TDM Edge Gateway: a flexible microservice-based edge gateway architecture for heterogeneous sensors*

Massimo Gaggero$^{1[0000-0003-3309-5854]}$, Giovanni Busonera$^{1[0000-0003-3266-7857]}$, Luca Pireddu$^{1[0000-0002-4663-5613]}$, and Gianluigi Zanetti$^{1[0000-0003-1683-7350]}$

CRS4 - Center For Advanced Studies, Research And Development In Sardinia, Loc. Piscina Manna, Edificio 1, 09050 Pula (Ca), Italy
{massimo.gaggero,giovanni.busonera,luca.pireddu,gianluigi.zanetti}@crs4.it
http://www.crs4.it

Abstract. How to effectively handle heterogeneous data sources is one of the main challenges in the design of large-scale research computing platforms to collect, analyze and integrate data from IoT sensors. The platform must seamlessly support the integration of myriads of data formats and communication protocols, many being introduced after the platform has been deployed. Edge gateways present a solution to this issue. These devices, deployed at the *edge* of the network, near the sensors, communicate with measurement stations using their proper protocol, receive and translate the messages to a standardized format, and forward the data to the processing platform. Edge Gateways can also support applications with low remote connectivity guarantees through local data buffering and preprocessing. In this work we present the *TDM Edge Gateway* architecture, which we have developed in a concrete problem scenario – in the context of the "Tessuto Digitale Metropolitano" research project – to meet the requirements of being self-built, low-cost, and compatible with current or future connected sensors. The architecture is based on a microservice-oriented design implemented with software containerization and leverages publish/subscribe Inter Process Communication to ensure a modularity and resiliency. Costs and construction simplicity are ensured by adopting the popular Raspberry Pi Single Board Computer.

Keywords: Edge computing · Smart cities · Iot · Sensor networks · MQTT · Publish Subscribe · Embedded · FIWARE.

* This work was supported by the TDM project funded by Sardinian Regional Authorities under grant agreement POR FESR 2014-2020 Azione 1.2 (D. 66/14 13.12.2016 S3-ICT).

1 Introduction

In recent years the spread of internet-connected devices has proven to be an important and bountiful source of data for the development of new services in different application domains in industry and research. However, research applications often suffer from the frequent use of closed and proprietary components and the impossibility to freely export the acquired data. At the same time, there are examples of research and non-profit projects that have successfully created large sensor networks supported by volunteers that build, install and operate sensor stations on their own premises [23]. While volunteer project participants cover deployment and maintenance costs, they receive in return the results and services provided by the research – such as environmental monitoring, weather and power-consumption forecasting, visualization tools and time-series statistics. The adoption is favoured by low costs and ease of customization and programming, factors that are motivating the adoption of microcontroller-based development board as hardware platforms for such use cases. However, using these kinds of devices poses important issues to the data collecting infrastructure from the point of view of security, reliability and, especially, device heterogeneity.

In this paper we describe a novel architecture of an Edge Device that is designed to effectively address the issue of device heterogeneity in distributed data acquisition networks for research. Its primary purpose is to translate sensor-specific data to a standard FIWARE-compliant data model [11] and form bridge between the sensor's native communication protocols and the computing platform. Different application scenarios are supported:

- edge device with no user interaction for data acquisition in buildings, plants and farmlands;
- edge station in private home with local data viewing and possible interaction with central facility;
- edge network gateway for wireless broadband communication in areas where no cable broadband link are available;
- mobile edge gateway that can provide store-and-forward capabilities for resource constrained sensors.

