This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

PortForward

D3.1 – Cloud Platform Integration concept and guidelines

V. Bracke (IMEC), J. Hoebeke (IMEC), D. Kerkhove (IMEC), Tobias Kutzler (IFF) and B. Volckaert (IMEC)

Document Number	D3.1
Document Title	Cloud Platform Integration concept and guidelines
Version	1.0
Status	Final
Deliverable Type	Report
Contractual Date of Delivery	30.04.2019
Actual Date of Delivery	30.04.2019
Contributors	IMEC, IFF
Keyword List	Cloud Platform, IoT Stack, Middleware, Integration
Dissemination level	Public

Change History

Version	Date	Status	Author (Partner)	Description
0.1	11/12/2018		Vincent Bracke	created
0.2	22/01/2019		Bruno Volckaert	commented
0.3	30/01/2019		Vincent Bracke	continued
0.4	06/03/2019		Vincent Bracke	'Integration of IoT Stack through Fort Knox' section to be completed before revision request
0.5	24/04/2019		Tobias Kutzler (IFF)	Added section 'Integration of IoT Stack through Fort Knox'
0.6	29/04/2019		Vincent Bracke Jeroen Hoebeke Bruno Volckaert	Last internal review at IMEC
1.0	29/04/2019		Vincent Bracke	Final version

Quality Check

Version Reviewed	Date	Reviewer (Partner)	Description
0.4	24/04/2019	Acciona	No comments
0.4	24/04/2019	Leitat	No comments
0.4	24/04/2019	IFF	Added content for section 5 No other comments

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

Abbreviations

AIOTI	Alliance for IOT Innovation
API	Application Programming Interface
AR	Augmented Reality
COAP	Constrained Application Protocol
DASH	Dynamic Adaptive Streaming over HTTP
DTLS	Datagram Transport Layer Security
DSS	Decision Support System
HTTP	HyperText Transfer Protocol
ICT	Information and Communications Technology
IoT	Internet of Things
IP	Internet Protocol
IPSO	Internet Protocol for Smart Objects
JWT	JSON Web Token
GPS	Global Positioning System
GS	Green Scheduling
JSON	JavaScript Object Notation
LoRaWAN	Long Range Wide Area Network
LwM2M	Lightweight Machine to Machine
LPWAN	Low-Power Wide Area Network
MSB	Manufacturing Service Bus (IFF-VFK)
OIDC	OpenID Connect
OMA	Open Mobile Alliance
PAN	Personal Area Network
REST	Representational State Transfer
RFID	Radio Frequency Identification
RPT	Requesting Party Token
SCHC	Static Context Header Compression
SDO	Standards Developing Organization
UDP	User Datagram Protocol
UI	User Interface
UMA	User-Managed Access
URI	Uniform Resource Identifier
VFK	Virtual Fort Knox
VINO	Virtual Network Operator
VOD	Video On Demand
WSN	Wireless Sensor Network

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

Executive Summary

The so called "IoT Stack" provided by IMEC is a state-of-the-art cloud platform designed to securely ingest, store and retrieve historical data from its connected devices, supporting multiple IoT protocols. It acts as a middleware layer by decoupling and shielding the data producer from the data consumer (and vice-versa); consequently easing integration, accessibility and future evolution of the connected systems.

This document is structured in five main sections, of which the first one contextualizes the underlying architectural drivers that justify the need of a cloud based middleware layer for the integration of data coming from IoT devices used in ports operations.

The second section introduces the so called 'IoT Stack' which is the platform that is provided by IMEC in the context of this project to realize this integration and illustrates it through a 'demo use-case'. It also provides useful tools aiming at supporting partners when developing and troubleshooting the integration of their software components with the IoT Stack. This section is then followed by two more detailed sections; the first one introduces the main underlying concepts defined within the IoT Stack while the latter mostly focuses on concrete instructions for authentication to the platform, pushing data to the platform and finally retrieving data from the platform.

Finally, the last section introduces the 'Virtual Fort Knox' (VFK) platform provided by Fraunhofer IFF for this project. This platform will be further defined in later specific deliverables, however as it will host the different software components that will be developed for this project and that those components will need to retrieve data from ports IoT devices, it is as well part of the 'cloud platform integration' and consequently is briefly introduced here.

This document is intended for technical partners, members of this consortium, developing in the context of this project device side and/or application side components and provides them the useful information for efficient use of the offered functionalities (and underlying concepts), illustrated when relevant by specific examples.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

Table of Contents

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

Figures

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

1 IoT Middleware as cornerstone in the PortForward architecture[1]

1.1 The challenges that today's ports are facing

Real time operation is the main challenge for any port having an aspiration to manage, minimize and reduce congestion events both within the port and also through the supply chain.

PortForward will attempt to address important challenges that today's ports are facing, in order to take a substantial step towards the Port of the Future. More specifically the following needs are addressed:

- Lack of efficiency in operations with heterogeneous freights such as inefficient land use, berth scheduling & quay crane allocation, quay crane scheduling & bay sequencing, yard configuration & stacking policies, and lack of monitoring of the depth of access channels and quays, which varies due to silting;
- Need for real time monitoring of freight flows through the use of end-to-end track-and-trace solutions in order to optimize port activities;
- Need for remote monitoring and management of important port operations, such as maintenance scheduling, cargo and passenger traffic, especially for short sea shipping cases;
- Interconnection with hinterland transportation with special focus on inland waterways;
- Interface with the surrounding urban environment;
- Experience sharing and transferability to other intermodal transport hubs (ports, airports, etc.);
- Environmental impact reduction through the use of green technologies and energy solutions saving.

To address the aforementioned problems, PortForward proposes a holistic approach that will lead to a smarter, greener and more sustainable port ecosystem and which will include the following features:

- The introduction of an Internet of Things (IoT) concept for port assets (infrastructure, vehicles, cargo, people and processes);
- Sensor deployment, including cameras, multi-modal tracking devices, etc.;
- Interconnection into one seamless, versatile and secure IoT network;
- Remote management and intelligent maintenance of port assets;
- Virtual Port tool embedded in the PortForward Dashboard providing centralized control and alternative visualizations;
- Novel smart logistics platform with Decision Support System (DSS);
- Environmental and energy monitoring/optimization system using Green Scheduling (GS);
- Augmented Reality (AR) for pilot assistance and remote assistance to workers/operators;
- Information exchange layer with other stakeholders, e.g. city services;
- The socio-economic analysis of the port interface with its surrounding area and the port-city, as well as the rest of the logistics value chain.

[1] Most of this section is extracted from the proposal document of the PortForward Project (Proposal number 769267-2) sent by C. Blobner on 19/10/2017 to the Participant Portal Submission Service of the European Commission.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

1.2 The objectives of the project

PortForward proposes a holistic approach that will lead to a smarter, greener and more sustainable port ecosystem and which will include the following features:

- The introduction of an Internet of Things (IoT) concept for port assets (infrastructure, vehicles, cargo, people);
- The socio-economic analysis of the port interface with its surrounding area and the port-city, as well as the rest of the logistic value chain.

There only is one way to eat an elephant: a bite at a time; so has it been for this project by dividing its overall objectives into 10 smaller ones, of which the objective number 3 is reminded hereafter:

O3: Internet of Things (IoT) middleware that facilitates the deployment, discovery and management and unifies the interaction with 1) heterogeneous connected sensors/actuators, 2) tracking devices and 3) connected workers, thereby making use of both short, medium and long range wireless connectivity and leveraging on open IoT standards such as LWM2M/IPSO (…).

1.3 The architectural solution

1.3.1 Summary of the defined architecture

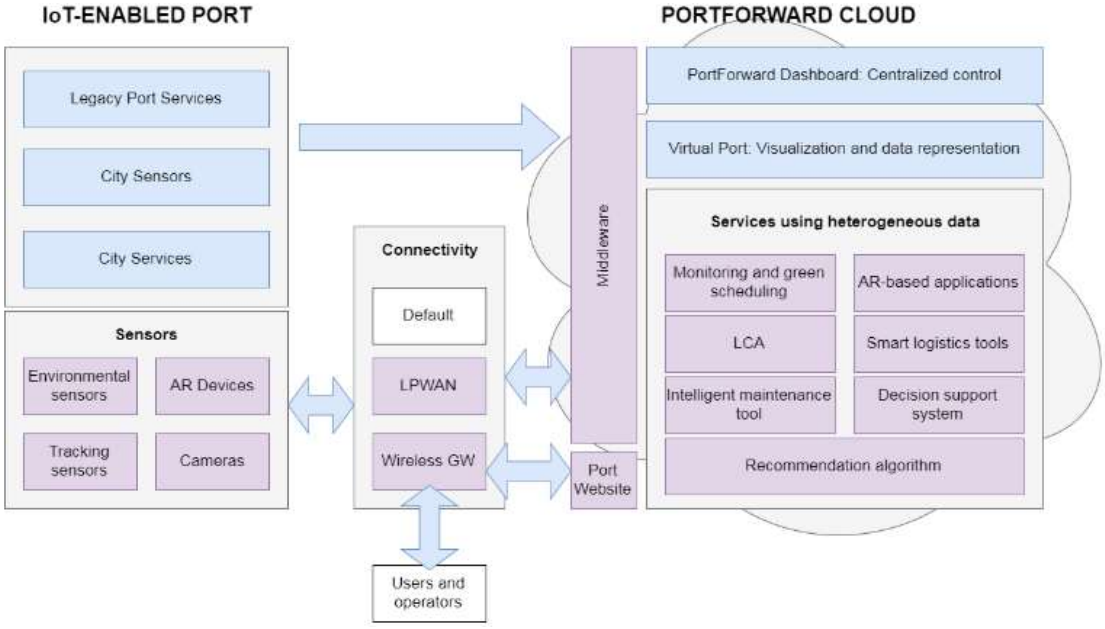

Figure 1 - PortForward Architectural Layers

Figure 1 gives a high-level overview of the PortForward architecture that will be composed of three different layers:

Sensor Layer (IoT-Enabled Port): The main function of this block is to get data from the real physical environment, and convert it into digital data. This includes the IoT data collection using specific hardware systems, specialized sensors, direct input from the human users (as feedback from the field), data coming from open sources, such as SafeSeaNet2, other environmental sensors, etc. According to each use case, there can be different kinds of communication networks: a) LPWA

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

network: for bandwidth limited single-hop networks to cover large areas (up to several km), b) Local Area Network: to cover data acquisition directly from 1m to 1km without multi-hop, c) WSN: for multi-hop monitoring networks and d) Personal Area Network: to cover static platforms, where the distance is below 1m.).

Middleware Layer (Connectivity): The main function of this block is to gather and pre-process the data coming from the sensor layer, using different wireless communication technologies, in order to be processed and consumed by the upper level of the PortForward system. This will enable data collection from very heterogeneous sources, with attention for easy deployment, open standards, management as well as secure communications. Today, interoperability remains one of the biggest constraints in the IoT industry and it will not be different for PortForward. Therefore, the recommendation to adhere to the principles pushed forward by the Alliance for IoT Innovation (AIOTI) and its Standardization Working Group, which aims to build consensus on open ways of working. For the sensor layer, attention will be given to light-weight specifications such as LWM2M that can be applied to a wide range of IoT devices, that tackle key aspects such as device management, data access and security and that build on open standards that are also used by other SDOs. This way, PortForward will ensure interoperability as well as alignment with the future European standards and directives.

Application Layer (PortForward Cloud): this involves the development of the PortForward integrated system which will comprise the Virtual Port implementation, the Decision Support System, the novel Smart Logistics Tool, the Intelligent Maintenance Module, the Green Scheduler and the front-end AR-based applications and user interfaces (PortForward Dashboard).

