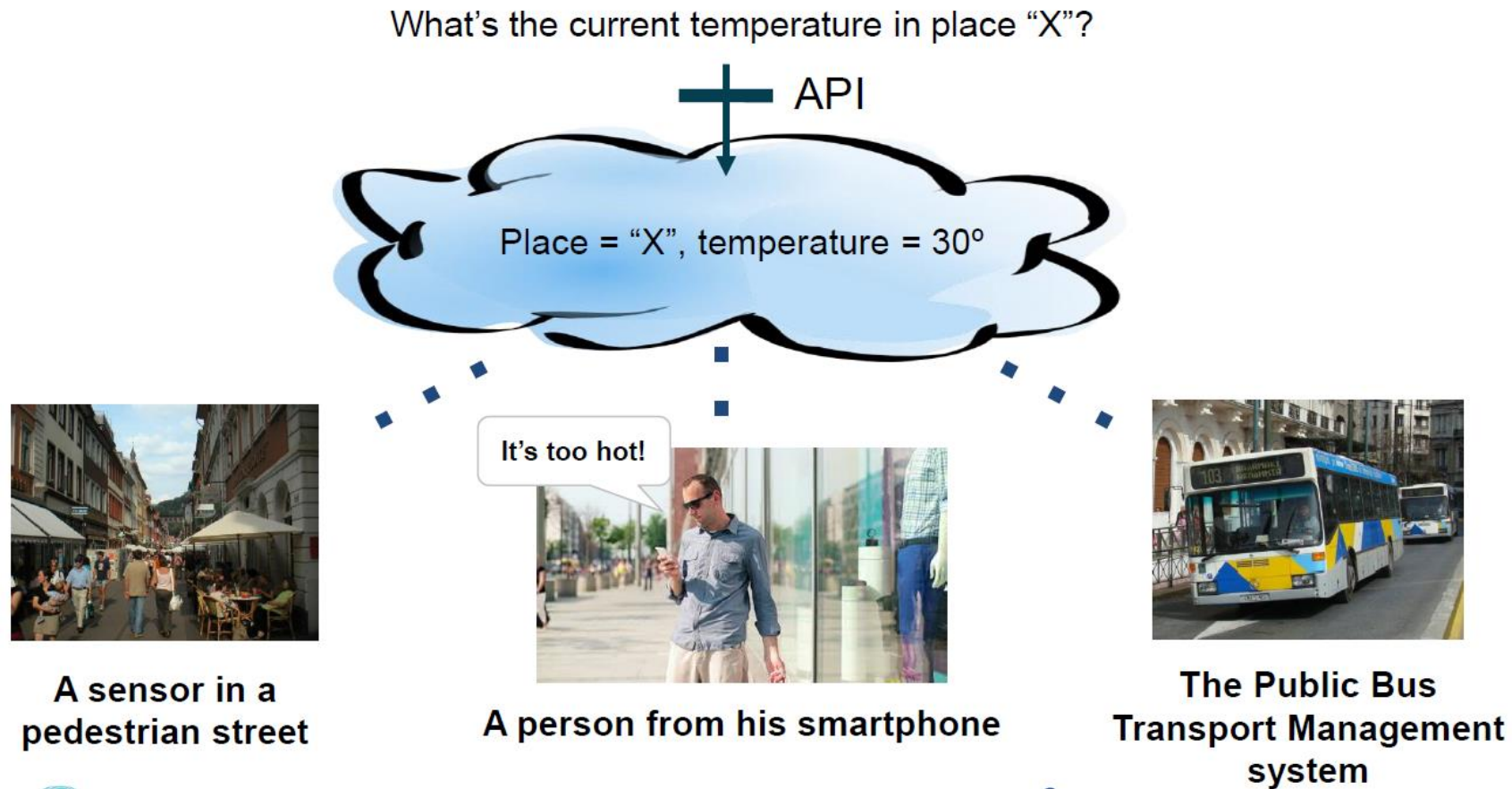
Context as the Key Driver of FIWARE

- A simple yet powerful standard API should be defined that helps programmers to manage Context information.
- Context information refers to the values of attributes characterizing entities relevant to applications



Context as the Key Driver of FIWARE

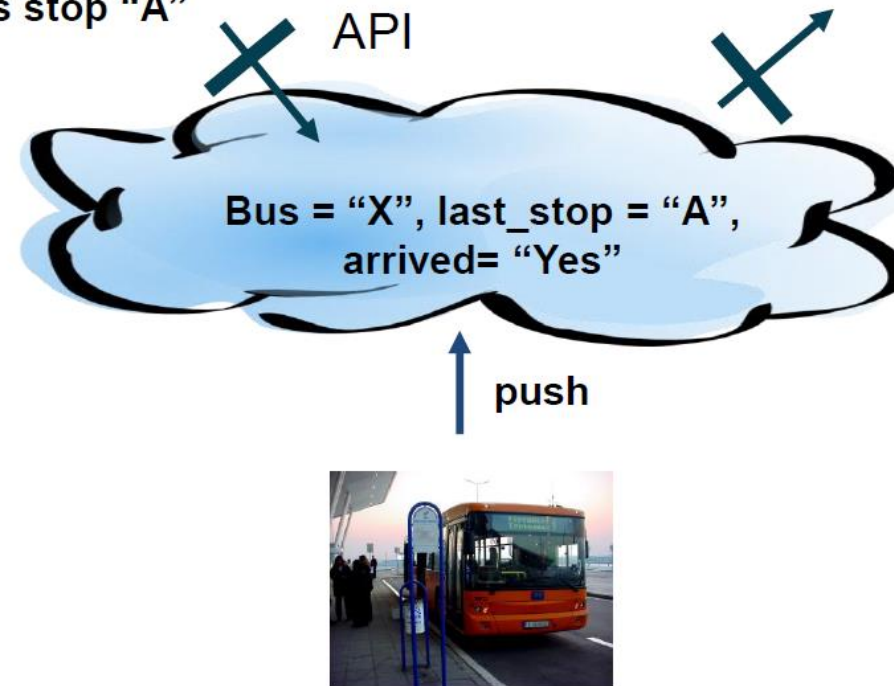
- Context information may come from many sources using different interfaces and protocols ... but programmers should just care about entities and their attributes ...



Context as the Key Driver of FIWARE

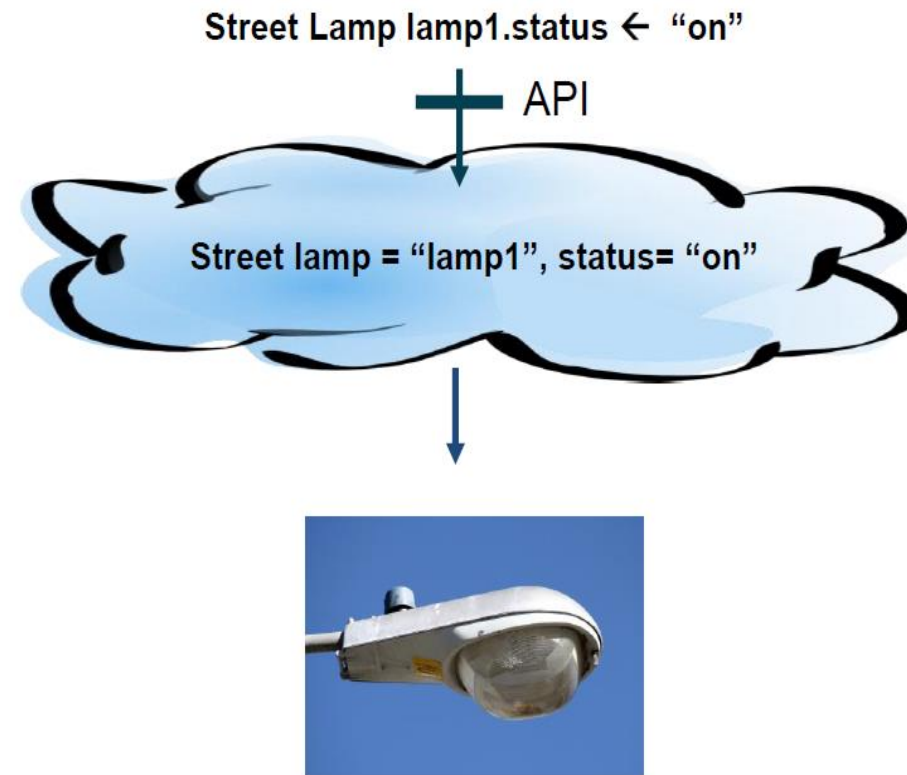
- Programmers may want to get notified when an update on context information takes place

Notify me when bus "X"
arrives at the bus stop "A"



Context as the Key Driver of FIWARE

- Acting on certain devices should be as easy as to change the value of attributes linked to certain entities



Why an open standard platform is required

- Avoid vendor lock-in:
 - Standard Southbound APIs for sensor providers.
 - Standard Northbound APIs offered to applications.
 - Portability among platform providers.
 - Interoperability of solutions enabled by the platform.
- Larger community of developers
 - True innovation.
 - Better prices.
- Not any standard is enough
 - Modularity.
 - Allow different business models.
 - Integration with standard open data platform.
 - Non-intrusive.

FIWARE Generic Enablers (GEs)

- A FIWARE Generic Enabler (GE):
 - Set of general-purpose platform functions available through APIs.
 - Building with other GEs a FIWARE Reference Architecture.
- FIWARE GE Specifications are open (public and royalty-free).
- FIWARE GE implementation (FIWARE GEi):
 - Platform product that implements a given GE Open Spec.
 - There might be multiple compliant GEis of each GE Open Spec.
- At least one open source reference implementation of FIWARE GEs (FIWARE GERis):
 - Well-known open source license.
 - Publicly available Technical Roadmap updated in every release.
- Available FIWARE GEis, GERis and incubated enablers published on the FIWARE Catalogue.

NGSI-LD

Next Generation Services Interface - Linked Data



What is NGSI-LD?

- NGSI-LD is an information model and API for publishing, querying and subscribing to context information.
- Goals:
 - Meant to facilitate the open exchange and sharing of structured information between different stakeholders.
 - Used across application domains such as Smart Cities, Smart Industry, Smart Agriculture, and more generally for the Internet of Things, Cyber-Physical Systems, Systems of systems and Digital Twins.
- Name:
 - The acronym NGSI stands for "Next Generation Service Interfaces", a suite of specifications originally issued by the OMA which included Context Interfaces.
 - The -LD suffix denotes this affiliation to the Linked Data universe.