Primary requirements for this Edge Gateway are *flexibility* and *robustness*. The Edge Gateway must guarantee seamless integration of future sensors and applications without affecting those already present. The adoption of a microservice-oriented approach provides the required modularity [9]: the various functionality provided by the device is distributed in different separate micro applications, each only implementing one service. Microservices are delivered by means of *lightweight software containers* packaged with all the required files and libraries. Microservices and containerization also provide robustness: a broken module does not directly affect the whole system or taint others executable context, while incompatible libraries are kept separated among different containers (the executable environment is insulated) [19]. Another point of robustness in our design is represented by the use of the *publish/subscribe* [27] paradigm for data exchange between microservices. This asynchronous communication pattern allows the different applications to work even if there are no sender or receiver modules alive. Again, module failures are not propagated by means of socket errors, timeouts or service unavailability. At the same time, the addition or removal of microservices does not require a system reboot or a message-routing reconfiguration.

This Edge Gateway has been developed, tested and its deployment in research context is currently underway withing the context of the "Tessuto Digitale Metropolitano"[1] (TDM) research project, which aims to develop innovative applications for Smart Cities in the fields of Weather Safety and Nowcasting, Citizen Energy Awareness, and Large Dataset Visualization for Cultural Heritage. These vertical applications depend on a mixture of data from different sources – from satellites to meteorological radar, and particularly to weather, air and energetic sensors distributed over the Metropolitan Area of Cagliari. To facilitate the recruitment of volunteer sensor deployers to the project, the sensors platform was chosen to be low-cost, ready-made and available, open source and open hardware. Moreover, given that all the developed code, documentation and Reference Designs have been made available to the public as open source, the creation and adoption of new sensors and the interest of diverse research fields are encouraged too. Thus, the Edge Gateway must bridge a multitude of sensor formats and protocols to our Lambda-architecture platform for storage and processing. One distinctive feature of TDM Edge Gateway is that it is natively *FIWARE compliant.* FIWARE is "*a curated framework of Open Source platform components to accelerate the development of smart solutions*". It provides a common set of Open Source APIs for data exchange, the *Next Generation Service Interface*, software components for IoT and Big Data integration, and a large set of *harmonized data models.* Given the portability and interoperability they provide to the infrastructure for Smart Cities and Smart Solutions, the FIWARE NGSI API has also been adopted by the Open and Agile Smart Cities (OASC) network for the integration of services among the cities taking part in the initiative [12].

The rest of the paper is structured as follows. In Section 2 we describe the context of self-built sensors and research projects that rely on them. In Section 3 we describe the architecture of our Edge Gateway and its components, while the developed system, its testbed and deployment is explained in Section 4. Section 5 discusses the state of the art of this specific research field and Section 6 concludes the manuscript summarizing its contribution and proposing future developments.

2 Low-Cost, Self-Built Smart Sensors

Online development platforms for open source projects – e.g., *GitHub*[2] – host a myriad of projects for weather stations and metering devices based on boards like Arduino, NodeMCU [21] and Raspberry Pi. The WiFi network capabilities and