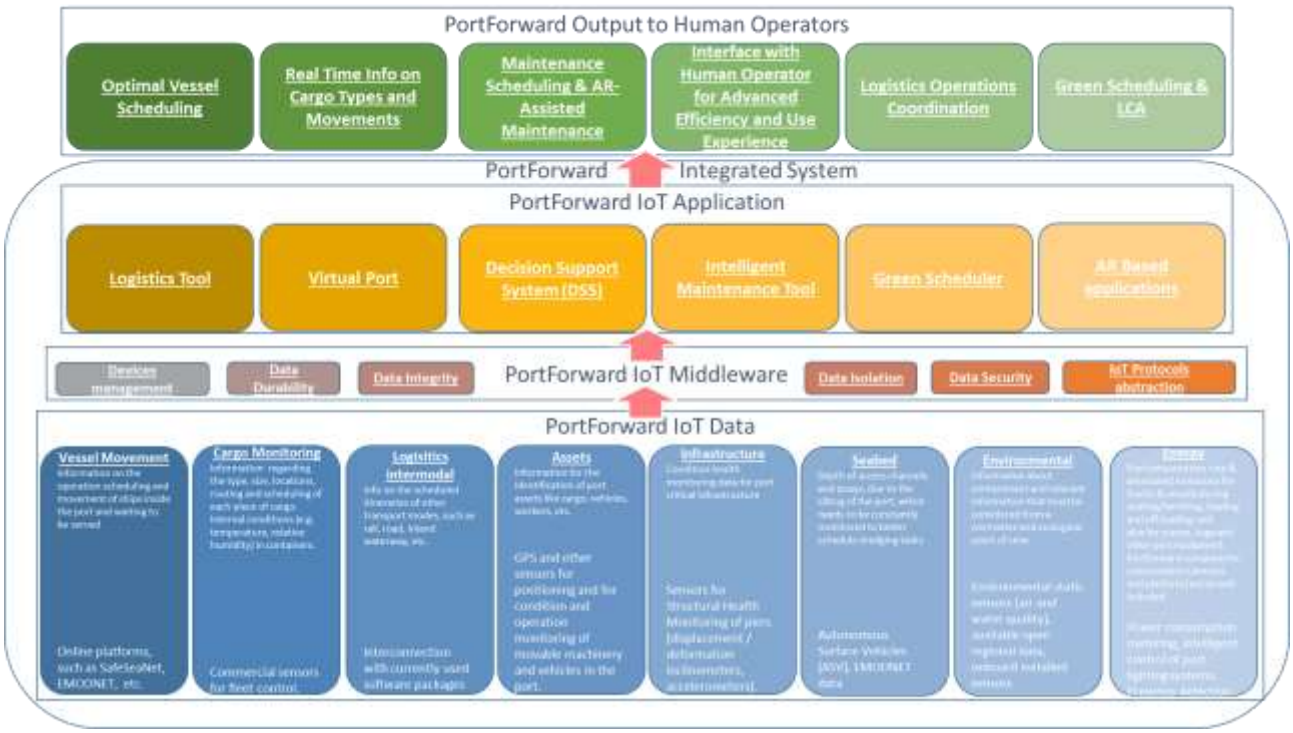

Figure 2 : Layered Component Architecture

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

1.3.2 The IoT Stack as Middleware Cloud Platform

As mentioned, the realization of the port of the future will involve the roll-out of a variety of IoT devices such as sensors, actuators, tracking devices, etc. as well as their integration with a variety of other systems such as the ports' ICT platforms as well as workers' devices. The collected data as well as two-way interactions will enable improved port monitoring, increased traceability, increased efficiency of processes and workers as well as improved environmental monitoring. In this context, IMEC's Internet of Things middleware layer will facilitate the integration of heterogeneous IoT devices that can make use of various wireless communication technologies (e.g. short range WSN technologies, medium-range wireless LAN technologies, long range LPWAN technologies).

The DYAMAND platform enables the integration and abstraction of different technologies and devices typically found in local networks. It is part of the larger City-of-Things framework for smart cities that offers generic APIs to access the collected data and build cross-technology smart applications. This will serve as the starting point for PortForward's IoT middleware and as an enabler for further innovations. In a second phase, IMEC will extend this platform with LWM2M-based device management, discovery and data access solutions. This open light-weight specification uses open standards (e.g. CoAP, IPv6/6LoWPAN) and data models to enable efficient, machine-understandable interactions between ICT backend systems and heterogeneous (embedded) devices (machines, sensors, actuators, etc.).

By leveraging on such standards, not only for the novel targeted IoT devices, but also for the newly to be deployed communication infrastructure (e.g. LoRaWAN connectivity), deployment and management will be facilitated and integration cost reduced. At the same time, support for off-the shelf devices will be achieved via adapters. The end result will be a unified view towards all deployed IoT devices for monitoring, data access and control by exposing standardized interfaces following open IoT specifications and data models, as well as discovery and management capabilities. This will facilitate and speed up the creation of novel services on top.

1.4 Link with the defined use-cases

All use-cases that this project will deliver have been identified and documented[2] and will use the IoT Stack to connect the port (IoT) data sources to the (to be) developed applicative services when relevant (i.e., when bringing added value in comparison to direct connection). Typically, data captured by systems and devices at ports premises[3] are pushed to the IoT Stack that offers this data to the consuming applicative services[4]:

- Protocol and addressing abstraction: data retrieved through standard HTTP REST API protocol (no need to deal with devices specific protocols nor to address them directly).
- Different rich query mechanisms: (combination of) time, location, metric and thing based queries.
- Efficiency, stability and scalability: the platform is built on proven open-source software designed for cloud based platforms having to efficiently process massive data sets.

[2] See : D1.3 – Technical requirements specification

[3] Environmental data (air quality, temperature, wind…), operations and assets related data (id, position, weight, status…), etc.

[4] The enumerated concepts are here only introduced and will be further described in the following sections

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

- Security and data isolation: supported amongst others by the OIDC protocol and the specific concept of scopes.

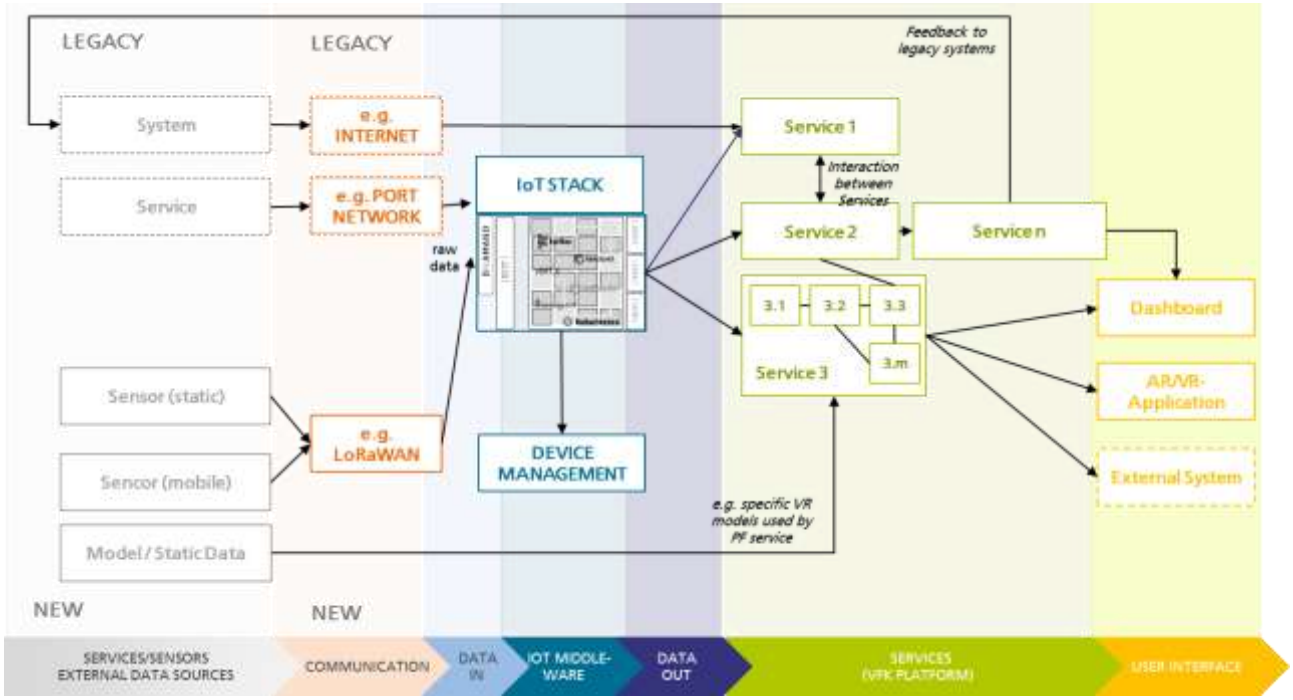

Figure 3 : Generic block diagram of PortForward Architecture approach

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

2 The IoT Stack introduced[5]

In order to reach the vision of an IoT-enabled port, it must be possible to integrate a variety of sensors and actuators, to interact with them and to provide interfaces to services on top for accessing pre-processed data. Starting from the existing IMEC IoT Stack middleware that is being used in the context of the City of Things (Antwerp)[6], T3.1 will set up a similar platform for PortForward and will extend it according to the needs of the innovations of PortForward.

For data access, the IoT platform will offer a generic, scalable API. Apart from providing basic data access, it will enable geoqueries, support statistics and retrieval of historical data. For the intake of data coming from a variety of sensor devices and triggering of actuators, a novel open REST API based on LWM2M/IPSO interfaces and data models for device management (see T3.2) and data access will be added. This will enable the plug-and-play integration of any IoT device adhering to this standard. For devices and systems (e.g. external data sources) that do not support the offered APIs, an adapter concept will be designed that enables the translation of legacy and proprietary data formats to these APIs. These adapters will ensure interoperability and offer a uniform way as to how data can be collected, independently of the underlying heterogeneity.

Alongside this, the platform will be extended to support novel PortForward IoT devices. First, it will be investigated to what extent IoT devices with very limited capabilities or running over bandwidth constrained networks (e.g. LPWAN) can directly make use of the proposed APIs and standards, or whether optimizations in terms of compression and batch transfers are needed to reduce the message size as well as number of messages. Secondly, the incorporation of localization and tracking information in the IoT platform will be added. This involves not only the collection of this data, but also its relations with the actual items being tracked or localized. Thirdly, support for IoT data exchange with AR headsets may be investigated however actuation of AR headsets may have too stringent timing constraints.

IMEC will set up the IoT platform early in the project and extend its features based on WP1 and WP2 outcomes. Together with the rest of the partners, the APIs and concept of adapters will be discussed and refined, after which parties can start the design and implementation of applications interfacing with the IoT platform.

Together with sensor-abstracting solutions like DYAMAND[7], the IoT Stack is able to capture data from all sensors in a scalable software stack.

This stack is deployed in a cloud environment with support for horizontal scaling. Developers can make use of this data and feed their application (e.g. End-User applications, real-time decision-making applications, analysis systems…) using a generic set of APIs.

[5] The content of this section is subject to evolve with the platform; therefore always prefer accessing it online on our website at https://obelisk.ilabt.imec.be/api/v1/docs/

[6] For more details, see: https://www.imec-int.com/en/cityofthings

[7] For more information, see: https://dyamand.ilabt.imec.be/public

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

2.1 Overall architecture

The IoT Stack is a cloud based platform that offers interfaces for IoT device integration within standard (web) applications. Two key aspects have been taken into account by design in the implementation of this multi-tenant platform:

- Scalability: the platform has been designed as a set of reactive micro-services running on top of Kubernetes. The technologies used are state-of-the-art technologies for the ingestion and processing of events streams. As per design, the cloud-based IoT platform is thus highly scalable and consequently there is no intrinsic known limit to the data throughput it can digest, process and render. This being said, it consumes computational, memory and storage resources proportionally to the volume of data having to be processed: therefore the definition of the requirements should remain reasonable. IMEC will review, with each Use-Case owner, the feasibility and sustainability of its requirements before validation (or adaptation): this mainly concerns topics like data retention, volumes of data and frequency of messages.

- Security: in addition to the concept of scopes (allowing data isolation), the platform also makes uses of modern and strong authentication & authorization mechanisms through OIDC.

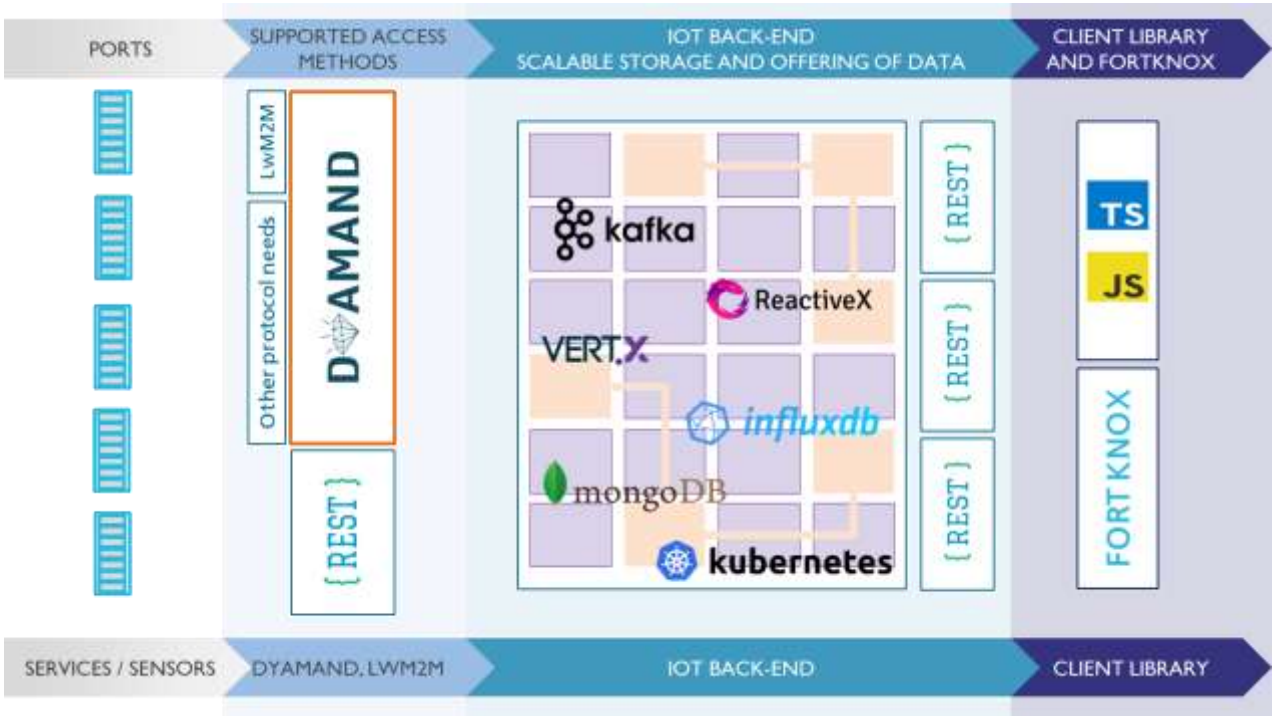

Figure 4 : High Level presentation of the IoT Stack

Importantly, and before further explanations on the platform, the reader should be aware that the IoT Stack will evolve (where possible in a backwards compatible way). When such IoT Stack evolutions occur, IMEC ensures clear and on-time information about new features / changes and will provide the required support to partners integrating these changes.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

2.2 The demo

For didactic purpose, mainly intended to technical partners of the consortium having to integrate their system with the Middleware IoT Stack, this simulation has been developed to quickly showcase how to publish data to / retrieve data from the IoT Stack. Figure 5 is a capture of the Web UI and contains:

1) A live graphical representation of the operations currently running on the port: it shows the trucks and the cranes moving and being (un)loaded as well as the gates opening and closing.