Linked Data

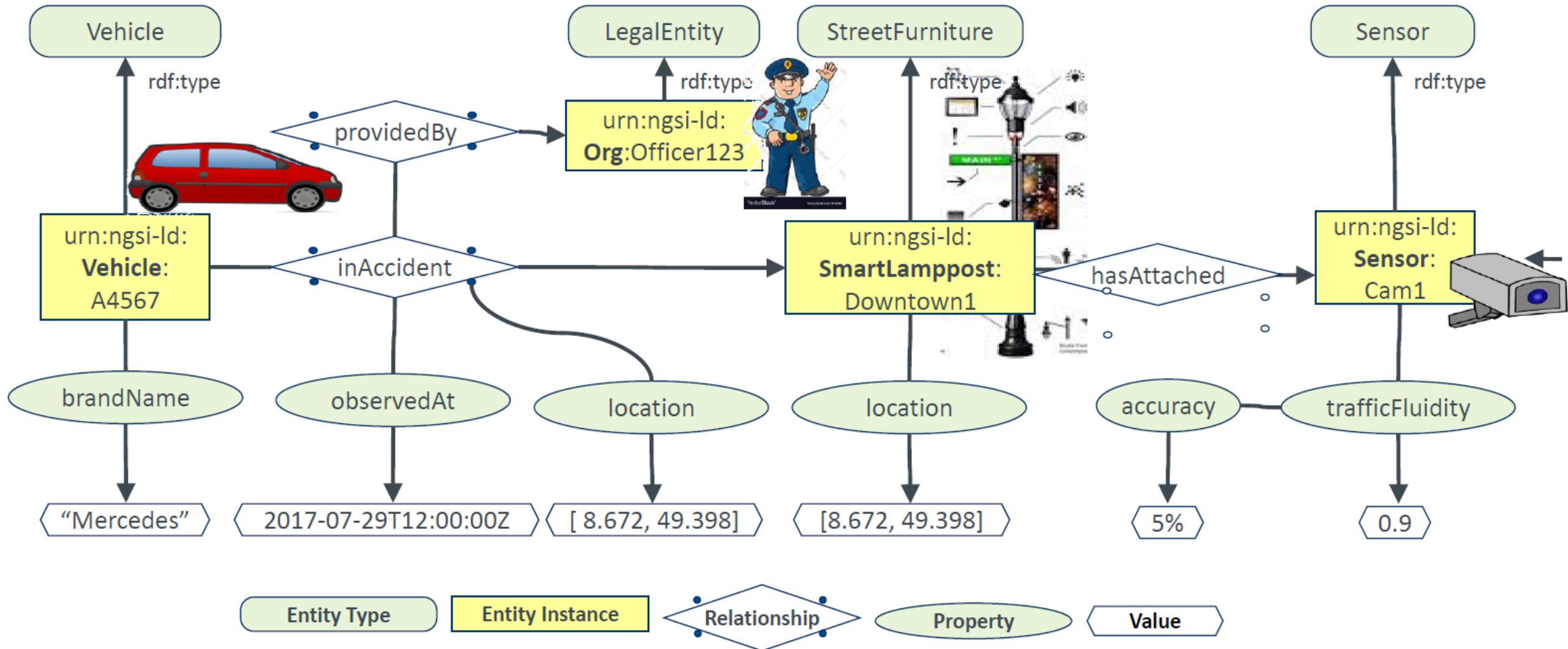
- Linked Data
 - **Linked data** is structured data which is interlinked with other data.
 - It builds upon standard Web technologies such as HTTP, Resource Description Framework (RDF) and URIs.
- JSON-LD
 - A JSON-based method of encoding linked data, designed around the concept of a "context" to provide additional mappings from JSON to a Resource Description Framework model.
 - The context links object properties in a JSON document to concepts in an ontology.
 - In order to map the JSON-LD syntax to RDF, JSON-LD allows values to be coerced to a specified type or to be tagged with a language.
- Relationship NGSI-LD and Linked Data
 - NGSI-LD can be serialized using JSON-LD (in fact, the NGSI-LD specification states that NGSI-LD is based on JSON-LD).
 - The @context in JSON-LD is used to map terms provided as strings to concepts specified as URIs.
 - The Core NGSI-LD (JSON-LD) @context is defined as a JSON-LD @context which contains:
 - The core terms needed to uniquely represent the key concepts defined by the NGSI-LD Information Model
 - The terms needed to uniquely represent all the members that define the API-related Data Types

```
{
  "@context": {
    "name": "http://xmlns.com/foaf/0.1/name",
    "homepage": {
      "@id": "http://xmlns.com/foaf/0.1/workplaceHomepage",
      "@type": "@id"
    },
    "Person": "http://xmlns.com/foaf/0.1/Person"
  },
  "@id": "https://me.example.com",
  "@type": "Person",
  "name": "John Smith",
  "homepage": "https://www.example.com/"
}
```


Information Model NGSI-LD

- The NGSI-LD information model represents Context Information as entities that have properties and relationships to other entities.
- The NGSI-LD meta-model formally defines these the following foundational concepts
 - **NGSI-LD Entity:** the informational representative of something (a *referent*) that is supposed to exist in the real world, outside of the computational platform using NGSI-LD.
 - **NGSI-LD Property:** an instance that associates a characteristic, an NGSI-LD Value, to either an NGSI-LD Entity, an NGSI-LD Relationship or another NGSI-LD Property.
 - **NGSI-LD Relationship:** a directed link between a subject (starting point), that may be an NGSI-LD Entity, an NGSI-LD Property, or another NGSI-LD Relationship, and an object (end-point), that is an NGSI-LD Entity.
 - **NGSI-LD value:** a JSON value (i.e. a string, a number, true or false, an object, an array), or a JSON-LD typed value (i.e. a string as the lexical form of the value together with a type, defined by an XSD base type or more generally an IRI), or a JSON-LD structured value (i.e. a set, a list, or a language-tagged string).
 - **NGSI-LD type:** an OWL class that is a subclass of either the NGSI-LD Entity, NGSI-LD Relationship, NGSI-LD Property or NGSI-LD Value classes defined in the NGSI-LD meta-model. NGSI-LD pre-defines a small number of types, but is otherwise open to any types defined by users.


Example: Combined data exchange using Property Graphs



Example: Entity "Vehicle" and its @context in NGSI-LD

```
{
  "id": "urn:ngsi-ld:Vehicle:A4567",
  "type": "Vehicle",
  "brandName": {
    "type": "Property",
    "value": "Mercedes"
  },
  "inAccident": {
    "type": "Relationship",
    "object": "urn:ngsi-ld:SmartLamppost:Downtown1",
    "observedAt": "2019-05-29T12:14:55Z",
    "providedBy": {
      "type": "Relationship",
      "object": "urn:ngsi-ld:Org:Officer123"
    }
  },
}
```

```
@context": [
  "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld",
  "https://example.org/vehicle/my-user-terms-context.jsonld"
]
```



Concepts of NGSI-LD:

- NGSI-LD Entity
- NGSI-LD Property
- NGSI-LD Relationship
- NGSI-LD Value
- NGSI-LD Type

Work Description



Parts of Assignment 2

1. Part 1: Getting Started with NGSI-V2
2. Part 2: IOT Sensors
3. Part 3: IOT over MQTT
4. Part 4: Persisting And Querying Time Series Data (Cratedb) - TBD
5. Part 5: Visualizing NGSI Data Using A Mashup - TBD

Reference: Important information!

- Linux or VM machine necessary in principle (for Windows, WSL may work, although no guarantees)
 - If using VM but SSH'ing from native OS, you need to port forward
- Installing Docker:
 - Convenience script: <https://docs.docker.com/engine/install/ubuntu/#install-using-the-convenience-script>
 - Run docker without *sudo* (strongly recommended): <https://docs.docker.com/engine/install/linux-postinstall/>
- Installing Docker Compose:
 - There are two versions of Docker, v1 or v2
 - If using v1: when running the tutorial scripts, do: `./services create legacy; ./services start legacy`
 - If using v2
 - Make sure to uninstall Compose v1: <https://docs.docker.com/compose/install/#uninstallation>
 - Follow these instructions: <https://docs.docker.com/compose/cli-command/#install-on-linux>
- Troubleshooting:
 - Check if all containers are running properly: `docker ps` (check for 'unhealthy' status)
 - Most cases: `sudo docker system prune` (clean whatever is not being used – containers, networks,...)
 - Seldom: Network has active end-points
 - 1. `docker network inspect <network>` (lists endpoints on that network)
 - 2. `docker network disconnect -f <network> <endpoint>` (do this for all endpoints in the previous command :\)

Part 1: Getting Started with NGSI-V2



Getting Started with NGSI-V2

- “The demo application will only make use of one FIWARE component - the Orion Context Broker.
- Usage of the Orion Context Broker (with proper context data flowing through it) is sufficient for an application to qualify as “Powered by FIWARE”.
- Currently, the Orion Context Broker relies on open source MongoDB technology to keep persistence of the context data it holds. Therefore, the architecture will consist of two elements:
 - The Orion Context Broker which will receive requests using NGSI-v2
 - The underlying MongoDB database : Used by the Orion Context Broker to hold context data information such as data entities, subscriptions and registrations
- Since all interactions between the two services are initiated by HTTP requests, the services can be containerized and run from exposed ports.”



Getting Started With Ngsi-V2

- Tutorial

- <https://fiware-tutorials.readthedocs.io/en/latest/getting-started.html>

- Motivation

- “At its heart, FIWARE is a system for managing context information, so let's add some context data into the system by creating two new entities (stores in Berlin).
 - Any entity must have a id and type attributes, additional attributes are optional and will depend on the system being described.
 - Each additional attribute should also have a defined type and a value attribute.”

Steps

1. Creating Context Data
2. Querying Context Data
3. Obtain Entity Data By Id
4. Obtain Entity Data By Type
5. Filter Context Data By Comparing The Values Of An Attribute
6. Filter Context Data By Comparing The Values Of A Sub-Attribute
7. Filter Context Data By Querying Metadata
8. Filter Context Data By Comparing The Values Of A Geo:Json Attribute

Thank you

Systems of Systems (SoS)

M.Sc. In Critical Computing Systems Engineering

ISEP/IPP – 2021/22, 2nd semester

Assignment 2:

FIWARE

Pedro Santos

IoT and FIWARE



How do IoT and FIWARE combine?