[1] http://www.tdm-project.it/

[2] `https://github.com/open-source`

the availability of a large set of open source communication libraries allow users to interact with the devices with nothing more than a smartphone and a web browser, while the acquired data can be forwarded to remote cloud infrastructures for further storage and analysis. There are several examples highlighting the possibilities offered by these platforms. For instance, Luftdaten project [23] was started in the OK lab Stuttgart with the goal of monitoring air quality in the entire city through a network of user-volunteered devices. The project provides detailed instructions on how to self-build a compatible measurement station and join the project. The device's design is based on the low-cost NodeMCU board. The core is the widely used WiFi-capable ESP8266 microcontroller and the open source firmware provides supports a large set of sensors like fine dust, temperature, humidity and barometric pressure. The user-friendliness and low-cost of their design has a enabled voluntary participation to the project to reach excellent coverage over much of central Europe and show various outposts across the globe, from North America to Australia. Another example arises from applications in agriculture, where such low-cost devices have been used for field monitoring and food chain tracking [26]. The work points out the importance and challenges of Wireless Sensor Network (WSN) in farmlands and greenhouses and the need for the cloud or fog computing paradigm to address the data analysis phase. A survey of WSN node platforms, wireless protocols and typical hardware requirements for applications in agriculture is presented as well as some real-world deployment issues as the impact of temperature and humidity to the node communications. Moreover, the paper discusses the problem of security and interoperability at different levels. Another example are smart home controllers for home environmental comfort. They use temperature, humidity and luminosity sensors to control HVAC systems or curtains but their data, relayed to a Cloud platform, can be also used for research on environment and energy efficiency [1]. In [25] is introduced a distributed low cost wireless sensor network (WSN) used as data center temperature monitoring system. Each node of the sensing system is composed by two temperature sensors connected to a ESP8266 that acts both as the DHT11 interface and as the wireless communication unit. Moreover, the ESP8266 performs some preprocessing jobs before sending data to an external cloud facility. From the point of view of Electrical Metering, the IotaWatt [16] is a device that monitors up to 15 channels. One is reserved for voltage measures whereas the remaining 14 can be used to acquire voltage or power measures. The latter are actually obtained by using a current transformer passive sensor. The IoTaWatt is based on the ESP8266 and is able to connect to a local WiFi for logging data to an external server. The capability of these sensor to send the data to remote Clouds and their spread in houses and public buildings (many of these sensors are used for education purposes) makes these device very attractive for research purposes in fields like Urban Computing, Weather Forecasting and Now-casting, agriculture [26] and citizen Energy Awareness.

2.1 Heterogeneity issues

Connected sensors represent a large source of information for Big Data analysis, due to the high numbers of installed devices, and due to the possibility of cover large areas with a high resolution. At the same time they represent for Big Data platforms a source of added complexity. Scalability, fast data access and real-time computation (i.e. in Lambda Architectures) usually require a common message format for data received from remote sensors. This in turn involves standardization of the devices used as measurement stations that means the adoption of same vendor, model and type devices or a customization of their firmware, where possible. This constitutes an access barrier for citizens, hobbyists and students who wants to join research projects, reducing the potential number of the sensors. On the other hand, to implement specialized endpoints for the different message formats increases the complexity and the costs of the collecting infrastructure, reduces the range of supported devices and inherently leads to another access barrier. A solution to the "standardize vs specialize" problem is represented by the insertion at the edge of the network and between sensors and the cloud of a *bridging* or *gateway* device. Edge Gateways are devices provided of enough computation and storage resources to intercept, collect and translate the messages received from the local sensors and forward them to the cloud facility in a common format using a common network protocol. This allows to address IoT heterogeneity issues and overcome some common sensor resource restrictions like lack of temporary storage, data pre-processing, compression, encryption and strong authentication, ability to cope with network disruption. In addition, Edge Gateways can host more resource intensive services like database, data visualization tools and data processing.

However, the introduction of the Edge Gateway in the architecture is a cost in terms of development, maintenance and added hardware. Nevertheless, a proper choice of the hardware platform can limit the additional costs of the Edge Gateway. ARM-based Single Board Computers like those of the Raspberry Pi have proven to have enough processing, memory and storage resources to be used as low-cost, energy-efficient computation nodes [14][5] and they are readily available on the marketplace. Costs in complexity can be reduced while reaching the goal of flexibility adopting for the software architecture a modular approach and technologies that allow that like lightweight virtualization, insulation and asynchronous communications.

3 TDM Edge Gateway Architecture

The TDM Edge Gateway follows a microservice-based design. The functionality it provides is distributed across separate micro-applications deployed as Docker [7] containers on the edge device. Data exchange between microservices is performed asynchronously through the publish/subscribe MQTT [22] protocol. MQTT is also used to forward data from the Edge Gateway to the acquisition and processing platform using an encrypted and authenticated channel. In addition, data is also written to a local InfluxDB instance. From here it can be queried by applications running directly on the Edge Gateway, such as the dashboard.