2) A heatmap selection box: allowing to render, as a superposed layer on the graphical representation of the port, a heat map of the most used routes of the port during the last X hours (another example of heat map might represent air-quality at different places of the port for instance). NB: this is typically a calculation that should be pre-processed to avoid heavy calculation at runtime (no pre-processing was implemented in the demo, resulting in delay between the selection of a period and the graphical rendering of the heat map).

3) Some statistics/reporting: in this demo two basic figures are reported: the number of trucks currently in the port as well as the average time spent by trucks in the port.

4) Last but not least an informative box presenting the events data stream received by the IoT Stack in (near) real-time.

Figure 5 : UI of the demo simulation

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

A high level design of this demo is presented in Figure 6; there are four main components[8]:

1. The World Simulator: simulating different 'Things' (gates, trucks and gates) running simultaneously and sending their data to the IoT Stack through the use of the 'ingest' API (after successful authentication through the dedicated API).

2. The IoT Stack: the Middleware platform that shall be used in the context of this project and that is being called by the different 'Things' (see previous bullet) but also by the dashboard application (see next bullet).

3. The Dashboard: that represents the 'consuming' application (or service) querying the IoT Stack to retrieve some specific data. Pre-processing should occur here if implemented.

4. The Browser, or end-user interface, here a graphical web interface querying the Dashboard application and presenting the formatted results to the end-user.

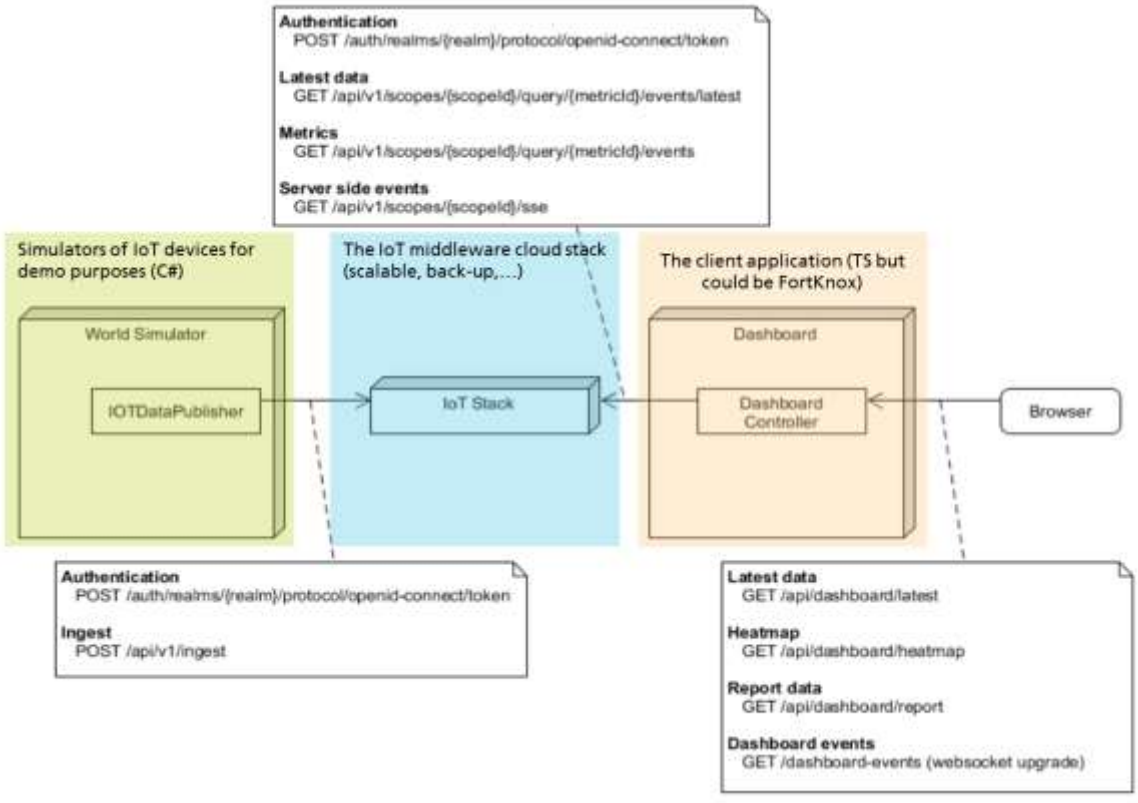

Figure 6 : High Level Design of the demo

[8] The programming languages used to code this demo are C# for the 'World Simulator' and TrueScript for the Dashboard. Other languages or libraries (e.g.: Java, JavaScript…) are also able to connect to and use the IoT Stack APIs.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

When zooming on the ingested events (the messages sent by the Devices/Things to the IoT Stack through the ingest API):

- The trucks and small cranes all send their id, weight and geolocation every second to the stack (even when none of those values have evolved since the last data exchanged).

- The gates that only send events when a truck enters/leaves the port with the license plate of the truck and their geolocation (always the same for each gate).

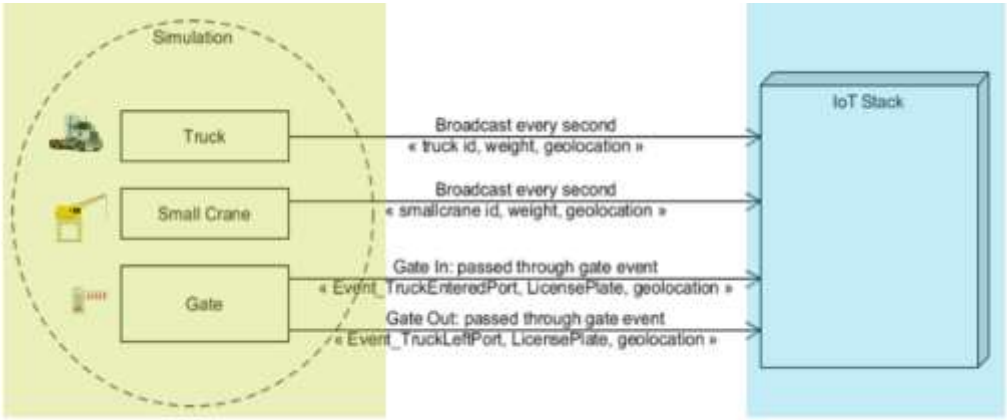

Figure 7 : Data ingestion

Last but not least, hereafter a more detailed focus on the data retrieval:

- The initialization phase collects the latest recorded value for each ‘metric’ of every ‘thing’. This way, the different trucks can be instantiated and placed according to their position on the map.

- The browser (through the Dashboard service) subscribes to Server-Sent-Events (‘SSE’), allowing direct streaming of events from the IoT Stack to the web interface. The things on the map start moving. The ‘Events’ section in the UI reports a filtered view on the ‘Event_TruckEnteredPort’ and ‘Event_TruckLeftPort’ events.

- When selecting a heat map duration in the UI, a heat map generation request is sent from the browser to the Dashboard service that performs a time-based query (last x hours). The dashboard application does some data filtering (truckID) and conversion (position) and returns the list of events to the browser that visually renders this input as an extra layer on the map. Typically as to reduce the latency experienced by the end-user when clicking on a duration, the Dashboard service should be improved as to periodically, in the background, pre-fetch, filter and convert data to avoid this being done at runtime.

This project has received funding from the
European Union’s Horizon 2020 research and innovation program under grant agreement No 769267.

- For the report section (‘number of trucks in port’ and ‘average time spent in port’) the browser periodically asks the Dashboard service to compute and return those. To that end, the Dashboard queries the IoT Stack, performs the needed computing and returns the requested values. Here again, some optimization are possible by at least decoupling the retrieval and computing from the answer to the browser. Another optimization could be that the dashboard service derives at runtime those figures from the events when passing by instead of performing an extra specific query.

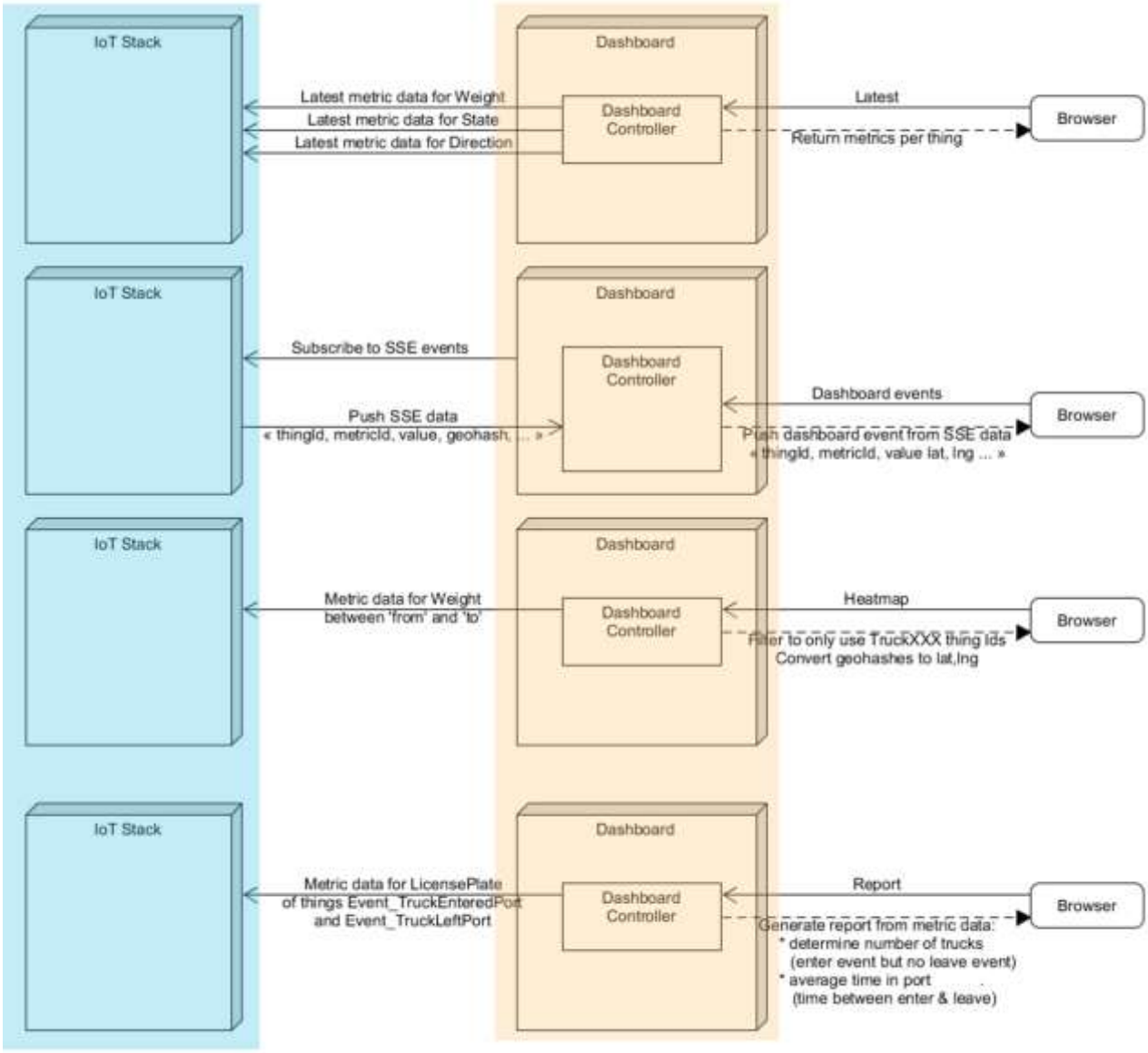

Figure 8 : Data retrieval

This project has received funding from the
European Union’s Horizon 2020 research and innovation program under grant agreement No 769267.

2.3 Tools and support for the developers

2.3.1 The Swagger UI

The complete server-side REST API is documented in OpenAPI 3.0 specification. IMEC is hosting a swagger endpoint to interact with it. After logging in, it offers documentation of the different allowed operations and provides a convenient and interactive way for experimenting those (sandbox).

2.3.2 The IoT Stack Explorer

The IoT Stack Explorer is a web interface aimed at supporting developers while developing and troubleshooting the integration of their component with the IoT Stack. It allows to look at historically ingested data as well as to observe in real-time the entering data. It is not intended for any other purposes (i.e. it must not be considered as a user interface for end-users).