- The Internet of Things (IoT) is a network of physical devices which are able to connect to a network and exchange data.
 - Each "thing" or "smart device" is a gadget with embedded electronics and software which can act as a sensor or actuator.
 - Sensors are able to report the state of the real-world around them.
 - Actuators are responsible for altering the state of the system, by responding to a control signal.
 - Each device is uniquely identifiable through its embedded computing system but is able to interoperate within the existing internet infrastructure.
- FIWARE is a system for managing context information.
 - For a smart solution based on the internet of Things, the context is provided by the array of attached IoT devices.
 - Since each IoT device is a physical object which exists in the real world, it will eventually be represented as a unique entity within the context.

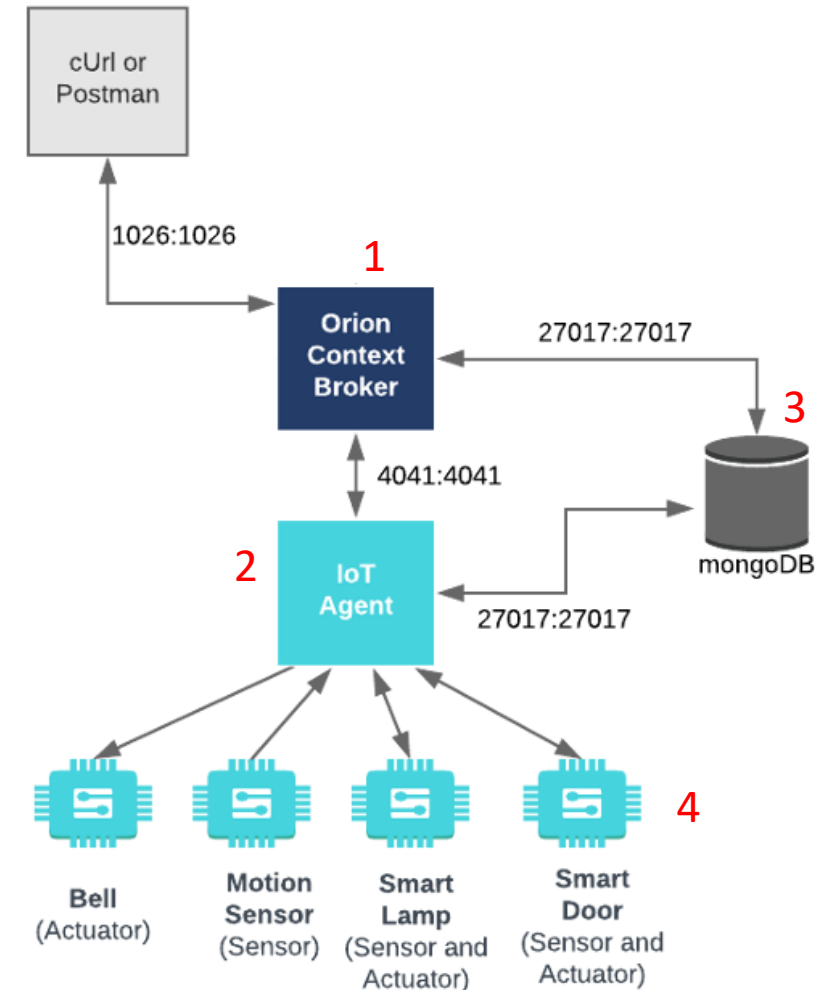
What are examples of IoT devices?

- IoT devices can range from simple to complex. Here are some examples of IoT devices which will be used within this tutorial:
 - A **Smart Door** is an electronic door which can be sent commands to be locked or unlocked remotely. It can also report on its current state (OPEN, CLOSED or LOCKED),
 - A **Bell** can be sent a command to activate and ring for a short period
 - A **Motion Sensor** can be queried to return the number of people who have passed by since it was last reset
 - A **Smart Lamp** can be switched on or off remotely. It can also report on its current state (ON or OFF). When switched on, a Motion Sensor within the device checks to see if light is needed and will dim if no-one is nearby. Furthermore the device can be report on the current luminosity of the bulb.
- As you can see, the Bell is an example of a pure actuator, as it only reacts to the given commands.
- Meanwhile, the Motion Sensor is an example of a pure sensor, since it will only report on the state of the world as it sees it.
- The other two devices are able to both respond to the commands and report on state in a meaningful way.

Reference Architecture for FIWARE and IoT

Components:

1. The FIWARE **Orion Context Broker** which will receive requests using NGSI-v2
2. The FIWARE **IoT Agent**
3. The **MongoDB database**:
 - Used by the Orion Context Broker to hold context data information such as data entities, subscriptions and registrations
 - Used by the IoT Agent to hold device information such as device URLs and Keys
4. A webserver acting as set of dummy **IoT devices** running a **proprietary protocol** running over HTTP.



Reference Architecture for FIWARE and IoT

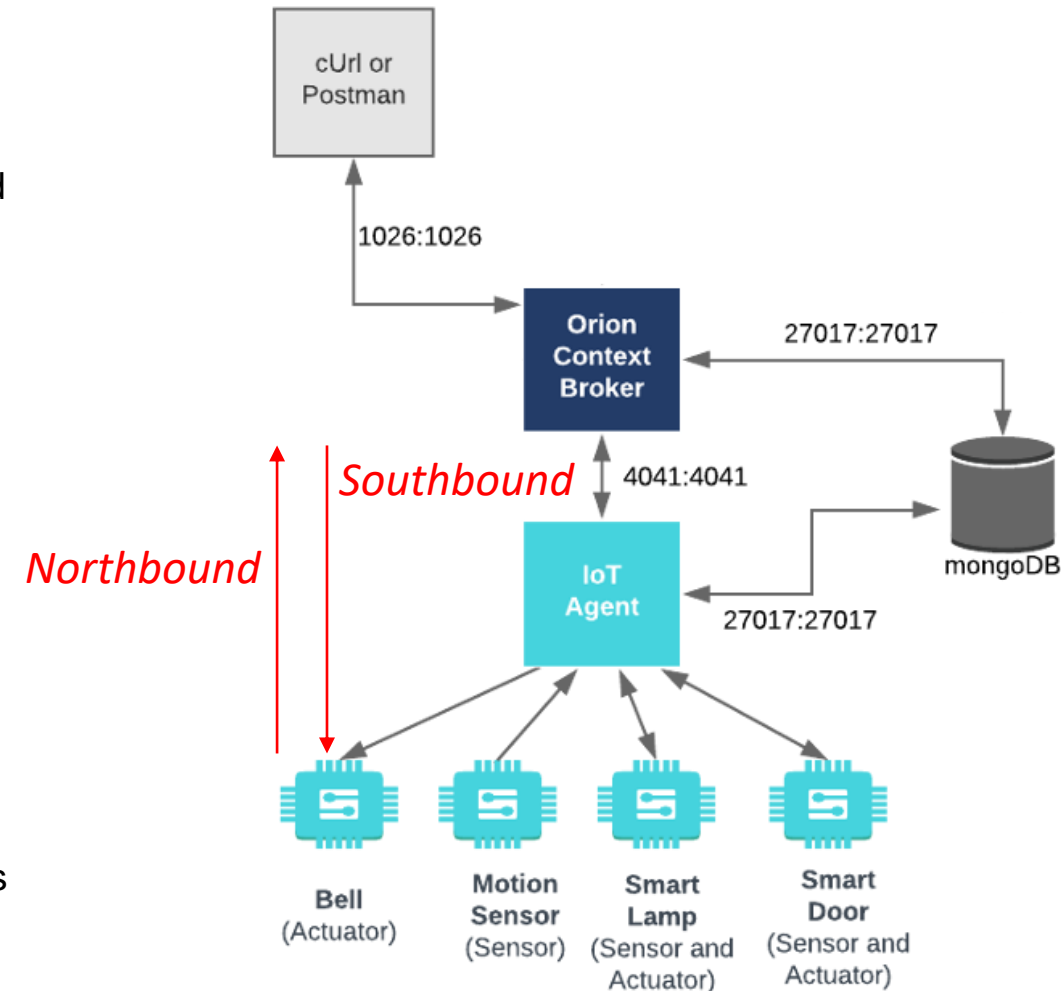
Traffic naming convention:

- **Southbound traffic**

- HTTP requests generated by the form the Context Broker and passed downwards towards an IoT device (via an IoT agent) are known as **southbound traffic**.
- Southbound traffic consists of commands made to actuator devices which alter the state of the real world by their actions

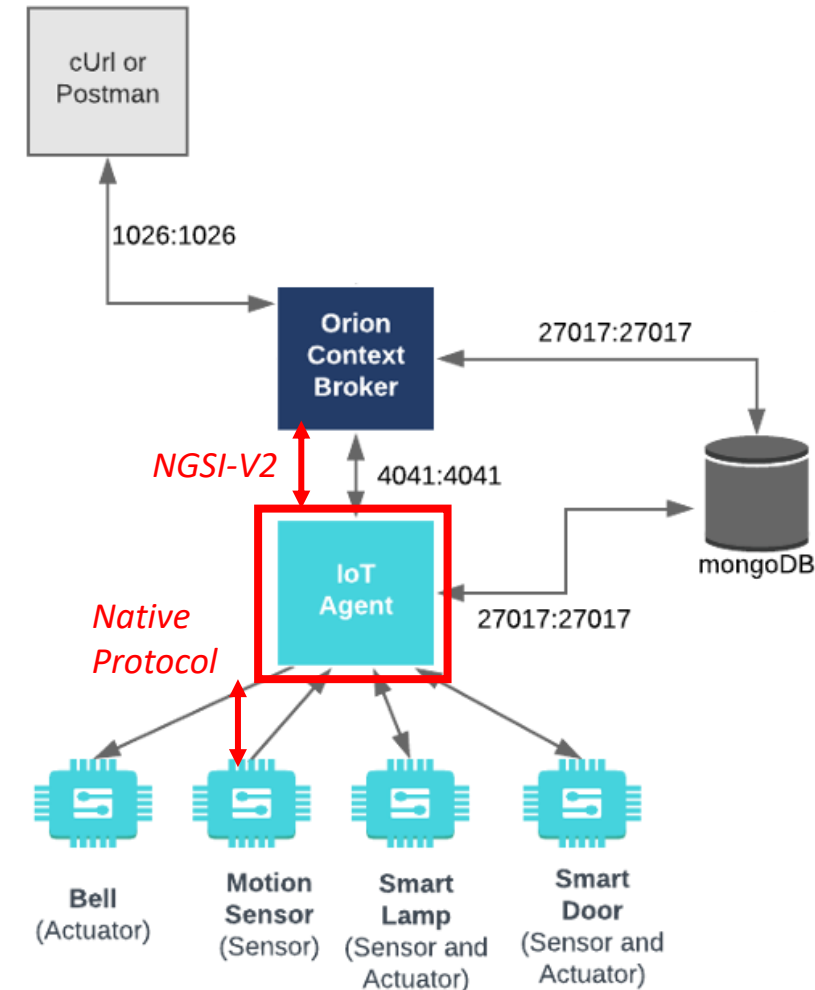
- **Northbound traffic**

- Requests generated from an IoT device and passed back upwards towards the Context Broker (via an IoT agent) are known as **northbound traffic**.
- Northbound traffic consists of measurements made by sensor devices and relays the state of the real world into the context data of the system.