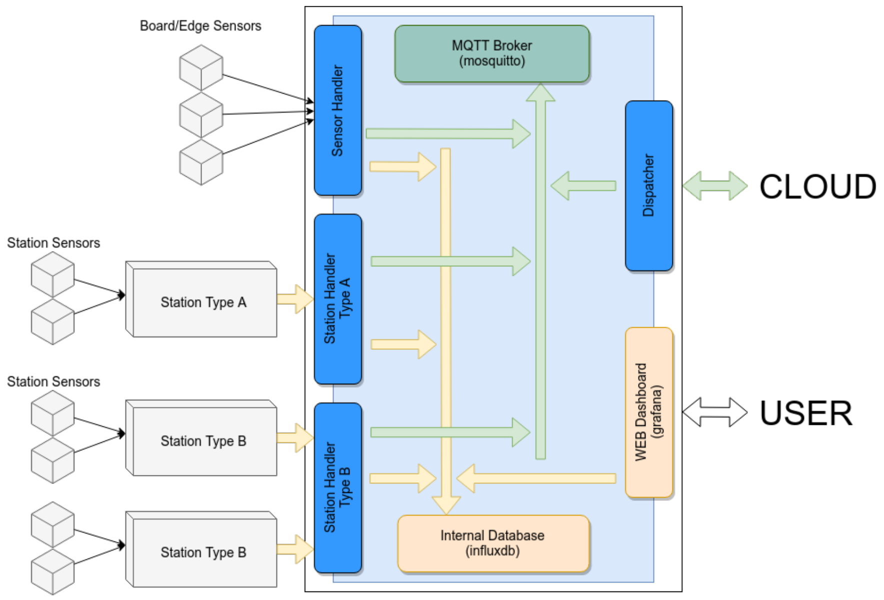


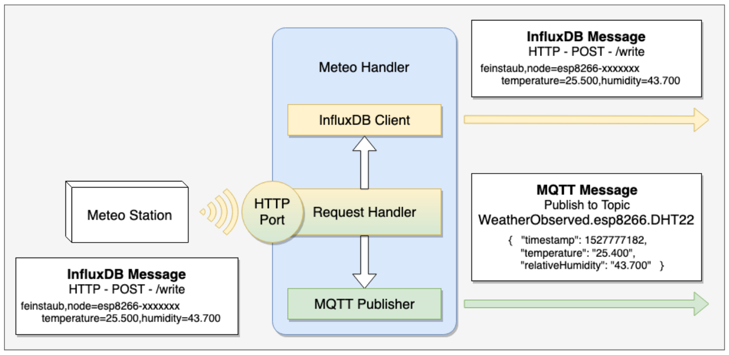
Fig. 1: Edge Gateway Architecture

TDM Edge Gateway microservices are divided into three roles: *handlers*, *consumer* or *dispatcher* and *ancillary services*. These are described in the following paragraphs.

Handlers. Handlers are the microservices that receive, translate and store the data that arrives from sensors and stations. Handlers are specialized for each type of sensor or station – rather than having a single service that deals with the heterogeneity of transmission protocols and message formats. Sensors and sensor stations may be remote, transmitting data via network, or they may be directly connected to the Edge Gateway. The handler's main tasks are to:

– establish and maintain communications with the sensor;
– receive and write data to the local InfluxDB database;
– translate the sensor message from the native to the cloud format;
– publishing the translated messages to the local MQTT broker.