2.3.3 The IoT Stack Client

The IoT Stack Client is a client library written in Typescript to interact with the IoT Stack API. This library makes use of RxJS (Reactive Extensions for JavaScript: a library for reactive programming using observables that makes it easier to compose asynchronous or callback-based code (RxJS)).

The client has a clear purpose:

- Make it easier to do follow up requests on Temporal Pages (see section 3.6 hereunder)
- Do the heavy lifting on authentication and authorization (see section 4.1 hereunder)

The client uses Keycloak's Javascript library to connect to IoT Stack back-end Keycloak Authorization Server and handles i) logging in to a supported Identity Provider (eg. Google), ii) getting the access token, iii) getting the RPT token, iv) refreshing tokens when needed and v) login/logout support.

The client allows to create *Endpoints* that can be acted on with methods like *execute()* or *get()*. An endpoint takes an API uri as argument.

2.3.4 Online documentation and remote support

Most of the useful information on the Middleware IoT Stack can be found at: https://obelisk.ilabt.imec.be/api/v1/docs/. Hereafter, a few documentation and support sources:

Topic	For	Link
Mailing list	PortForward WP3	portforwardwp3@lists.ugent.be
Support service desk	Issues, requests, scope problems	https://iothelpdesk.ilabt.imec.be
IoT-stack API	HTTP Rest API Swagger	https://obelisk.ilabt.imec.be/swagger
IoT-stack client library API	Typescript client API	https://www.npmjs.com/package/@obelisk/client
IoT-stack Explorer	IoT Explorer Webapp	https://obelisk.ilabt.imec.be/explorer

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

3 Main concepts of IMEC's IoT Stack

3.1 Things

A Thing is the all-encompassing name for sensors and actuators. They are addressable with an id, and can measure more than one type of measurement (called a metric – see after). A Thing can in fact be a single hardware device (with one id) that has multiple sensor-heads. This means it can for instance measure GPS, air humidity and temperature at the same time (all different metrics).

Sensors and actuators are typically made by several different companies, all speaking different protocols. This is where an abstraction layer like DYAMAND or the support for an IoT standard like LwM2M come in. They allow to abstract everything down to generic sensors or actuators that send understandable messages to the IoT Stack cloud backend, 'whatever' their underlying communication protocol is.

3.2 Metrics

A metric is a type of measurement. Examples are air quality, temperature, humidity, etc. It is an important aspect of how data is stored; data of the same metric is kept together. The name of the metric is also an important identifier for querying data via the API.

3.3 Events

Sensor devices detect state changes of what they are observing (lightening, air quality, position, temperature, weight, timer, etc.) and send this as input to an application for further treatment. Therefore the Middleware IoT Stack uses the concept of events to store the data that has been sent by a device in reaction to some trigger. The Middleware IoT Stack doesn't query a device when an application requires data about it but rather collects and returns (a collection of) its last registered value(s).

3.4 Scopes

The IoT platform is built with data isolation in mind. Pragmatically speaking this means that users or developers using/creating an application that shows data from company A, should not see any data of company B, although it is hosted on the same platform.

A scope allows to answer those two questions: *which data are we talking about? and who can access the data?*

A scope has a few ways to isolate data:

- Time period
  - Bounded: only records in time period (from - to) can be accessed
  - Unbounded: only records after a certain data can be accessed (from - no-end)
- Geographic area: only records within the area can be accessed
- Metrics: list of metrics that can be accessed
- Things: list of things (sensor ids) that can be accessed
- Data selectors: to select a group of sensors based on internal context tags (set when data was sent).

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

A scope can assign users or groups of users. Each of these has some associated rights on which class of API calls they can do. In the context of the PortForward project, IMEC's proposal is to structure scopes as illustrated in Figure 9 hereunder.

portforward.{*env*}.{*port*}(.{*uc*})	
{*env*}	s (for staging) / p (for production)
{*port*}	3/4-letters abbreviation for port identification
(.{*uc*})	(Optional) The use-case name (abbreviations preferred, e.g. 'gs' for green scheduler, etc.). To be used if specific data isolation is needed for a use-case.

Figure 9 : Proposed structure for Portforward scopes

3.5 Geohashing

Geohashes, created by Gustavo Niemeyer in 2008 and placed in the public domain, are an elegant and succinct geographic encoding. Geohashes work by reducing a two-dimensional longitude, latitude pair into a single alphanumeric string where each additional character adds precision to the location.

The algorithm for creating a geohash works by consecutively splitting the world in four parts, once longitudinal, once latitudinal. This way a long binary string (encoded in base 32) is created that identifies a smaller and smaller space on the world map, the longer the string gets (as illustrated in Figure 10).

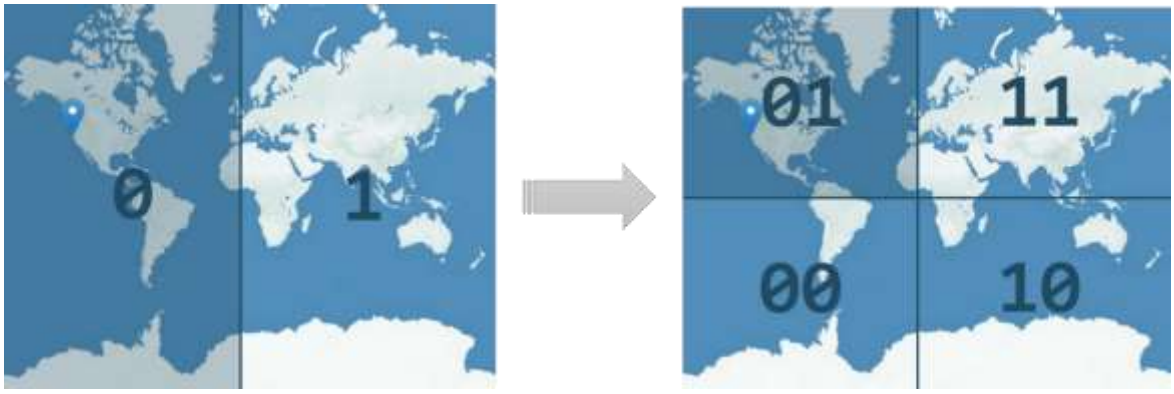

Figure 10: Principle of Geohashing

As an example, the Eiffel Tower (Paris, France) is located at:

Geohash coordinates	Lat/Long coordinates
u09tunqu1	48.8583 / 2.2945

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

Interesting properties

There are some interesting properties associated with geohashes that can be exploited in favor of scalability and performance:

1. URL friendly mechanism
   - `/locations/u09tunqu1/{typeId}/events` vs `/locations/48.8583,2.2945,range=1km/{typeId}/events`
2. Limit false queries
   - Geohashes with wrong prefix, are outside the search area.
3. Zooming and neighbour search
   - The more digits, the more precise: drop (least significant) digits and the area zooms out.
   - Easy to discover neighbouring geohashes.
4. Cacheable
   - Deterministic and absolute locations which is excellent for caching behaviour.
   - By limiting the precision of the geohashes to 6-9 characters, there is a bigger chance of overlapping request patterns / reuse.
5. Performance gains
   - Searching geospatial results with a simple prefix text search.

LatLong vs geohash

The above reasons already go a long way towards advocating the use of geohashes. Although more widely known than a geo-location mechanism, LatLng is not that handy to use.

1. You cannot just guess a LatLng, you have to look it up in a tool like Google Maps.
2. A LatLng exists of 2 doubles, which are more cumbersome in urls.
3. A LatLng only defines a point, you need a range for an area.
4. LatLng + radius is a circular area, which in a lot of cases is not ideal.
5. In those cases that a circular area makes sense, there isn't really a reason not to use a square area. However, when really needed, libraries are available to return a set of geohashes that cover a given circular area.
6. LatLng + radius is complex for monitoring something like a road for instance.

3.6 Temporal paging

In a classical paging system you can ask a page of X results and then request the next page. Implementing something similar would hurt the caching ability (since query parameters are not cached) and it would mean in order to correctly handle incoming requests that a lot more states had to be kept. That is why this system tries to combine both concepts: scalability of not having complex queries that take too long and return way too much results and the added benefit that everything stays as cacheable as possible without introducing more state.

Temporal brackets

In the backend, the frequency/rate of the sensor data is analyzed. According to that a dynamic temporal strategy can be chosen (daily, hourly, minutely). Brackets will be split according to this strategy (e.g. in the case of hourly: `[15h-16h[,[16h-17h[`, etc.). Along with the responses a Link header will be sent with links to the previous and next brackets. This is done according to RFC 5988.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

Caution

All timestamps are UTC based. Locale conversions should be handled in the client if needed.

Example use cases

These use cases illustrate the 2 different ways one can come in contact with the temporal paging mechanism. Both images show a completely expanded view of a series of requests.

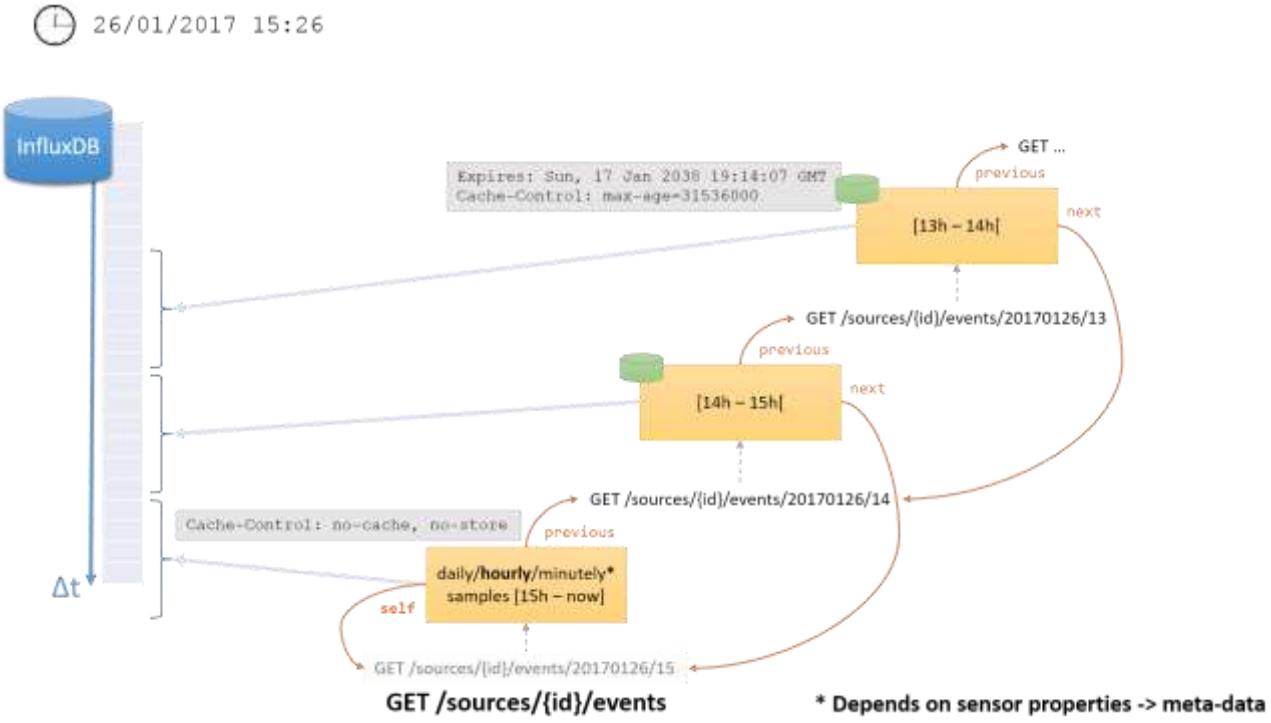

Figure 11 : Temporal Paging - Last available data

In this case everything is triggered by a simple request `GET /sources/{id}/events`. What happens next:

1. The back-end automatically determines that the best temporal strategy for this source is hourly.
2. The back-end returns the bracket from the previous full hour until now
   - This is a currently filling bracket, so it is marked as not cacheable in the header.
   - There is a `Link` header part that contains information about how to fetch the `next` bracket.
   - There is a `Link` header part that contains information about how to fetch this exact bracket again: `self`.
3. If the client wants to go back a bit more to have more data, it can follow the `next` link and do a `GET` request on that URL.
4. This time the server responds with the full hour bracket and marks it as cacheable.
   - Again there is a `Link` header that contains the `self`, `next` and this time also `prev` links.
5. Once the client decides that it has enough data, it stops sending requests.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