Why do we need the IoT Agent?

- An IoT Agent is a component that lets a group of devices sends their data to and be managed from a Context Broker **using their own native protocols**.
- Examples of such native protocols include:
 - *IoTAgent-UL* - bridges HTTP/MQTT messaging (**UltraLight2.0** payload) and NGSI
 - *IoTAgent-JSON* - bridges HTTP/MQTT messaging (**JSON** payload) and NGSI
 - *IoTAgent-LWM2M* - bridges the **Lightweight M2M** protocol and NGSI
 - *IoTAgent-LoRaWAN* – bridges the **LoRaWAN** protocol and NGSI
- In turn, the Orion Context Broker uses **exclusively** NGSI-v2 requests.
- With this solution, **each group of IoT devices are able to use their own proprietary protocols and disparate transport mechanisms**.
- IoT Agents should also be able to deal with security aspects of the FIWARE platform (authentication and authorization of the channel) and provide other common services to the device programmer.



Part 2:

Querying/Controlling IoT Sensors using the Ultralight 2.0 Protocol

Ultralight 2.0

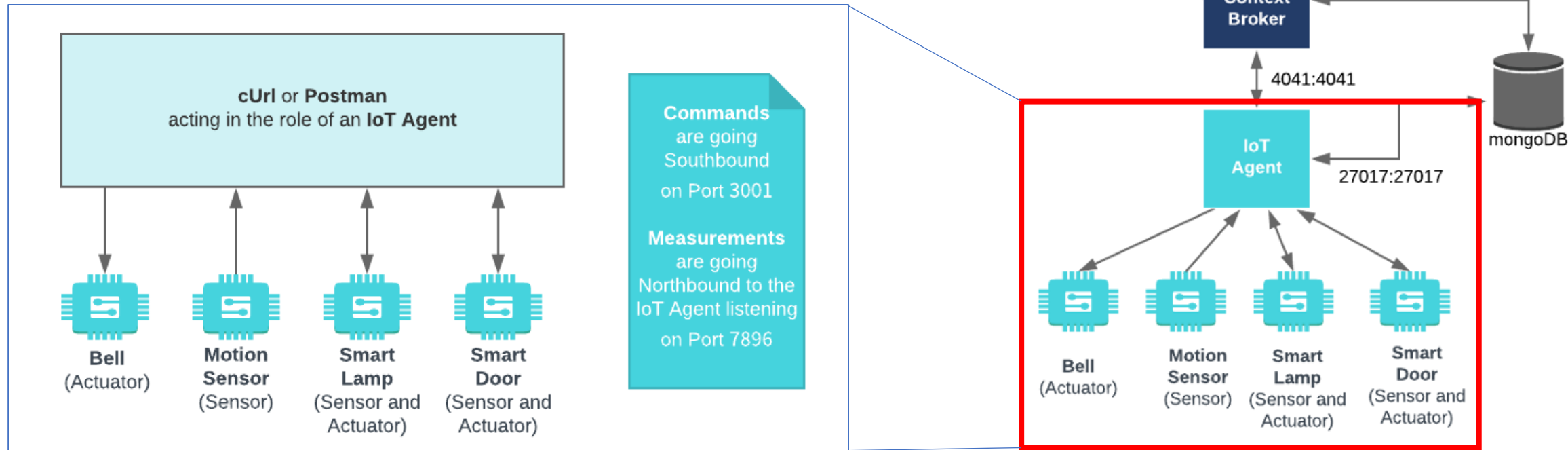
- UltraLight 2.0 is a lightweight text based protocol for constrained devices and communications where bandwidth and device memory resources are limited.
- It is one of the *native protocols* mentioned earlier.
- The payload for measurement requests is a list of key-value pairs separated by the pipe | character.

```
<key>|<value>|<key>|<value>|<key>|<value> etc..
```

```
t|15|k|abc
```

Architecture for this part

- In this tutorial we will not use the entire architecture; we will just look into the communication between the IoT agent (for Ultralight 2.0) and the sensors
- In practice, we will be controlling directly the sensors



Communicating With IoT Devices

- Tutorial
 - <https://fiware-tutorials.readthedocs.io/en/latest/iot-sensors.html>
- Motivation
 - “This tutorial is an introduction to IoT devices and the usage of the UltraLight 2.0 Protocol for constrained devices.
 - The tutorial introduces a series of dummy IoT devices which are displayed within the browser and allows a user to interact with them.
 - A complete understanding of all the terms and concepts defined in this tutorial is necessary before proceeding to connect the IoT devices to the Orion Context Broker via a real IoT Agent.”
- We will learn how to query and actuate over context data
 - Ring A Bell
 - Switch On A Smart Lamp
 - Unlock A Door
 - Etc.

Reference: Important information!

- Linux or VM machine necessary in principle (for Windows, WSL may work, although no guarantees)
 - If using VM but SSH'ing from native OS, you need to port forward
- Installing Docker:
 - Convenience script: <https://docs.docker.com/engine/install/ubuntu/#install-using-the-convenience-script>
 - Run docker without *sudo* (strongly recommended): <https://docs.docker.com/engine/install/linux-postinstall/>
 - <https://docs.docker.com/engine/security/rootless/>
- Installing Docker Compose:
 - There are two versions of Docker, v1 or v2
 - If using v1: when running the tutorial scripts, do: “`./services create legacy; ./services start legacy`”
 - If using v2
 - Make sure to uninstall Compose v1: <https://docs.docker.com/compose/install/#uninstallation>
 - Follow these instructions: <https://docs.docker.com/compose/cli-command/#install-on-linux>
- Troubleshooting:
 - Check if all containers are running properly: `docker ps` (check for ‘unhealthy’ status)
 - Most cases: `sudo docker system prune` (clean whatever is not being used – containers, networks,...)
 - Seldom: Network has active end-points
 - 1. `docker network inspect <network>` (lists endpoints on that network)
 - 2. `docker network disconnect -f <network> <endpoint>` (do this for all endpoints in the previous command :\)

Example: Ring a bell

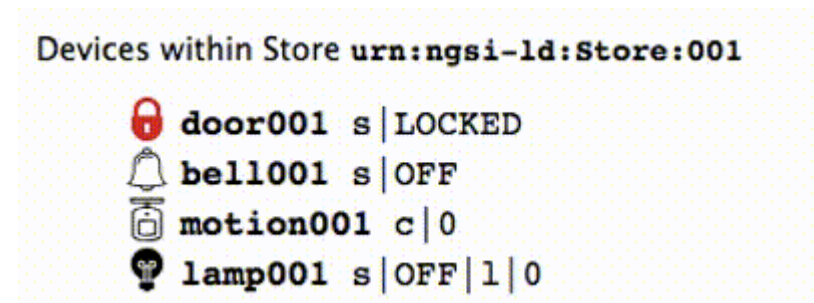
1 REQUEST:

```
curl -iX POST \  
  --url 'http://localhost:3001/iot/bell001' \  
  --data urn:ngsi-ld:Bell:001@ring
```

RESPONSE:

```
urn:ngsi-ld:Bell:001@ring| ring OK
```

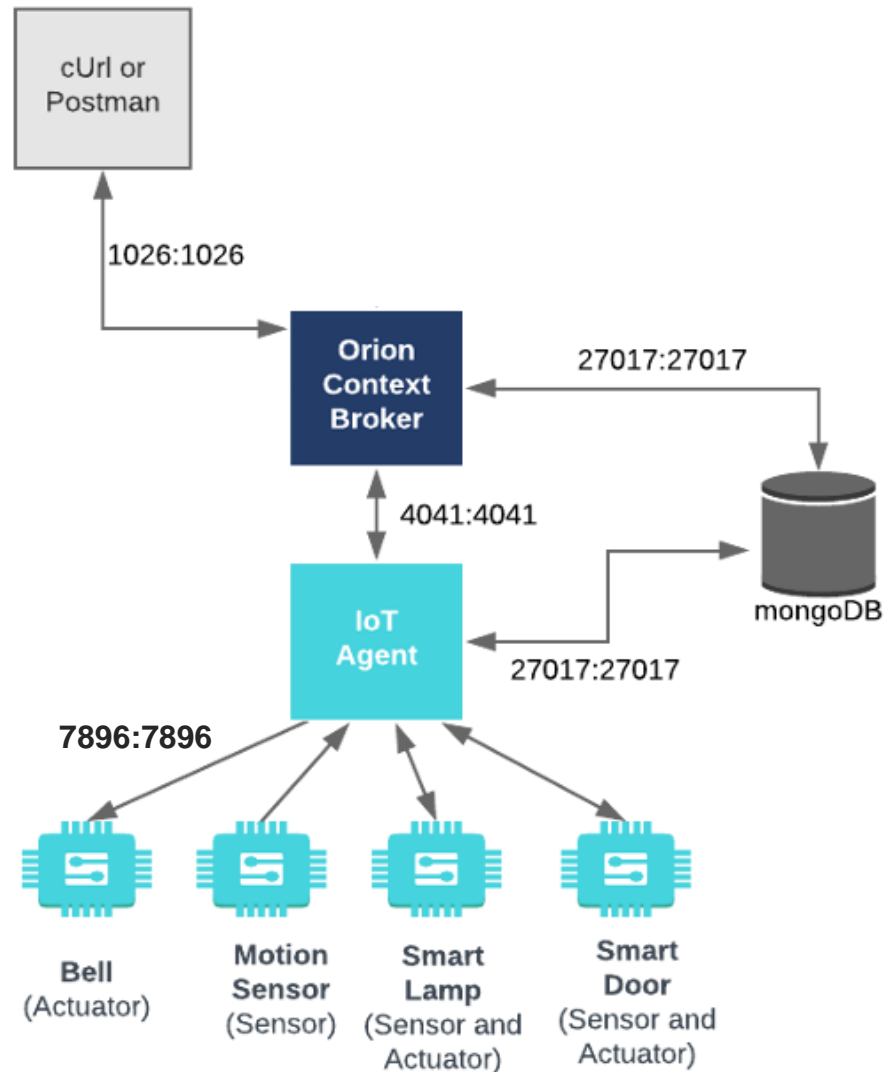
- What did you observe in the UltraLight device monitor web page?
 - In <http://localhost:3000/device/monitor>