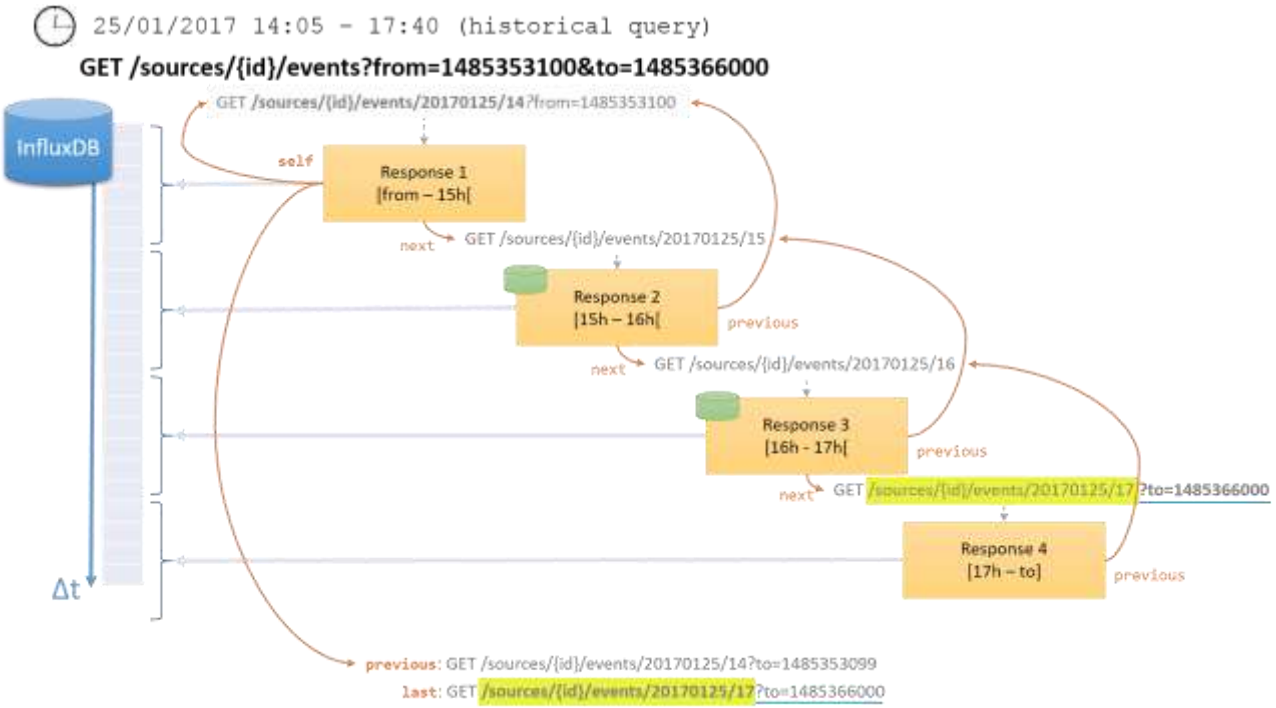

Figure 12 : Temporal paging - Historical data

In this case everything is triggered by a simple request `GET /sources/{id}/events?from=1485353100&to=1485366000`. What happens next:

1. The back-end automatically determines that the best temporal strategy for this source is hourly.
2. The back-end returns the bracket from the 'from timestamp' up until the next full hour.
   - This is not a complete bracket, so it is marked as not cacheable in the header.
   - There is a `Link` header part that contains information about how to fetch the `next` bracket.
   - There is a `Link` header part that contains information about how to fetch the `prev` bracket.
   - There is a `Link` header part that contains information about how to fetch this exact bracket again: `self`.
   - There is a `Link` header part `last` too, that references the URL of the last bracket.
3. To get the subsequent data of the from-to interval, the client must follow the `next` links.
   - These brackets are all full and are marked as cacheable.
4. On each response, the client checks if the `next` link does not have the same path as the `last` link from step 2.
5. Once the path of the `next` and the `last` link are equal, the client must append the querystring to the last request.
6. The server will return the incomplete bracket for the last full hour up until the to timestamp.
   - This will be marked uncacheable, since it is not a full bracket.
7. The client now has the complete requested interval and stops sending requests.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

3.7 The concepts in practice (Swagger UI)

This section aims at more in-depth looking at the usage of those concepts and how they have been implemented within the IoT Stack APIs (see https://obelisk.ilabt.imec.be/swagger for the exhaustive list of available APIs with their documentation). As a first example, this query: `/api/v1/scopes/portforward.demo1/things/SmallCrane1/metrics/Weight/events/20181121/12` (see the exact query in the Figure 13 – note the bearer has been replaced by 'xxx'). Conceptually this means: "retrieve all raw events received on the 21st of November 2018 between noon and 1 PM from the Thing (/device) SmallCrane1 about the Metric Weight". As this thing also systematically sends its position it is returned as well. A list of 2995 records is retrieved, of which the 60 first are listed in Figure 13 as illustration (for the sake of readability the column RowID has been added and the Unix Epoch Timestamp has been transformed into a human readable format. Finally some redundant information like the scope have been removed).

curl -i -X GET "https://idlab-iot.tengu.io/api/v1/scopes/portforward.demo1/things/SmallCrane1/metrics/Weight/events/20181121/12" -H "accept: application/json" -H "authorization: Bearer xxx"

RowID	Unix Epoch Timestamp as Human readable format	GeoHash	Thing	Weigh	RowID	Unix Epoch Timestamp as Human readable format	GeoHash	Thing	Weigh
1	2018-11-21 12:00:00.657	u15h3qs2g9n2	SmallCrane1	2000	31	2018-11-21 12:00:33.515	u1551njt3x3d	SmallCrane1	2000
2	2018-11-21 12:00:01.468	u15h3kh75eqr	SmallCrane1	2000	32	2018-11-21 12:00:34.870	u1551njt3x3d	SmallCrane1	2000
3	2018-11-21 12:00:02.842	u15h32k2g8y3	SmallCrane1	2000	33	2018-11-21 12:00:35.775	u1551njt3x3d	SmallCrane1	2000
4	2018-11-21 12:00:03.345	u15h1qsk7en6	SmallCrane1	2000	34	2018-11-21 12:00:36.782	u1551njt3x3d	SmallCrane1	2000
5	2018-11-21 12:00:04.756	u15h16kk7tqm	SmallCrane1	2000	35	2018-11-21 12:00:37.587	u1551njt3x3d	SmallCrane1	2000
6	2018-11-21 12:00:05.261	u15h12urg8q2	SmallCrane1	2000	36	2018-11-21 12:00:39.401	u1551njt3x3d	SmallCrane1	2000
7	2018-11-21 12:00:06.349	u155cqb1bgk1	SmallCrane1	2000	37	2018-11-21 12:00:40.187	u1551njt3x3d	SmallCrane1	45000
8	2018-11-21 12:00:08.063	u155c4j83w3t	SmallCrane1	2000	38	2018-11-21 12:00:43.693	u1551pvece1w	SmallCrane1	45000
9	2018-11-21 12:00:08.567	u155c0tec91x	SmallCrane1	2000	39	2018-11-21 12:00:44.497	u15534m9989w	SmallCrane1	45000
10	2018-11-21 12:00:09.474	u1559jv898c8	SmallCrane1	2000	40	2018-11-21 12:00:46.741	u15535vtcw3t	SmallCrane1	45000
11	2018-11-21 12:00:10.481	u15594tx3tce	SmallCrane1	2000	41	2018-11-21 12:00:48.457	u15590vx1ecs	SmallCrane1	45000
12	2018-11-21 12:00:11.389	u15590jectcw	SmallCrane1	2000	42	2018-11-21 12:00:50.043	u1559jts9s3x	SmallCrane1	45000
13	2018-11-21 12:00:12.898	u15535mw3t3d	SmallCrane1	2000	43	2018-11-21 12:00:51.048	u155c0tt9x3s	SmallCrane1	45000
14	2018-11-21 12:00:14.613	u1551nmt1xcs	SmallCrane1	2000	44	2018-11-21 12:00:52.153	u155c5vs9s9t	SmallCrane1	45000
15	2018-11-21 12:00:16.122	u1551njt3x3d	SmallCrane1	2000	45	2018-11-21 12:00:53.160	u155cnucver1	SmallCrane1	45000
16	2018-11-21 12:00:17.697	u1551njt3x3d	SmallCrane1	2000	46	2018-11-21 12:00:54.366	u155bwf3fg31	SmallCrane1	45000
17	2018-11-21 12:00:19.061	u1551njt3x3d	SmallCrane1	2000	47	2018-11-21 12:00:55.170	u155bnvcy5m1	SmallCrane1	45000
18	2018-11-21 12:00:20.392	u1551njt3x3d	SmallCrane1	2000	48	2018-11-21 12:00:56.175	u14gzwy3y7k1	SmallCrane1	45000
19	2018-11-21 12:00:20.996	u1551njt3x3d	SmallCrane1	2000	49	2018-11-21 12:00:56.995	u14gzqfcze21	SmallCrane1	45000
20	2018-11-21 12:00:22.608	u1551njt3x3d	SmallCrane1	2000	50	2018-11-21 12:00:58.002	u14gyzj593cx	SmallCrane1	45000
21	2018-11-21 12:00:23.290	u1551njt3x3d	SmallCrane1	2000	51	2018-11-21 12:00:59.308	u14unfvjck9t	SmallCrane1	45000
22	2018-11-21 12:00:24.699	u1551njt3x3d	SmallCrane1	2000	52	2018-11-21 12:01:00.782	u14unzjjc63d	SmallCrane1	45000
23	2018-11-21 12:00:25.203	u1551njt3x3d	SmallCrane1	2000	53	2018-11-21 12:01:01.085	u14uqbjnc69t	SmallCrane1	45000
24	2018-11-21 12:00:26.789	u1551njt3x3d	SmallCrane1	2000	54	2018-11-21 12:01:01.991	u14uqfv4369e	SmallCrane1	45000
25	2018-11-21 12:00:27.493	u1551njt3x3d	SmallCrane1	2000	55	2018-11-21 12:01:04.214	u14uwbm116cx	SmallCrane1	45000
26	2018-11-21 12:00:28.886	u1551njt3x3d	SmallCrane1	2000	56	2018-11-21 12:01:04.818	u14uwfj5371s	SmallCrane1	45000
27	2018-11-21 12:00:29.291	u1551njt3x3d	SmallCrane1	2000	57	2018-11-21 12:01:05.724	u14uwfmncrcw	SmallCrane1	45000
28	2018-11-21 12:00:30.498	u1551njt3x3d	SmallCrane1	2000	58	2018-11-21 12:01:06.730	u14uwfmncrcw	SmallCrane1	2000
29	2018-11-21 12:00:31.805	u1551njt3x3d	SmallCrane1	2000	59	2018-11-21 12:01:08.370	u14uqztp97c8	SmallCrane1	2000
30	2018-11-21 12:00:32.509	u1551njt3x3d	SmallCrane1	2000	60	2018-11-21 12:01:09.476	u14uqut09q3w	SmallCrane1	2000

Figure 13 : A thing based query

The crane moves unloaded before stopping and waiting for a new container that it then gets (+43000 in weight, row 37) and then starts moving again before unloading it (row 58) and leaving again (row 59).

Now as a second example, complementary to the first one, a location based query will be examined: `/api/v1/scopes/portforward.demo1/locations/u1551njt3x3d/Weight/events/20181121/12` (see the exact query in the Figure hereafter – note the bearer has, here again, been replaced by 'xxx'). Conceptually this means: "retrieve all raw events received on the 21st of November 2018 between noon and 1 PM from Things that passed through this position and that report on/measure

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

their Weight". A list of 1412 records is being retrieved, of which the 30 first are listed in Figure 14 as an illustration (for the sake of readability, here again, the column RowID has been added and the Unix Epoch Timestamp has been transformed into a human readable format. Finally some redundant information like the scope has been removed).

curl -X GET "https://idlab-iot.tengu.io/api/v1/scopes/portforward.demo1/locations/u1551njt3x3d/Weight/events/20181121/12" -H "accept: application/json" -H "authorization: Bearer xxx"

Row ID	Unix Epoch Timestamp as Human readable format	GeoHash	Thing	Weight
1	2018-11-21 12:00:16.122	u1551njt3x3d	SmallCrane1	2000
2	2018-11-21 12:00:17.697	u1551njt3x3d	SmallCrane1	2000
3	2018-11-21 12:00:19.061	u1551njt3x3d	SmallCrane1	2000
4	2018-11-21 12:00:20.392	u1551njt3x3d	SmallCrane1	2000
5	2018-11-21 12:00:20.996	u1551njt3x3d	SmallCrane1	2000
6	2018-11-21 12:00:22.608	u1551njt3x3d	SmallCrane1	2000
7	2018-11-21 12:00:23.290	u1551njt3x3d	SmallCrane1	2000
8	2018-11-21 12:00:24.699	u1551njt3x3d	SmallCrane1	2000
9	2018-11-21 12:00:25.203	u1551njt3x3d	SmallCrane1	2000
10	2018-11-21 12:00:26.789	u1551njt3x3d	SmallCrane1	2000
11	2018-11-21 12:00:27.493	u1551njt3x3d	SmallCrane1	2000
12	2018-11-21 12:00:28.886	u1551njt3x3d	SmallCrane1	2000
13	2018-11-21 12:00:29.291	u1551njt3x3d	SmallCrane1	2000
14	2018-11-21 12:00:30.498	u1551njt3x3d	SmallCrane1	2000
15	2018-11-21 12:00:31.805	u1551njt3x3d	SmallCrane1	2000
16	2018-11-21 12:00:32.509	u1551njt3x3d	SmallCrane1	2000
17	2018-11-21 12:00:33.515	u1551njt3x3d	SmallCrane1	2000
18	2018-11-21 12:00:34.870	u1551njt3x3d	SmallCrane1	2000
19	2018-11-21 12:00:35.775	u1551njt3x3d	SmallCrane1	2000
20	2018-11-21 12:00:36.782	u1551njt3x3d	SmallCrane1	2000
21	2018-11-21 12:00:37.587	u1551njt3x3d	SmallCrane1	2000
22	2018-11-21 12:00:39.201	u1551njt3x3d	Truck860	45000
23	2018-11-21 12:00:39.401	u1551njt3x3d	SmallCrane1	2000
24	2018-11-21 12:00:40.187	u1551njt3x3d	SmallCrane1	45000
25	2018-11-21 12:00:40.187	u1551njt3x3d	Truck860	2000
26	2018-11-21 12:01:11.291	u1551njt3x3d	SmallCrane0	2000
27	2018-11-21 12:01:12.197	u1551njt3x3d	SmallCrane0	2000
28	2018-11-21 12:01:13.487	u1551njt3x3d	SmallCrane0	2000
29	2018-11-21 12:01:14.091	u1551njt3x3d	SmallCrane0	2000
30	2018-11-21 12:01:14.999	u1551njt3x3d	SmallCrane0	2000

Figure 14 : A location based query

On its own this table already provides interesting information: the Thing 'SmallCrane1' remained at this location during 24sec and 65 ms. It was unloaded and received its container from 'Truck 860' in a little bit less than 1sec (fictive values from the demo) before immediately leaving the area. 31 seconds and 104ms after, 'Smallcrane0' arrived empty and waited at least for 3sec and 708ms.

Now combining the information contained in both queries, some first bricks for effective management and scheduling of port operations appear: where SmallCrane1 came from and went to, at which precise time it was loaded/unloaded and from/to which truck, etc.

It is now clear that this layer is no standard Middleware for multi-purpose integration (like ESBs, MQ Systems, etc.) but rather a specialized integration layer for the IoT.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

3.8 Planned feature: video streaming

The IoT Stack can not only process simple sensor data, it can also handle larger audio/video streams. Streams can be ingested in a similar fashion to sensor data and are automatically transcoded to the desired qualities and stored on disk.

Once a stream is ingested it can be viewed as a live stream, using various qualities to produce an adaptive bit rate (ABR) live feed using the DASH technology. It also supports video-on-demand (VOD) by requesting a specific time range to only get that specific segment in a desired quality. This makes it easy for post processors to retrieve the live data in chunks, extract information and then continue with the next chunk.

To prevent having an infinitely growing storage problem, older VODs are gradually degraded, i.e. the highest qualities get removed first. The VOD API has a few strategies for handling missing quality: it can fall back to reduced quality, it can skip these chunks altogether or it can reject the request.

The video API also allows to retrieve a specific snapshot at a given time. This can be useful to combine with the post processor's result.

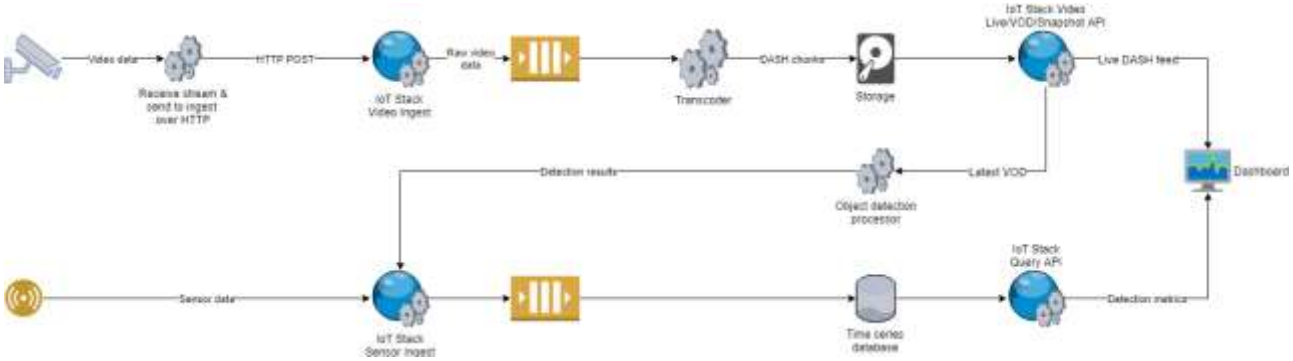

Figure 15 : Video streaming with object detection

Figure 15 represents the overall mechanism of this feature. The video stream is being ingested through the VideoIngest API that transcode it to the various qualities (240p, 480p…) that it needs to be transcoded to. For instance, if a stream from a CCTV camera is ingested and the end user wants to know if persons were detected between 02:00 and 04:00 in the morning then the following needs to be set up:

1. A stream needs to be defined along with the various qualities (240p, 480p, …) that it needs to be transcoded to.
2. The data of the newly created stream needs to be pushed to the video ingest API of the IoT Stack.
3. A post processing unit does:
    a. Infinitely retrieve the x last minutes of the stream from the VOD API.
    b. Run the object detection module (e.g. YoloV3).
    c. Push the detection result as "sensor" data to the non-video parts of the IoT Stack.
4. With the query API all 'detection result' metrics between 02:00 and 04:00 can be retrieved and analyzed whether the 'person' object class is present.
5. If persons were detected, the snapshot endpoint can be used to retrieve an image from the stored data at the given time of detection for visual confirmation.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

These video API extensions to the IoT Stack are currently implemented as a prototype and its endpoints are subject to change.

3.9 Features provisioned outside of the IoT Stack

So far what the Middleware IoT Stack platform is or shall be has mostly been discussed but no time has been spent to explain what it is not (and consequently does not and shall not offer), something that will be further detailed in this section but can already be summarized as such: this Middleware layer is not intended to host 'business specific' logic or data; it mainly is to remain a 'universal' cloud connector for IoT devices to be easily integrated within classical applications.

For the sake of clarity, the importance of the hereafter listed topics for a successful end-to-end implementation of this project objectives must not be underestimated, and for those different topics IMEC would actively contribute to (if not keep the ownership of) their design and implementation as part of this project deliverables.

3.9.1 Pre-processing

(Pre-)Processing can be described as the activities and tasks to perform cleaning, preparation, transformation, wrangling, edition, linking, computation, reduction, sorting, ordering and storing on an incoming stream of data **to accommodate any needed business specific outflow**.[9]

There is of course no debate on the need for data (pre-)processing as this feature brings great added value when it comes to seamlessly render results of large computation to an end user as experienced in the demo use-case with the heat map[10] where no pre-processing capability has been put in place; there are many such and even more complex components in the scope of this project that would definitely need this capability (Decision Support System, Virtual Reality, Augmented Reality, etc.) that this key dimension can't just be ignored.

However, and as explained in previous sub-section, the IoT Stack, as IoT specialized Middleware integration layer, is not the best place for this and therefore it should be recommended to implement this in the upper architectural layers (ideally through reactive[11] micro-services[12] as well) like in the applicative containers hosted in the Virtual Fort Knox environment. Figure 16 depicts such an approach with (low-level) data services processing the incoming flow (or querying data) from the IoT Stack and persisting the processed outcome as desired. Higher level applicative services (like the Green Scheduler for instance) making use of those generic and reusable low-level data oriented services.

[9] This definition and the different interpretation that could be done of it should not be considered as binding but rather as an illustration.

[10] On a more technical perspective, besides the impact on the end-user, having no preprocessing in place would negatively impact the overall performance of the IoT Stack and thus of all the applicative components, including the end-user facing ones

[11] Reactive programming particularly fits event based data streams (what IoT applications typically are)

[12] A micro-services architecture, which besides its numerous advantages, fits cloud platform deployment (most especially elasticity).

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

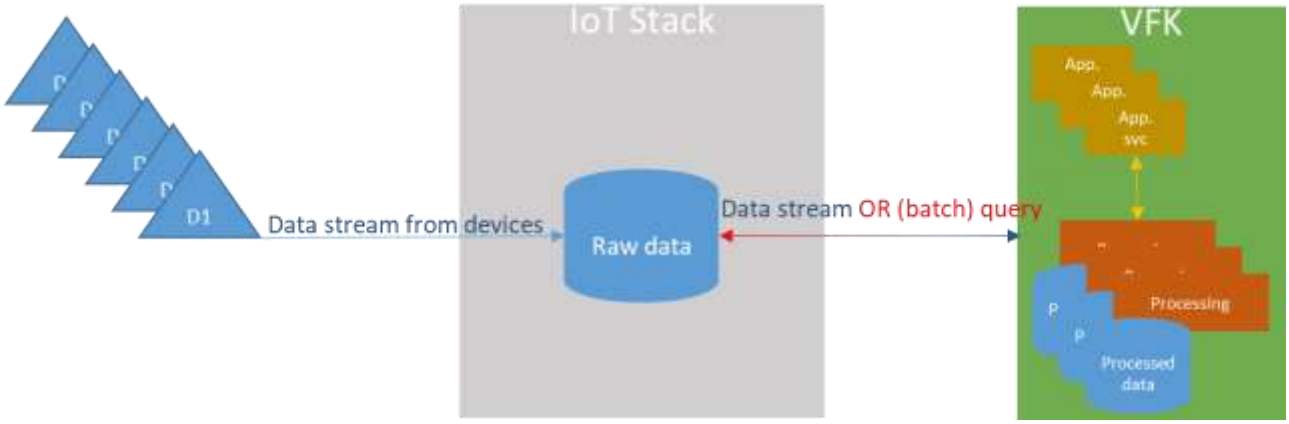

Figure 16 : (Pre-)Processing of data in VFK

Now taking the video ingestion capability (see section 3.8) as an example, the Middleware IoT Stack performs internally processing when transcoding or when performing object detection on the video stream. As this happens the same way for any type of video stream and is not different for one use case or another, it is deemed being functionally agnostic (i.e. having no purpose of accommodating any business specific outflow) and consequently can be seen as some sort of black-box provided feature. To avoid any potential misunderstanding, it won't be further commented in the context of the IoT Stack: (pre-)processing only occurs within the consuming upper layer(s) (like the containers hosted within the VFK environment).

3.9.2 Meta-Data and other non-IoT data

As explained in the introduction of this section, the Middleware IoT Stack "*mainly is to remain a 'universal' cloud connector for IoT devices to be easily integrated within classical applications*"; it should therefore not be compared with or understood as a classical multi-purpose middleware (like an Enterprise Service Bus or a Message Queue Broker for instance); the very nature of its design makes it efficient for IoT driven data stream ingestion, so typically for frequent, small sized events messages representing the state of a sensor device. Accordingly, the data structure used within the IoT Stack has been designed to store devices ('things') and their related metrics attributes (position, weight, temperature, state, humidity, etc.); so it is not tailored to large unstructured data that can typically be found within classical applicative databases. To integrate such content from ports' legacy systems within PortForward, an ad-hoc and direct integration should be preferred.

For instance: in the case of a container, a typical data stream that would be ingested by the IoT Stack would be its current location, temperature, humidity level and state (open/closed) but it should not be data from a legacy asset management system like owning company, description of the goods it contains, clearance documents, last maintenance date and operator, etc.

In order to avoid doubt, IMEC will review, with each Use-Case owner, the type and relevance of its requirements before validation (or adaptation).

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

4 Usage of the IoT Stack platform

4.1 Authentication and Authorization

The IoT Stack uses Keycloak (by RedHat) as an Identity Server. It implements the latest User-Managed Access specification (UMA 2.0) which essentially is an extension of the OAuth 2.0 specification with a new grant type. It allows for complex sharing situations between different resources owners and actors on behalf of owners.

Authentication must be seen as the act of logging in; nothing more than that, a guarantee that one is who he/she claims to be. For authentication the IoT Stack relies on Keycloak as an OIDC Provider broker. This means that the IoT Stack is not storing any passwords, just linking user accounts with harmless identifiers received from existing and well-known authentication services[13]. When logging in to the API, the user is able to select one of the supported identity providers to log in to, and afterwards is redirected back to Keycloak (and eventually the application that initiated the flow)[14].

Authorization is checking whether the authenticated person/application/device is allowed to view or act on the resources requested. This is done on two levels. The first level will deny requests if one tries to access an endpoint he/she doesn't have access to. The second level denies accessing information one is not allowed to view (although he/she may technically access the endpoint).

Before any authorization (read: allowed use of the APIs) can happen, one must thus be first authenticated (read: he/she is well who he/she claims to be). This results in an access token representing the client, i.e. a device or an application (and possibly as well the user using the application).

The client (device/application) will be registered as an application acting on its own (or on behalf of users) and receives from the IoT Stack a client_id and a client_secret (only a client_id if application acting on behalf of the user). These are called the client credentials and will be needed at some point in the authentication process.

For authenticating either the client as itself (a device or an application), or the client on behalf of the user, one of the two OIDC flows must be used. Depending on the flow, the authentication process might go slightly different[15].

The implicit flow[16] lets a client application or device immediately receive an access token to access the authorization APIs. It does so at the expense of sending the tokens in the redirect_uri. Because these clients typically can't keep a secret from the user agent or other processes on the client host,

[13] For the time being Google is the only supported OIDC provider. In the future more providers might be added, the focus being at this time however to first get the most important features working.

[14] Logging in at the identity provider's own website, guarantees that the IoT Stack is never able to read or get the user's password.

[15] There are indeed two main flows of authentication: Implicit flow and Authorization code flow. While the former is the simplest, the latter is the most secure. However in the case of browser applications Authorization code flows can't be guaranteed to be secure, that's why the implicit flow exists.

[16] There are not a lot of cases where implicit is a good fit. There is no refresh token, meaning that the complete auth procedure must be executed again once the token expires. That is why this is typically suited for a one-off task.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

these client themselves are not authenticated (just identified by a client_id). This flow is typical for browsers.