Part 3:

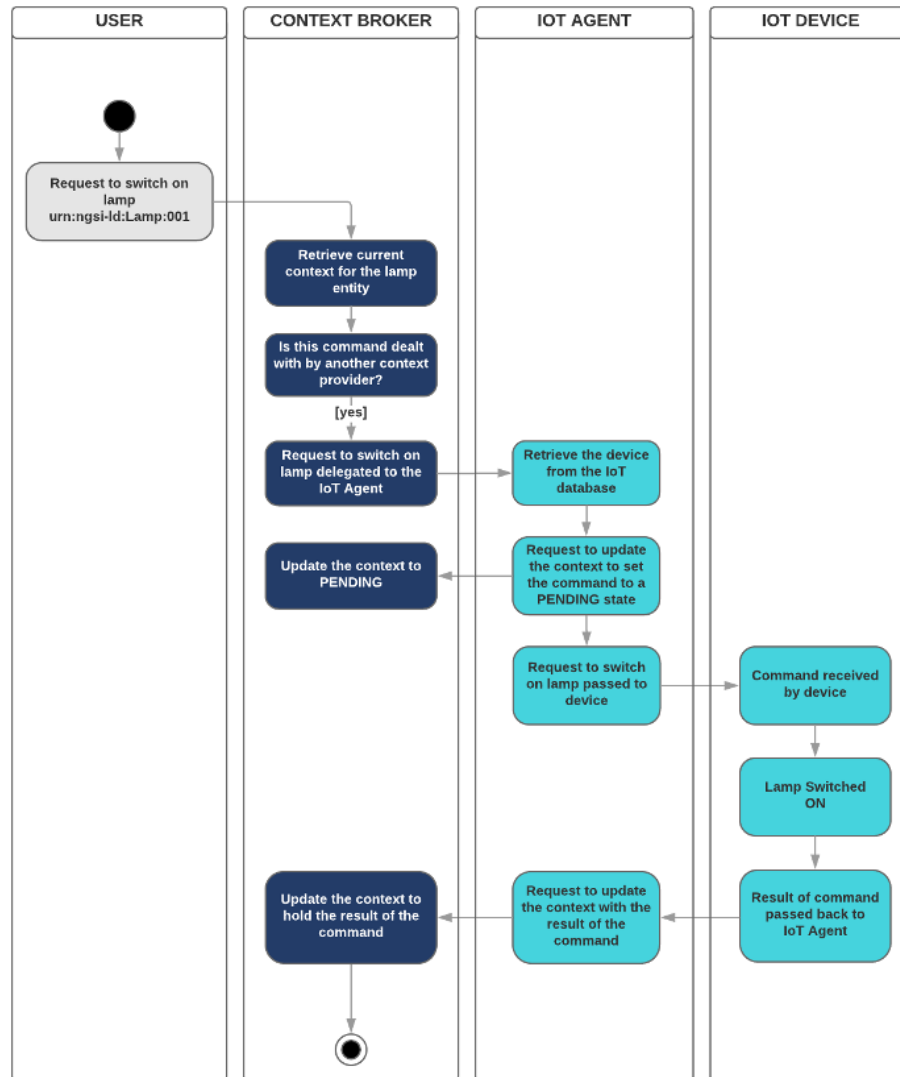
IOT Agent (Ultralight 2.0)

Reference Architecture for this Part

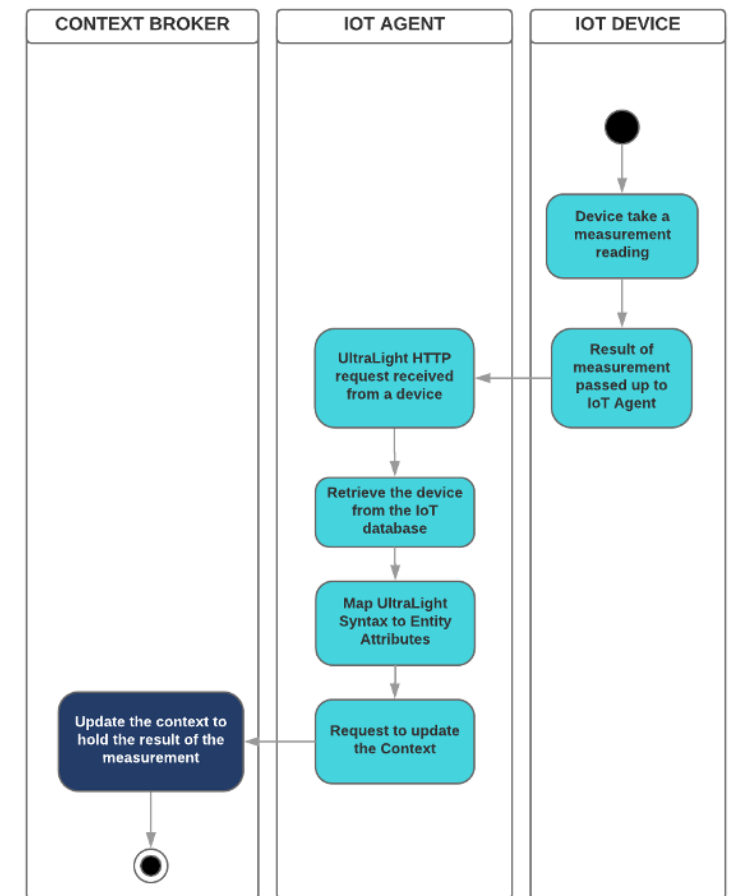


Workflows

Southbound



Northbound



IOT Agent (Ultralight 2.0)

- Tutorial
 - <https://fiware-tutorials.readthedocs.io/en/latest/iot-agent.html>
 - BEWARE! It may be necessary to re-run Part 1 to re-register the Store contexts.
 - Carry out until section “Switching On The Smart Lamp” - Step 14

SWITCHING ON THE SMART LAMP

To switch on the **Smart Lamp**, the `on` attribute must be updated in the context.

14 REQUEST:

- Motivation
 - “This tutorial introduces the concept of an IoT Agent and wires up the dummy UltraLight 2.0 IoT devices created in the previous tutorial so that measurements can be read and commands can be sent using NGSI-v2 requests sent to the Orion Context Broker.”

Connecting IoT Devices

- The IoT Agent acts as a middleware between the IoT devices and the context broker. **It therefore needs to be able to create context data entities with unique IDs.**
- Once a service has been provisioned and an unknown device makes a measurement, the IoT Agent add this to the context using the supplied **<device-id>** (unless the device is recognized and can be mapped to a known ID).
- There is no guarantee that every supplied IoT device <device-id> will always be unique, therefore all provisioning requests to the IoT Agent require two mandatory headers:
 - *fiware-service* header is defined so that entities for a given service can be held in a separate mongoDB database.
 - *fiware-servicepath* can be used to differentiate between arrays of devices.
- Example:
 - Within a smart city application you would expect different *fiware-service* headers for different departments (e.g. parks, transport, refuse collection etc.) and each *fiware-servicepath* would refer to specific park and so on.
 - This would mean that data and devices for each service can be identified and separated as needed, but the data would not be siloed - for example data from a Smart Bin within a park can be combined with the GPS Unit of a refuse truck to alter the route of the truck in an efficient manner.

Provisioning A Service Group

- Invoking group provision is always the first command to send to the IoT agent in order to connect devices.
 - It is always necessary to supply an authentication key with each measurement, and
 - The IoT Agent will not initially know which URL the context broker is responding on.
- This example provisions an anonymous group of devices. It tells the IoT Agent that a series of devices will be sending messages to the port where the IoT Agent is listening for Northbound communications (7896)
- In the example the IoT Agent is informed that the `/iot/d` endpoint will be used, and that devices will authenticate themselves by including the token `4jggokgpepnvsb2uv4s40d59ov`.
- For an UltraLight IoT Agent this means devices will be sending GET or POST requests to:

`http://iot-agent:7896/iot/d?i=<device_id>&k=4jggokgpepnvsb2uv4s40d59ov`

Note! We're communicating directly to the IoT Agent!
(Check Reference Architecture a few slides back)

2 REQUEST:

```
curl -iX POST \  
  'http://localhost:4041/iot/services' \  
  -H 'Content-Type: application/json' \  
  -H 'fiware-service: openiot' \  
  -H 'fiware-servicepath: /' \  
  -d '{  
    "services": [  
      {  
        "apikey": "4jggokgpepnvsb2uv4s40d59ov",  
        "cbroker": "http://orion:1026",  
        "entity_type": "Thing",  
        "resource": "/iot/d"  
      }  
    ]  
  }'
```

Provisioning A Motion Sensor

3 REQUEST:

- This device will be associated to one of the Store contexts we created in Part 1 of this work.
- Three types of measurement attributes can be provisioned:
 - **attributes** - are active readings from the device
 - **static_attributes** - are as the name suggests static data about the device (such as relationships) passed on to the context broker.