Authorization code flow[17] makes a client application or device take an extra indirection step, making sure the token can't be easily intercepted by different attacks. After initial authentication, the client receives an access code (rather than an access token) that cannot immediately be used to access the authorization APIs. This code must then be sent to the token endpoint (`https://obelisk.ilabt.imec.be/auth/realms/idlab-iot/protocol/openid-connect/token`) to receive the actual access_token. This must be done with an HTTP POST request, not with a redirect. The response will contain an access_token and a refresh_token. The sections hereafter further describe the steps required to complete the auth process and end up with an RPT token that can be used to access the APIs.

4.1.1 Authentication to the identity server

There are two different ways to authenticate: either the client is logging in on behalf of the user, or the client/service authenticates as itself.

4.1.1.1 On behalf of the user

The client wants to identify the user along with itself to the API. In this case the client is an application that is meant to be used by users as a tool that inherits their rights (to which they agree) [18]; to this end, the two main steps to be implemented are described hereafter.

a) **Redirect to user login:** This can be done through a link to the following url: `https://obelisk.ilabt.imec.be/auth/realms/idlab-iot/protocol/openid-connect/auth?parameters...`. The parameters are described in the table hereafter:

Parameter	values	description
client_id	-	This is the client_id from the client credentials received when registering the client application.
redirect_uri	-	This is the uri where the browser will be redirected to, once the login procedure is over.
State	-	Opaque value used to maintain state between the request and the callback. Typically, Cross-Site Request Forgery (CSRF, XSRF) mitigation is done by cryptographically binding the value of this parameter with a browser cookie.

[17] The client library uses this flow by default and calls it the 'standard' flow.

[18] The client library can do most of the heavy lifting for authentication/authorization, in case of a client-side JS/TS application (see: https://obelisk.ilabt.imec.be/api/v1/docs/api/client-lib/). If you need to do this manually, follow the defined steps.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

Nonce	-	A cryptographic string, that will be represented to the client in the requested token, to mitigate replay attacks[19].
response_mode	**fragment** query form_post	How the token will be returned to the *redirect_uri* (fragment is highly recommended). In case of response_mode fragment the redirect uri will contain a hash fragment that contains a token.
response_type	(see table hereafter)	Which OAuth 2.0 flow is being used. Public clients should use implicit.

response_type	OAuth flow	Description
Code	Authorization code flow	Response will contain a code that should be exchanged for a token at the token endpoint (client should be able to keep code a secret from browser/other code, e.g. a server backend that can do a back-channel request).
id_token	Implicit flow	Response will contain an id_token immediately (when clients can't keep a code secret, this is the go to flow).
id_token token	Implicit flow	Response will contain an id_token and access_token immediately (when clients can't keep a code secret, this is the go to flow). Tip: start with id_token token as this is the simplest authN flow.

b) **Retrieve code/token from hash:** In case of implicit flow, the access_token and possibly the id_token are in the redirect_uri hash and there is no need for an extra POST request. The id_token is for user customization in the client and of no importance for the IoT Stack anymore. The format is JWT. The access_token will be used in section 4.1.2.

In case of Authorization code flow (recommended), at the redirect_uri location there should be a script that is able to extract the code from the hash (in case of use of response_mode fragment). After the login procedure is over, the redirect_uri will be called and in the hash fragment there will be a parameter code. This value must be used to do a POST request to `https://obelisk.ilabt.imec.be/auth/realms/idlab-iot/protocol/openid-connect/token` with the following headers:

Header key	Header value
Content-Type	application/x-www-form-urlencoded

[19] More info available at: https://en.wikipedia.org/wiki/Cryptographic_nonce and https://auth0.com/docs/api-auth/tutorials/nonce .

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

… and body (concatenated as form data param=value&param2=value2):

Form param key	Form param value
grant_type	authorization_code
code	code_value
redirect_uri	https://my-example.app.com
client_id	client_id

The received JSON response contains access_token, id_token and refresh_token (for the access_token).

4.1.1.2 The client as itself

To authenticate the client application or service, a valid client_id and client_secret pair are necessary[20]. The client credentials are encoded in an authString `Base64Encode(client_id:client_secret)`. This authString is then used to do a POST request to `https://obelisk.ilabt.imec.be/auth/realms/idlab-iot/protocol/openid-connect/token` with the following headers:

Header key	Header value
Content-Type	application/x-www-form-urlencoded
Authorization	Basic authString

… and body (concatenated as form data `param=value&param2=value2`):

Form param key	Form param value
grant_type	client_credentials

The received JSON response contains access_token and refresh_token (for the access_token).

4.1.2 Getting the RPT token

The final step is to get the RPT, which is the actual access token needed to talk to the IoT Stack APIs. For this a HTTP POST request must be sent to this url: `https://obelisk.ilabt.imec.be/auth/realms/idlab-iot/protocol/openid-connect/token`.

[20] For first request, see : https://obelisk.ilabt.imec.be/api/v1/docs/getting-started/request-access/

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

The request will contains these headers:

Header key	Header value	Description
Authorization	Bearer access_token	The access token received in previous step prefixed with Bearer.
Content-Type	application/x-www-form-urlencoded	Form format to send two important form parameters in the body.

… and these required form parameters:

Form param key	Form param value
grant_type	urn:ietf:params:oauth:grant-type:uma-ticket
audience	policy-enforcer
Scope *(optional)*	offline_access[21]

The IoT Stack will return a message that looks like this:

```
{
    "upgraded": false,
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiA...",
    "expires_in": 300,
    "refresh_expires_in": 1800,
    "refresh_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSl...",
    "token_type": "Bearer",
    "not-before-policy": 0
}
```

The access_token field is the RPT token. This is what must be used for every request to the API. The refresh_token can be used to quickly refresh an expired access_token. The details are explained in section 4.1.4.

4.1.3 Call a resource

Now a resource on the API can be called by including this header: `Authorization: Bearer RPT`.

[21] For more info on offline_access, see : https://obelisk.ilabt.imec.be/api/v1/docs/security/offline_access/

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

4.1.4 Refreshing the RPT

When getting an `ERROR 401` response on a correct API resource call (as defined in section 4.1.3), it probably means that the RPT token expired. In section 4.1.2 there was an expires_in field, which can be used to timely refresh the token. Refreshing the token is done by sending a HTTP POST request to this url: `https://obelisk.ilabt.imec.be/auth/realms/idlab-iot/protocol/openid-connect/token`.

It should have the following header: `Content-Type: application/x-www-form-urlencoded` and contain the following form parameters:

Form param key	Form param value
grant_type	refresh_token
refresh_token	refresh_token (received in 4.1.2)
client_id	cliend_id

The IoT Stack will return a message that looks like this

```
{
    "upgraded": false,
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiA...",
    "expires_in": 300,
    "refresh_expires_in": 1800,
    "refresh_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSl...",
    "token_type": "Bearer",
    "not-before-policy": 0
}
```

The access_token field is the new RPT token. This is what will be used for every new request to the API. The refresh_token is also important, it has a new expiry date and should replace the previous refresh_token to be able to refresh continuously.

4.2 How to push data through the REST API?

A generic ingest API is provided for devices and applications that need to push data to the IoT Stack. It is used by devices that need to push new data to the IoT Stack by providing the scope to which they 'belong', their position as well as the metrics (associated to things) and their respective values. This API is defined within the Swagger UI and illustrated in Figure 17 hereafter.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

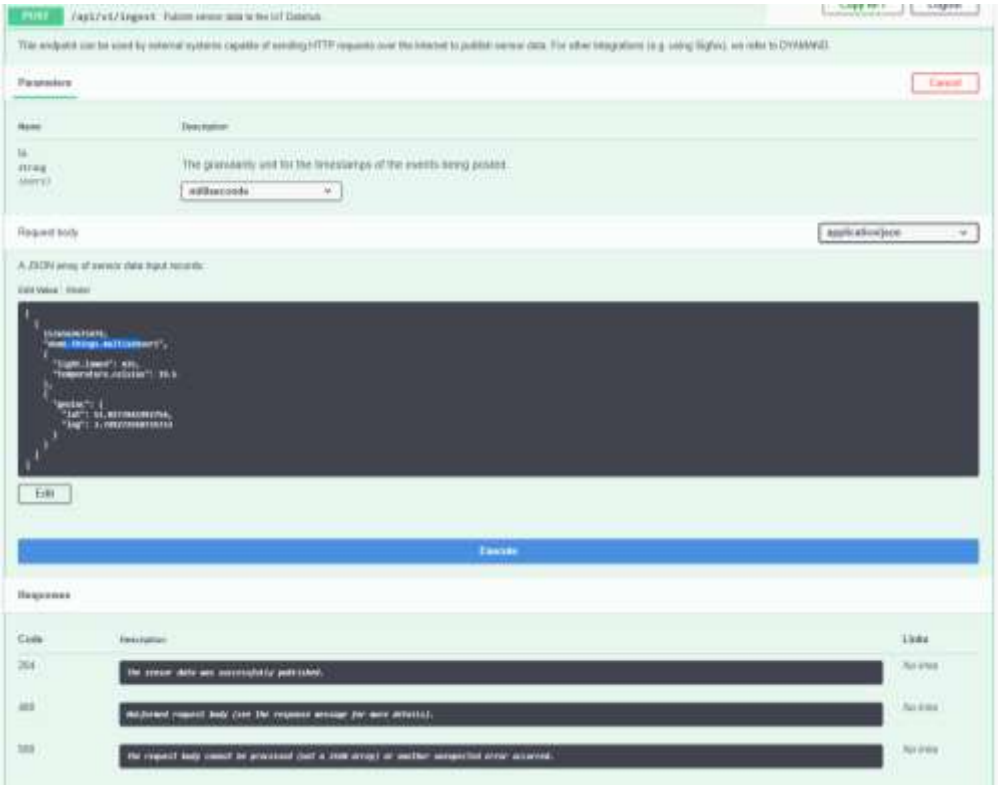

Figure 17 : Screenshot from ingest API as defined in the Swagger UI

When needed, a specific API for video ingestion can also be provided: however it is not yet documented within Swagger UI as it still is being prototyped at the time of editing this document.

4.3 How to retrieve data?

There are mainly two different ways for a client application to retrieve data from the IoT Stack:

a) By the use of queries: this mode should be preferred for applications needing to query (historical) data when required (e.g. at end-user click) or for batch oriented applications. It is mainly composed of three categories:
   1. Scope Metadata Operations: Operations for accessing Scope metadata (e.g.: What is measured? Who has access?)
   2. Scope Thing Query Operations: Operations for querying raw or derived metric data for a single Thing.
   3. Scope Location Query Operations: Operations for querying raw or derived metric data for a Location.

b) By using Server-Sent-Events (SSE): this mode should be preferred for applications needing 'near real-time' notifications from the IoT Stack each time a new event is triggered. It opens a Server-Sent-Events stream to receive push events for the scope. This mode may be configured as to keep or not keep a local copy (in the IoT Stack) of historical data; in both cases however data is buffered to allow replay in case of client temporary disconnection. It has been designed for data stream oriented use-cases.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

5 Integration of IoT Stack through Virtual Fort Knox

Introduction and Integration of cyber-physical systems or Internet of Things (IoT) in common lead to disruptive changes and increasing complexity for IT solutions in the field of production and logistics. The use of service oriented architectures can be one possible solution to increase stability, security, reliability, traceability and flexibility. However, due to the lack of security requirements, they are not applied.

The Virtual Fort Knox platform ('VFK' hereafter) offers manufacturing and logistics companies a Private Cloud solution within a secure and scalable cloud environment. It allows the integration and connection between different systems like IT infrastructures, Cloud Platforms, cyber-physical systems and digital services by providing a homogenous, service based integration layer called "Manufacturing Service Bus" ('MSB' hereafter). The VFK platform together with the MSB can operate as a central interface for the connection and integration of different cyber-physical systems and services which is shown on Figure 18.

The following description shows how it is possible to use the VFK platform within the PortForward project to operate services and integrate different systems (sensors, actors, cyber-physical systems, 3rd party port management systems etc.). It gives a high level overview how to deploy and operate services which can be connected to the IoT Stack of IMEC described earlier in this document, too. A detailed description how to implement a service running on VFK platform will be given in Deliverable D3.3.