IMPORTANT! Associating this sensor to the URN of the Store 1 context we created in Part 1!

```
curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
    "devices": [
      {
        "device_id": "motion001",
        "entity_name": "urn:ngsi-ld:Motion:001",
        "entity_type": "Motion",
        "timezone": "Europe/Berlin",
        "attributes": [
          { "object_id": "c", "name": "count", "type": "Integer" }
        ],
        "static_attributes": [
          { "name": "refStore", "type": "Relationship", "value": "urn:ngsi-ld:Store:001" }
        ]
      }
    ]
  },
```

device_id
Name of sensor
Type of sensor
Attribute (only 1)

Checking Association

- We now **verify the association between the Motion sensor device** (and its measurements) and **the corresponding entity stored in the Context Broker**.
- Let's start by simulating a dummy IoT device measurement coming from the Motion Sensor device `motion001`, by making the following request
- Now, let's retrieve the entity data from the context broker.
- The response shows that the Motion Sensor device with `id=motion001` has been successfully identified by the IoT Agent and mapped to the NGSI-v2 entity `id=urn:ngsi-ld:Motion:001`.
- This new entity has been created within the context data.
- The `c` attribute from the dummy device measurement request has been mapped to the more meaningful `count` attribute within the context.

4 REQUEST:

Simulating a sensor measurement to the IoT agent

```
curl -iX POST \
  'http://localhost:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=motion001' \
  -H 'Content-Type: text/plain' \
  -d 'c|1'
```

5 REQUEST:

Contacting the Orion Broker

```
curl -G -X GET \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Motion:001' \
  -d 'type=Motion' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

RESPONSE:

```
{
  "id": "urn:ngsi-ld:Motion:001", "type": "Motion",
  "timeInstant": {
    "type": "ISO8601", "value": "2018-05-25T10:51:32.00Z",
    "metadata": {}
  },
  "count": {
    "type": "Integer", "value": "1",
    "metadata": {
      "timeInstant": { "type": "ISO8601", "value": "2018-05-25T10:51:32.646Z" }
    }
  },
  "refStore": {
    "type": "Relationship",
    "value": "urn:ngsi-ld:Store:001",
    "metadata": {
      "timeInstant": { "type": "ISO8601", "value": "2018-05-25T10:51:32.646Z" }
    }
  }
}
```

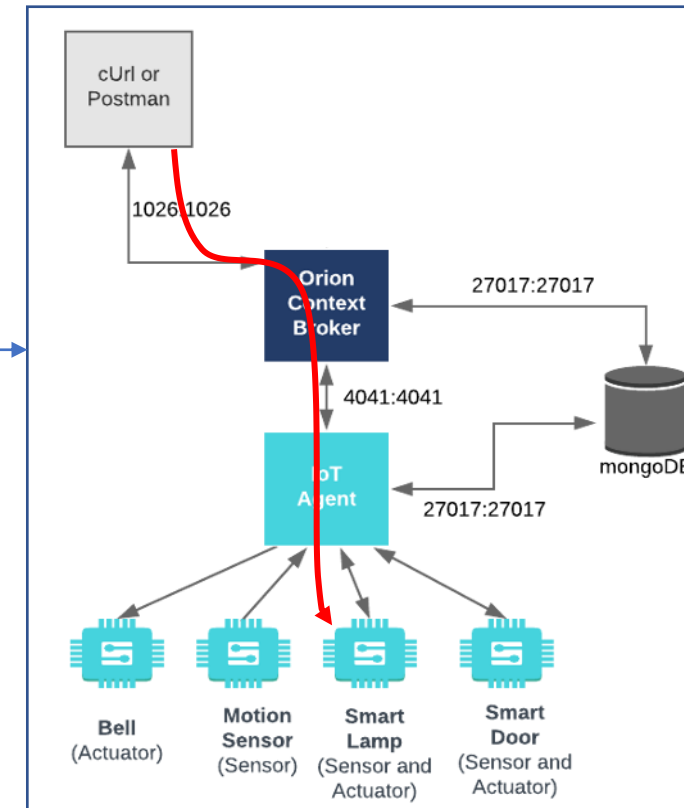
Commands from Context Broker

- Ringing the bell

13 REQUEST:

```
curl -iX PATCH \
  'http://localhost:1026/v2/entities/urn:ngsi-id:bell:001/attrs' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
    "ring": {
      "type": "command",
      "value": ""
    }
  }'
```

1. Note! Now we're communicating to the Orion Broker! (Check Reference Architecture)



2. Furthermore, we identify the device we wish to access by its NGSI-v2 id

- What did you observe in the UltraLight device monitor web page?
 - In <http://localhost:3000/device/monitor>

Reminder

- Carry out until section “Switching On The Smart Lamp” - Step 14

SWITCHING ON THE SMART LAMP

To switch on the **Smart Lamp**, the `on` attribute must be updated in the context.

14 REQUEST:

Other things to try (not mandatory)

- An IoT Agent for a different protocol - JSON IoT Agent
 - <https://fiware-tutorials.readthedocs.io/en/latest/iot-agent-json.html>
- IoT over MQTT
 - <https://fiware-tutorials.readthedocs.io/en/latest/iot-over-mqtt.html>

Thank you

Systems of Systems (SoS)

M.Sc. In Critical Computing Systems Engineering

ISEP/IPP – 2021/22, 2nd semester

Assignment 2:

FIWARE

Pedro Santos

Part 4

Persisting And Querying Time Series Data

Revisiting Parts 1, 2 & 3

- Part 1: Creating Context
 - We created two entities – two stores in Berlin, with attributes:
 - ID
 - Name
 - Address
 - Coordinates
- Part 2: Querying/Controlling IoT Sensors
 - We provisioned a few sensors:
 - Bell
 - Motion Detector
 - Smart Lamp
 - Door Lock
 - We associated them to the two entities defined in Part 1
 - We performed a number of queries to read or activate the sensors/actuators





```
curl -iX POST \
'http://localhost:1026/v2/entities' \
-H 'Content-Type: application/json' \
-d '
{
  "id": "urn:ngsi-ld:Store:001",
  "type": "Store",
  "address": {
    "type": "PostalAddress",
    "value": {
      "streetAddress": "Bornholmer Straße 65",
      "addressRegion": "Berlin",
      "addressLocality": "Prenzlauer Berg",
      "postalCode": "10439"
    }
  },
  "metadata": {
    "verified": {
      "value": true,
      "type": "Boolean"
    }
  }
},
"location": {
  "type": "geo:json",
  "value": {
    "type": "Point",
    "coordinates": [13.3986, 52.5547]
  }
},
"name": {
  "type": "Text",
  "value": "Bösebrücke Einkauf"
}
}'