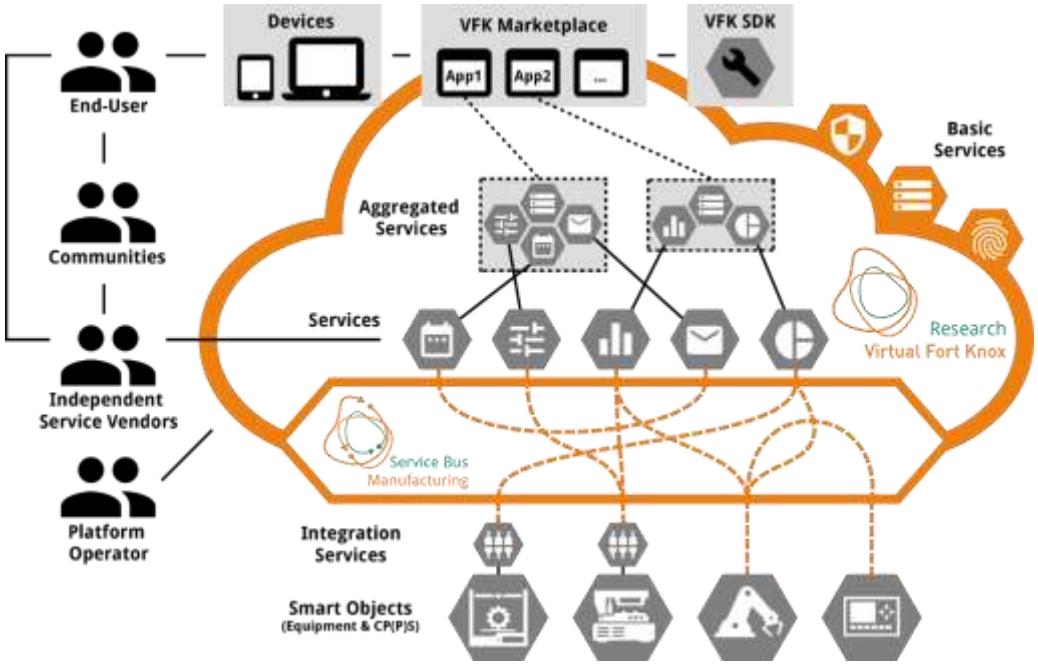

Figure 18: Virtual Fort Knox Components and Roles (Source: Virtual Fort Knox Research)

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

5.1 Service Deployment

VFK offers different ways to develop, aggregate and deploy an own service. The first option is the use of Cloud Service Automation together with Operations Orchestration using OO Flows[22 & 23] what could lead to slightly higher expenses in porting or migrating existing services and currently limits the usable technologies for implementation to Java or .NET.

The second option is the use of Docker by containerizing existing or new services or applications and deploy them to the VFK platform. The only restrictions on technology level are based on restrictions of Docker itself as well as some current restrictions by using Docker on VFK:

- Services may include a configuration directory to store persistent data like databases and configuration files.
- Services currently may only contain one Docker Container. Docker-Compose is not supported yet.
- Services are restricted to one IP Network Port (customizable) – this limitation will be changed in further releases

It is possible to use (nearly) any Application Environment and runtime which only has the restriction that it should be containerized into a Docker container.

Deploying services based on Docker Images allows the reuse of existing services or applications already deployed based on Docker or the easy „containerization“ of existing applications and services. The VFK platform offers all necessary basic components like the management and runtime environment as well as a Docker Host where the Docker containers are instantiated (see Figure 19).

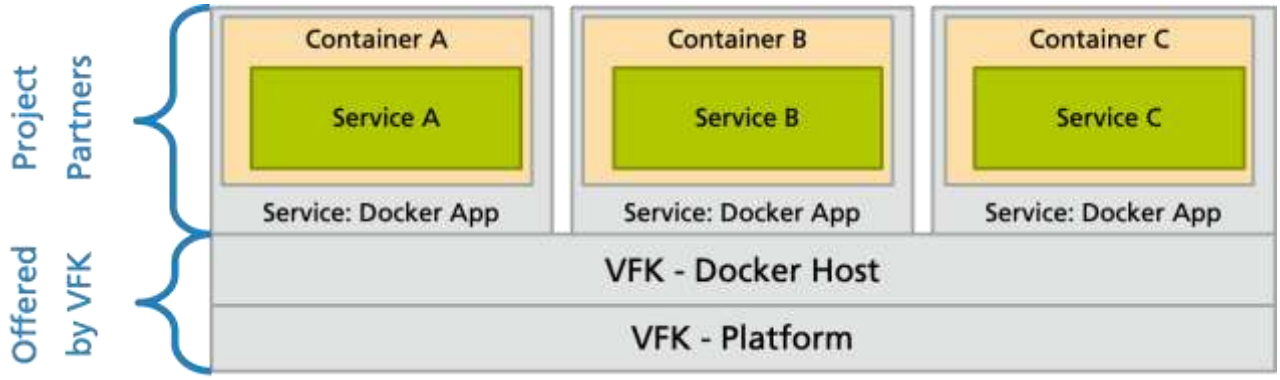

Figure 19: Service deployment on Virtual Fort Knox by using Docker

To deploy a service as a Docker container it is necessary to provide a Docker container configuration and create a service offering in VFK (a service offering is like a template for an application). This service offering can be used to create an instance of the service which can be accessed via the provided IP Address and Port (request and response pattern) or through connecting the service to the MSB (publish-subscribe pattern by collecting and redistributing data via a message broker).

[22] https://www.microfocus.com/en-us/products/operations-orchestration-it-process-automation/overview

[23] http://www.hp.com/hpinfo/newsroom/press_kits/2009/HPSoftwareUniverseHamburg09/HPOODataSheet.pdf

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

5.2 Services types

According to Figure 18, Services running on the VFK platform can be separated into three different types:

- Integration Services
- Services
- Aggregated Services

All three types of services can be deployed based on OO Flows or by the use of Docker images. These three types of service differ in what functionality they implement and how they receive, process, and deliver data.

Integration Services

Integration Services can be used to connect third party systems like sensors, actors, cyber-physical systems in general or the IoT Stack provided by IMEC to exchange data between services and third party systems by using the MSB. The advantage here is obvious: for connecting third-party systems, only one integration service has to be developed, and all other services can access this data via the MSB or make data available.

Basic Services

Basic Services can be used to implement specific applications or functions for transforming, calculating or storing any data. However, the granularity of services should be limited to specific functions or tasks and not represent complex applications. Additionally integrating third party systems is not only limited to Integration Services. It is possible to integrate third party systems by the use of Services without using the MSB.

Aggregated Services

Aggregated Services are used for the implementation of complex functionality or applications like representations of Digital/Virtual Twins, Port Management Applications or Dashboards by integrating or combining different services. These services are mainly published to end users in line with their respective business processes.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

5.3 Data Flow

Exchanging data between services at the VFK platform and third party systems like the IoT Stack of IMEC can be realized via two options which is shown in Figure 20:

1. Pull Request and Response
2. Push

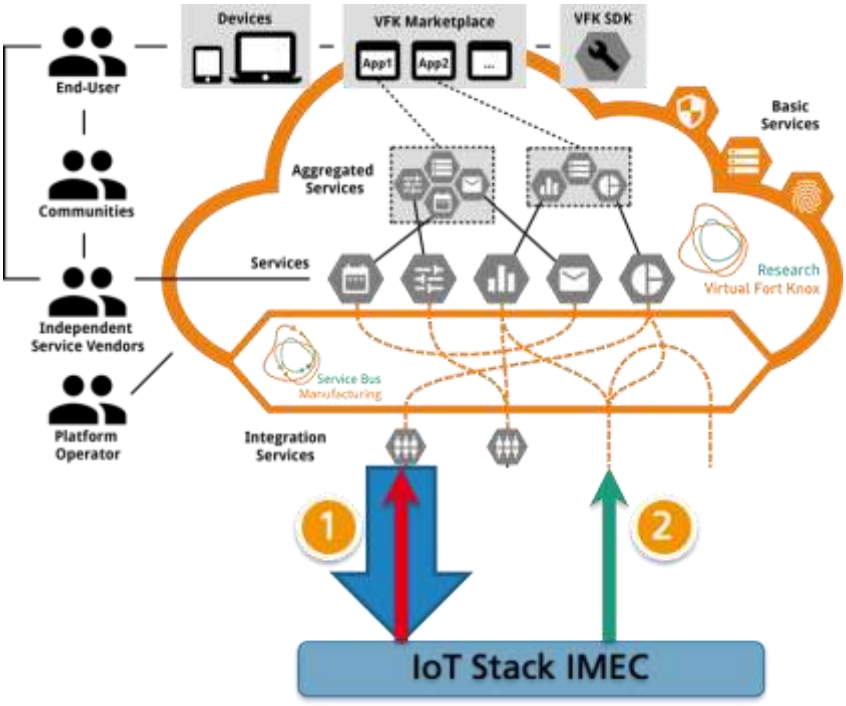

Figure 20: Data Flow via Pull/Response (1) or Push of Data (2)

Pull Request and Response

An integration service can be used to connect to the IoT Stack and directly request data from a given endpoint. This option allows to request specific or filtered data and is mainly used to access additional or raw data for further analysis. The pull request can be used to send data to the IoT Stack like configuration parameters, too.

Pushing Data

As the IoT Stack integrates a lot of sensors or any third party systems which mainly create data based on events it generates a flow of data. To receive all generated data by using the pull request it is necessary to implement a polling mechanism requesting an API endpoint in a certain interval which may lead to unnecessary requests if no new data has been generated or to high loads if the interval is defined too big or too much data is being generated. A much more efficient option is to use a push mechanism where the generated data of an event is being transferred as soon as the data has been created. The IoT Stack can directly send (or push) data to the MSB where all connected services will receive the data immediately.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

5.4 Use Case example

Initial situation

Containers shall be tracked and identified by the use of modern tracking (GPS and mobile network based) and identification (e.g. RFID) technologies. As there exist a lot of different systems providing tracking and identification functionalities but offering different types of APIs and data formats it is very difficult to integrate tracking and identification solutions into own service.

Solution

Figure 21 shows an example how to implement Services on the VFK platform combined with the IoT Stack followed by a short description of the different types of services used to realize this use case example.

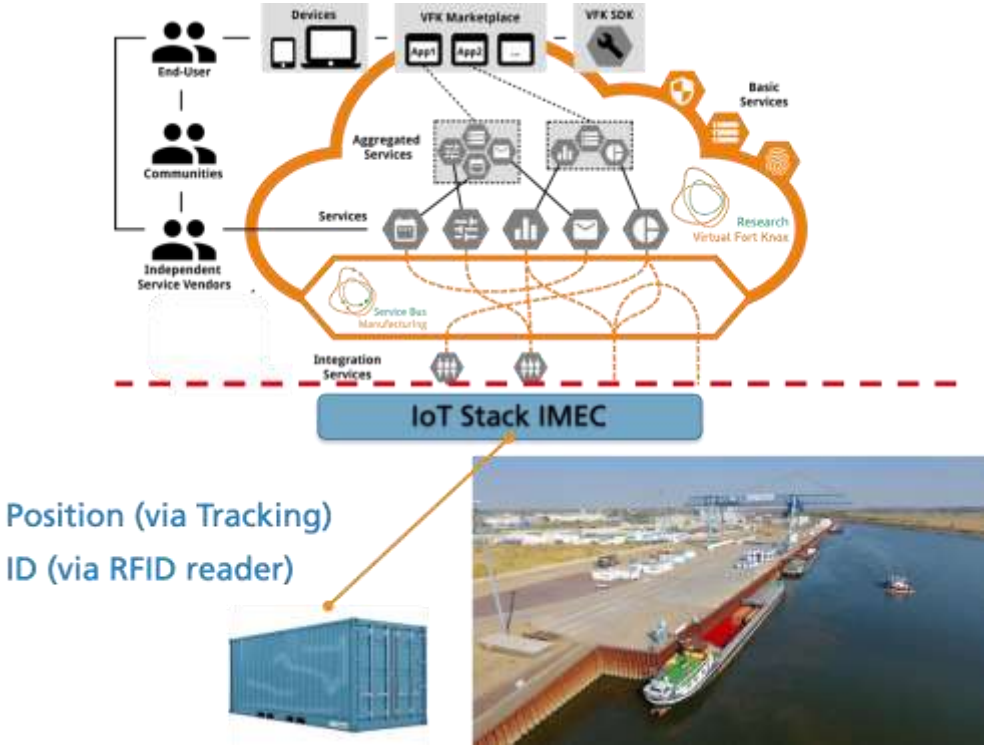

Figure 21: Possible Solution for integrating Tracking and Identification Technologies together with the IoT Stack with services on VFK

The connection between services and solutions providing position and identification information can be realized by the use of systems like the IoT Stack provided by IMEC or any other provider specific middleware communicating with sensors or tracking devices. These components provide a technology and system independent integration into own systems with less effort than implementing each protocol for each tracking or identification solution (Integration Services).

Basic Services can be used for pre-processing, evaluation and analysis of data. These small services can implement functions like storing positions, events (moving, reading of a RFID tag), analyzing position data (e.g. Geofencing).

Based on these Basic Services aggregated services like Storage Management can be implemented offering much more complex functionalities supporting the business processes inside a port.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.

6 Conclusion

This document aimed at providing introductive guidelines on the cloud based IoT integration layer (the so called IoT Stack, provided by IMEC) to technical partners of the PortForward project's consortium. It contextualized the rationale of the project's layered architecture where the IoT Stack plays a central role, before providing technical insights on the main features offered by the platform, illustrated when possible by a demo use-case.

Located in the middle of the data stream flowing from the IoT devices to the software applications and services that will be developed for PortForward, a specific focus has been put on the data ingestion and retrieval features together with their associated security mechanism. For the same reason, the VFK platform (provided by IFF) that will host the PortForward use-cases applications and services has been briefly introduced, again mainly in an end-to-end data flow perspective.

As a matter of fact the platform will continue to evolve all along the project lifetime, therefore the commitment taken by IMEC to accompany the partners in the realization of the PortForward use-cases with on-time announcement, updated documentation and the support they may require.

This project has received funding from the
European Union's Horizon 2020 research and innovation program under grant agreement No 769267.