curl -iX POST \
'http://localhost:1026/v2/entities' \
-H 'Content-Type: application/json' \
-d '
{
  "type": "Store",
  "id": "urn:ngsi-ld:Store:002",
  "address": {
    "type": "PostalAddress",
    "value": {
      "streetAddress": "Friedrichstraße 44",
      "addressRegion": "Berlin",
      "addressLocality": "Kreuzberg",
      "postalCode": "10969"
    }
  },
  "metadata": {
    "verified": {
      "value": true,
      "type": "Boolean"
    }
  }
},
"location": {
  "type": "geo:json",
  "value": {
    "type": "Point",
    "coordinates": [13.3903, 52.5075]
  }
},
"name": {
  "type": "Text",
  "value": "Checkpoint Markt"
}
}'
```







UltraLight IoT Devices

Devices within Store **urn:ngsi-ld:Store:001**

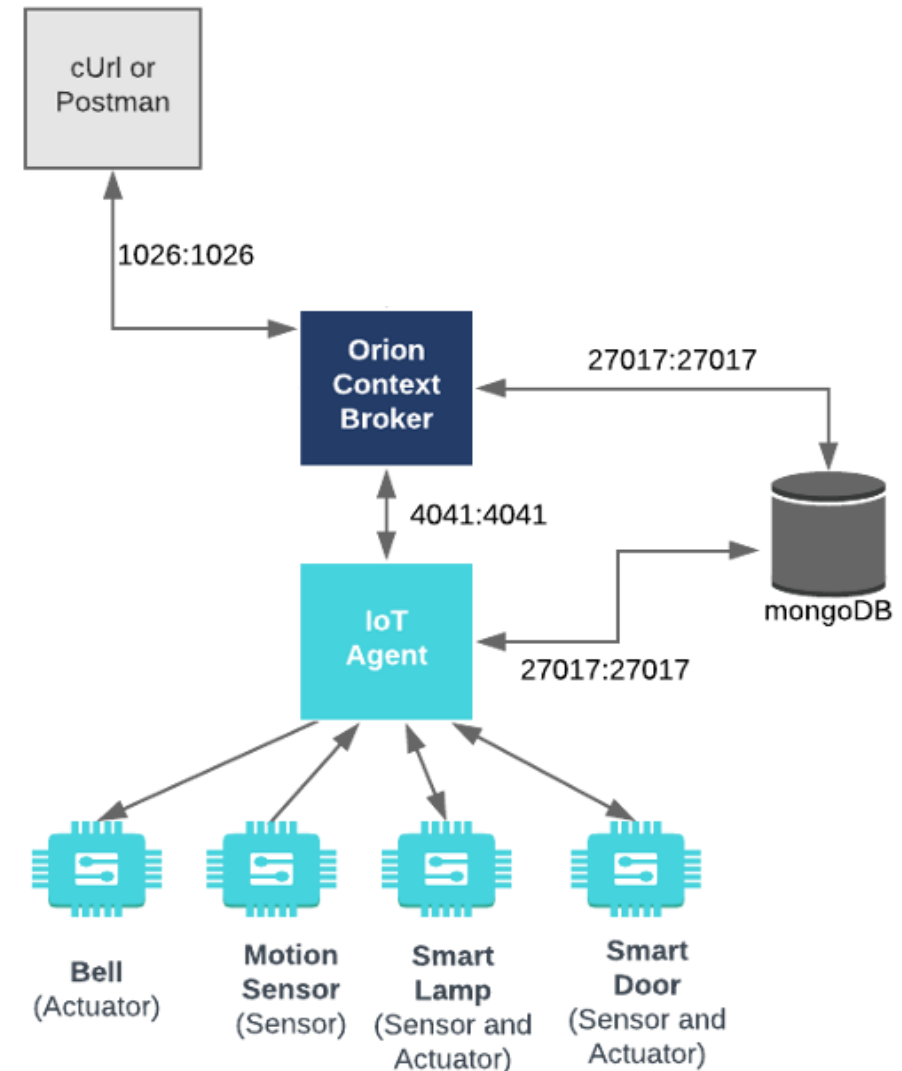
 **door001** s | CLOSED
 **bell001** s | OFF
 **motion001** c | 1589
 **lamp001** s | ON, 1 | 1900

Devices within Store **urn:ngsi-ld:Store:003**

 **door003** s | LOCKED
 **bell003** s | OFF
 **motion003** c | 0
 **lamp003** s | OFF, 1 | 0

Revisiting Part 3 & Motivating Part 4

- Part 3: IOT Agent
 - We provisioned a number of IoT devices through a IoT agent (Ultralight 2.0)
 - The IoT Agent abstracts the FIWARE infrastructure from whichever protocols the sensors/actuators are using.
- Goals for Part 4:
 - How do we store data persistently?
 - How to create a dashboard to visualize data?



How to store data persistently & display it?

- Motivation

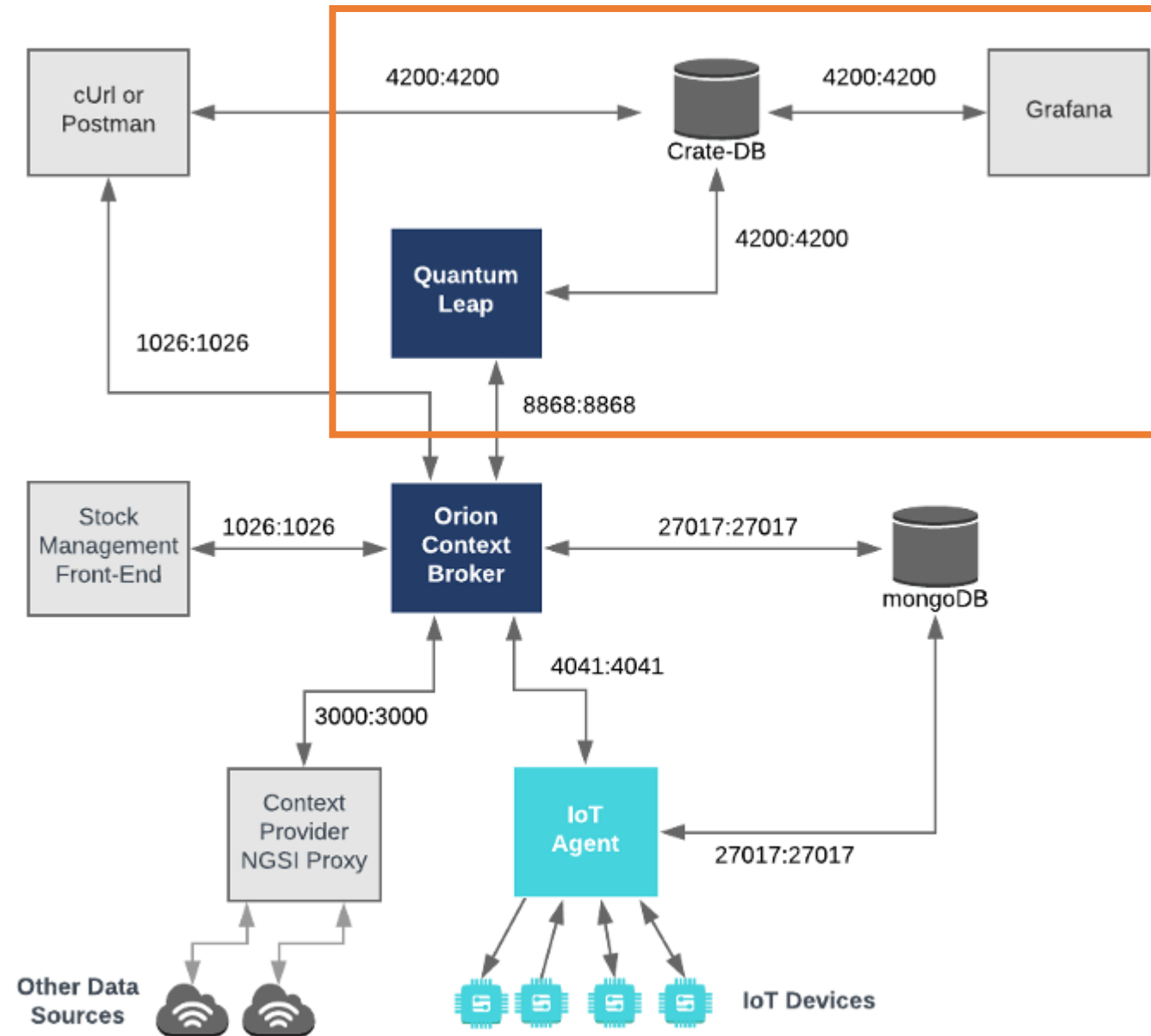
- The Orion Broker does not store data persistently
- In general, brokers in publish-subscriber do not store data
- A database is required to this end.

- FIWARE Tool Ecosystem for Time-series Storage

- **FIWARE QuantumLeap**: Generic enabler that offers an API to persist and query time-series database
- **CrateDB**: distributed SQL Database Management System (DBMS)
 - Designed for use with the internet of Things.
 - It is capable of ingesting a large number of data points per second and can be queried in real-time.
 - The database is designed for the execution of complex queries such as geospatial and time series data.
 - Retrieval of this historic context data allows for the creation of graphs and dashboards displaying trends over time.
- **Grafana**: dashboard tool

Architecture

- **FIWARE QuantumLeap**: Generic enabler that offers an API to persist and query time-series database (CrateDB)
- **CrateDB**: distributed SQL Database Management System (DBMS)
- **Grafana**: dashboard tool



Tutorial: “Persisting And Querying Time Series Data”

- Base tutorial: <https://github.com/FIWARE/tutorials.Time-Series-Data>
 - However, **I recommend you follow these slides**: there are some errors in the tutorial and the slides offer a more-to-the-point structure. Consult the link above for clarification on any missing element.
- Motivation
 - FIWARE QuantumLeap is a generic enabler which is used to persist context data into a CrateDB database.
 - In this tutorial, we will activate two IoT sensors – Smart Lamp and Motion Detector – and take measurements that will be stored persistently – i.e., in the database.
 - To retrieve time-based aggregations of such data, users can either use QuantumLeap query API or connect directly to the CrateDB HTTP endpoint.
 - Results are visualised on a graph or via the Grafana time series analytics tool.
- We will learn:
 1. Connecting Fiware to A CrateDB Database via QuantumLeap
 2. Perform time series data queries
 3. Displaying CrateDB data as a Grafana Dashboard

Installing Docker

- Linux or VM machine necessary in principle (for Windows, WSL may work, although no guarantees)
 - If using VM but SSH'ing from native OS, you need to port forward
- Installing Docker:
 - Convenience script: <https://docs.docker.com/engine/install/ubuntu/#install-using-the-convenience-script>
 - Run docker without *sudo* (strongly recommended): <https://docs.docker.com/engine/install/linux-postinstall/>
 - <https://docs.docker.com/engine/security/rootless/>
- Installing Docker Compose:
 - There are two versions of Docker, v1 or v2
 - If using v1: when running the tutorial scripts, do: “./services create **legacy**; ./services start **legacy**”
 - If using v2
 - Make sure to uninstall Compose v1: <https://docs.docker.com/compose/install/#uninstallation>
 - Follow these instructions: <https://docs.docker.com/compose/cli-command/#install-on-linux>

1. Starting Tutorial

1. First steps:

```
git clone https://github.com/FIWARE/tutorials.Time-Series-Data.git
cd tutorials.Time-Series-Data
git checkout NGSI-v2
sudo sysctl -w vm.max_map_count=262144

./services create
```

```
./services start
```

2. After a while, run “**docker ps**”.

Make sure you see **8** containers running with no errors (‘unhealthy’ status).

- If this happens, check the ‘troubleshooting’ section in the next section.

```
p@p-box:~$ docker ps
CONTAINER ID   IMAGE
6d8e68eca9dc   fiware/tutorials.context-provider
373b7ee74b54   grafana/grafana:6.1.6
cdf5507bf2e3   orchestracities/quantumleap:0.8.3
25947298824d   fiware/iotagent-ul:1.20.0-distroless
62477b816523   fiware/orion:3.5.1
5c7a3f238db4   redis:6
93520d340830   mongo:4.4
2af3673ba745   crate:4.1.4
```


Troubleshooting Tutorial Start





- Starting all FIWARE services: `./services create; ./services start`
- Possible errors & solutions:
 - ***“Error response from daemon: network 8be72451a6fc749d863d7c2a2e8700fcbc457491071d086bec5fc409f3a84faa not found”***
 1. Run **“docker system prune”**
 2. Restart
 - **Not all containers start, or are in unhealthy state**
 1. Stop (see below) and restart
- Stopping all FIWARE services: `./services stop`
- Possible errors & solutions:
 - ***“Error response from daemon: error while removing network: network fiware_default id e3fc126ec172a85d21f280bfd3b679e51f26eb031516234e3c3c0b9897823322 has active endpoints”***
 1. Run **“docker ps”** to get a list of active dockers
 2. Perform **“docker stop <docker ID1> <docker ID2> ... <docker IDx>”** for all active dockers
 3. Repeat **“./services stop”**

2. Create Sensor/Actuator Data





1. Go to <http://localhost:3000/device/monitor>
2. Unlock a Smart Door and switch on a Smart Lamp a few times
 - In this manner we will producing some data for the time-series DB.



Devices within Store `urn:ngsi-ld:Store:001`

 **door001** s | LOCKED
 **bell001** s | OFF
 **motion001** c | 4
 **lamp001** s | ON | 1 | 1000

Devices within Store `urn:ngsi-ld:Store:003`

 **door003** s | LOCKED
 **bell003** s | OFF
 **motion003** c | 0
 **lamp003** s | OFF | 1 | 0

Northbound Traffic

- 2:09:09 PM - `/iot/d?i=lamp001&k=1234&d=s|ON||1400`
- 2:09:12 PM - `/iot/d?i=lamp001&k=1234&d=s|ON||1300`
- 2:09:15 PM - `/iot/d?i=lamp001&k=1234&d=s|ON||1200`
- 2:09:18 PM - `/iot/d?i=lamp001&k=1234&d=s|ON||1100`
- 2:09:21 PM - `/iot/d?i=lamp001&k=1234&d=s|ON||1000`

2. Create subscriptions to Sensor/Actuator Data

- Lamp & Motion Sensor

(1 Request in base tutorial – see footer)

```
curl -iX POST \
  'http://localhost:1026/v2/subscriptions/' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
    "description": "Notify QuantumLeap on luminosity changes on any Lamp",
    "subject": {
      "entities": [
        {
          "idPattern": "Lamp.*"
        }
      ],
      "condition": {
        "attrs": [
          "luminosity",
          "location"
        ]
      }
    },
    "notification": {
      "http": {
        "url": "http://quantumleap:8668/v2/notify"
      },
      "attrs": [
        "luminosity", "location"
      ],
      "metadata": ["dateCreated", "dateModified"]
    },
    "throttling": 1
  }'
```

(2 Request)

```
curl -iX POST \
  'http://localhost:1026/v2/subscriptions/' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
    "description": "Notify QuantumLeap of count changes of any Motion Sensor",
    "subject": {
      "entities": [
        {
          "idPattern": "Motion.*"
        }
      ],
      "condition": {
        "attrs": [
          "count"
        ]
      }
    },
    "notification": {
      "http": {
        "url": "http://quantumleap:8668/v2/notify"
      },
      "attrs": [
        "count"
      ],
      "metadata": ["dateCreated", "dateModified"]
    },
    "throttling": 1
  }'
```

3. Querying Time Series Data

1. QuantumLeap API - List the first N Sampled Values (4 Request)

```
curl -X GET \
  'http://localhost:8668/v2/entities/Lamp:001/attrs/luminosity?limit=3' \
  -H 'Accept: application/json' \
  -H 'Fiware-Service: openiot' \
  -H 'Fiware-ServicePath: /'
```

2. QuantumLeap API - List N Sampled Values at an Offset (5 Request)

```
curl -X GET \
  'http://localhost:8668/v2/entities/Motion:001/attrs/count?offset=3&limit=3' \
  -H 'Accept: application/json' \
  -H 'Fiware-Service: openiot' \
  -H 'Fiware-ServicePath: /'
```

3. QuantumLeap API - List the latest N Sampled Values (6 Request)

```
curl -X GET \
  'http://localhost:8668/v2/entities/Motion:001/attrs/count?lastN=3' \
  -H 'Accept: application/json' \
  -H 'Fiware-Service: openiot' \
  -H 'Fiware-ServicePath: /'
```

3. Querying Time Series Data

4. CrateDB API - List the first N Sampled Values (14 Request)

```
curl -iX POST \  
  'http://localhost:4200/_sql' \  
  -H 'Content-Type: application/json' \  
  -d '{"stmt":"SELECT * FROM mtopeniot.etlamp WHERE entity_id = '\''Lamp:001'\'' ORDER BY time_index ASC LIMIT 3"}'
```

5. CrateDB API - List N Sampled Values at an Offset (15 Request)

```
curl -iX POST \  
  'http://localhost:4200/_sql' \  
  -H 'Content-Type: application/json' \  
  -d '{"stmt":"SELECT * FROM mtopeniot.etmotion WHERE entity_id = '\''Motion:001'\'' order by time_index ASC LIMIT 3 OFFSET 3"}'
```

6. CrateDB API - List the latest N Sampled Values (16 Request)

```
curl -X GET \  
  'http://localhost:8668/v2/entities/Motion:001/attrs/count?lastN=3' \  
  -H 'Accept: application/json' \  
  -H 'Fiware-Service: openiot' \  
  -H 'Fiware-ServicePath: /'
```

3. Querying Time Series Data

7. QuantumLeap API - List the Sum of values grouped by a time period (7 Request)

```
curl -X GET \  
  'http://localhost:8668/v2/entities/Motion:001/attrs/count?aggrMethod=count&aggrPeriod=minute&lastN=3' \  
  -H 'Accept: application/json' \  
  -H 'Fiware-Service: openiot' \  
  -H 'Fiware-ServicePath: /'
```

8. QuantumLeap API - List the Minimum Values grouped by a Time Period (8 Request)

```
curl -X GET \  
  'http://localhost:8668/v2/entities/Lamp:001/attrs/luminosity?aggrMethod=min&aggrPeriod=minute&lastN=3' \  
  -H 'Accept: application/json' \  
  -H 'Fiware-Service: openiot' \  
  -H 'Fiware-ServicePath: /'
```

9. QuantumLeap API - List the Maximum Value over a Time Period (9 Request)

```
curl -X GET \  
  'http://localhost:8668/v2/entities/Lamp:001/attrs/luminosity?aggrMethod=max&fromDate=2018-06-27T09:00:00&toDate=2018-06-30T23:59:59' \  
  -H 'Accept: application/json' \  
  -H 'Fiware-Service: openiot' \  
  -H 'Fiware-ServicePath: /'
```

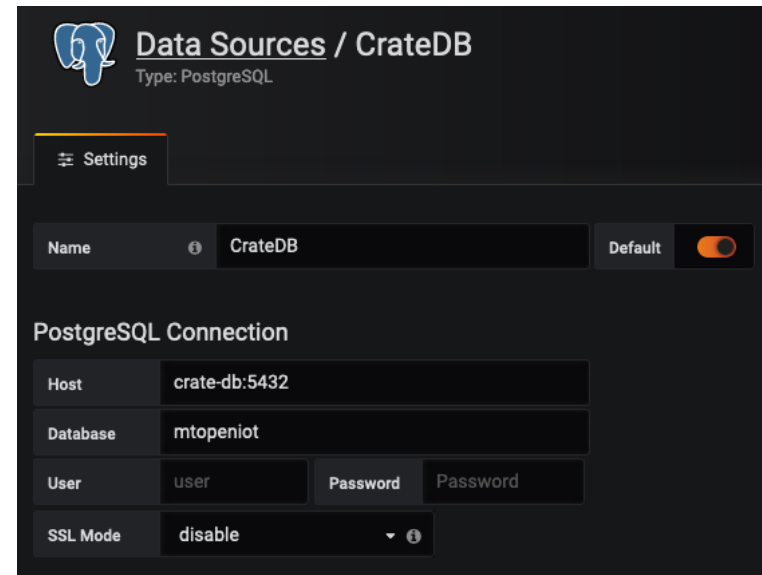
3. Querying Time Series Data Near A Point

10. QuantumLeap API - List The Latest N Sampled Values Of Devices Near A Point

```
curl -X GET \  
  'http://localhost:8668/v2/types/Lamp/attrs/luminosity?lastN=4&georel=near;maxDistance:5000&geometry=point&coords=13.3986,52.5547' \  
  -H 'Accept: application/json' \  
  -H 'Fiware-Service: openiot' \  
  -H 'Fiware-ServicePath: /'
```

4. Displaying CrateDB data as a Grafana Dashboard

- CrateDB integrates seamlessly with the Grafana time series analytics tool, that can be used to display aggregated sensor data.
- The instructions below summarize how to connect and display a graph of the Lamp luminosity data:
 1. Log in: <http://localhost:3003/login>. The default username is admin and the default password is admin.
 2. After logging in, a PostgreSQL datasource must be set up.
Go to: <http://localhost:3003/datasources>
 3. Fill in the following values
 - Name CrateDB
 - Host crate-db:5432
 - Database mtopeniot
 - User crate
 - SSL Mode disable

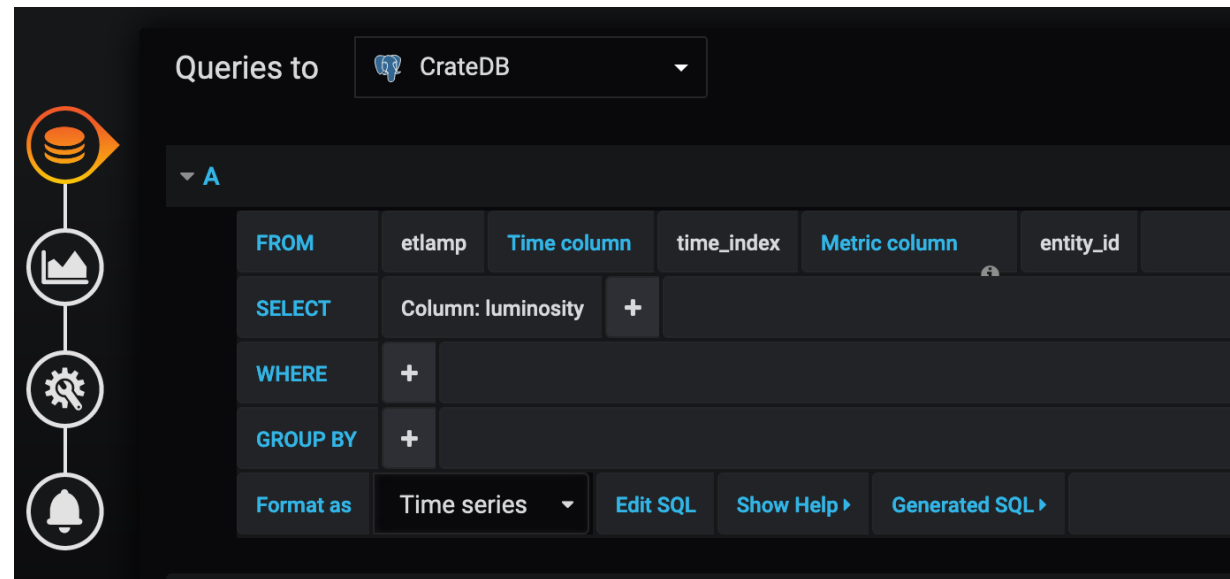


The screenshot shows the Grafana 'Data Sources' configuration page for a PostgreSQL connection. The title is 'Data Sources / CrateDB' with a sub-label 'Type: PostgreSQL'. A 'Settings' tab is selected. The 'Name' field is 'CrateDB' and the 'Default' toggle is turned on. Under the 'PostgreSQL Connection' section, the 'Host' is 'crate-db:5432', the 'Database' is 'mtopeniot', the 'User' is 'user', the 'Password' field is empty, and the 'SSL Mode' is set to 'disable'.

Data Sources / CrateDB	
Type: PostgreSQL	
Settings	
Name	CrateDB
Default	<input checked="" type="checkbox"/>
PostgreSQL Connection	
Host	crate-db:5432
Database	mtopeniot
User	user
Password	Password
SSL Mode	disable

4. Displaying CrateDB data as a Grafana Dashboard

4. To display a new dashboard, you can either click the + button and select Dashboard or go directly to <http://localhost:3003/dashboard/new?orgId=1>.
5. Thereafter click Add Query.
6. The following values in bold text need to be placed in the graphing wizard
 - Queries to CrateDB (the previously created Data Source)
 - FROM etlamp
 - Time column **time_index**
 - Metric column **entity_id**
 - Select value column: **luminosity**



Thank You