By publishing the data on the MQTT bus, the handlers make the data available to the any other subscribed microservices on the Edge Gateway, making easy to insert additional functionality. Handler services are further divided in *sensor handlers*, which deal with sensors directly attached to the Edge Gateway, and *station handlers*, which deal with remote stations equipped with one or many sensors (i.e. weather stations with sensors for wind, rain and atmospheric particulate). Edge Gateways can easily support multiple stations (i.e. stations in different rooms or apartment in a building). They are data producers and MQTT publishers.



```
feinstaub,node=esp8266-xxxxxxx
    temperature=25.500,humidity=43.700
```

```
{  "timestamp": 1527777182,
   "temperature": "25.400",
   "relativeHumidity": "43.700"  }
```

```
feinstaub,node=esp8266-xxxxxxx
    temperature=25.500,humidity=43.700
```

Fig. 2: Data Flow in Station Handler

Dispatcher. The *dispatcher* forwards all messages passing through the MQTT broker to the data processing platform. It is a real-time consumer of the all the data published on the MQTT bus – since it is subscribed to all the MQTT topics. The dispatcher is also a publisher to the MQTT broker on the data processing platform.

Ancillary Services. A number of ancillary services run on the Edge Gateway to provide supporting functionality for application microservices and for user interaction. The *Mosquitto* [10] Open Source lightweight MQTT broker provides the internal publish/subscribe MQTT bus. Indeed, Mosquitto is a core component of the system as it provides the flexible and scalable interconnection between message handlers and dispatchers. Second, the Open Source time series database *Influxdb* [15] is used to locally store data received from sensors and stations connected to the Edge Gateway. While real-time data can be retrieved by other services on the edge device using the MQTT internal bus, services that require deferred access or that need to process data in batches can query the Influxdb database. The final ancillary service, the *Grafana* [13] Open Source web-based dashboard, is an example of this use case. It is provided to visualize data stored in the local Influxdb database – potentially as customized graphs, gauges and entire dashboards – and also to set-up alarms.

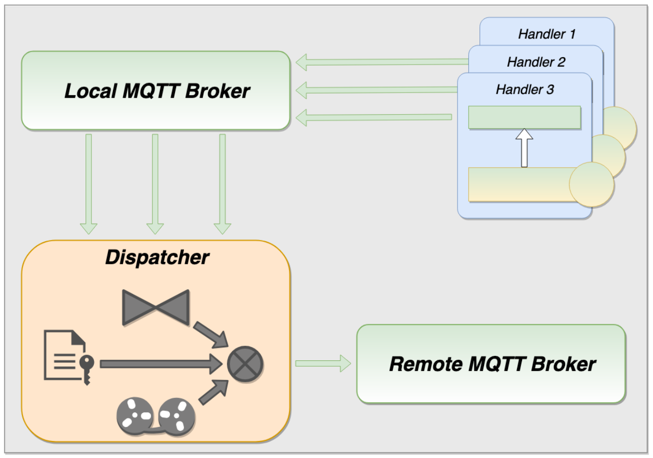


Fig. 3: Data Flow in Dispatcher

3.1 Software Containers

Individual microservices are run on the Edge Gateway in separate Docker containers. Deploying these components as containers enables us to better leverage the microservice architecture, allowing each one to be independently developed, updated, or deployed while providing better protection against compromising the functionality of other microservices and of the system as a whole. The adoption of software containers for our use case does not impose any significant overhead in terms of computing resources [4, 6], particularly since the services are long-running. Using Docker container images does impose a slight storage overhead as compared to native software installation due to the write-only layered image storage approach – which can easily result in some degree of file duplication – but that effect is controlled by compiling the images carefully. In addition to minimizing storage requirements, minimizing container image sizes also improves container download and start-up time [8], thus accelerating software updates. In our Edge Gateway, the container-based services are composed with the *Docker Compose* tool. It creates and manages the entire deployment consisting of the container-based microservices, an isolated internal virtual network to connected them, the external service endpoints, and persistent volumes. It also automatically downloaded the containers images if they are not already cached locally on the Edge Gateway.

4 Implementation

4.1 Edge Gateway Hardware and Operating System

The hardware specifications Raspberry Pi Ver. 3 Model B+ (RPi) used for our implementation are outlined in table 1. The RPi runs *Arch Linux for Raspberry*

Pi 3 [3]. We customized the default OS image to simplify its installation, configuration and maintenance by inexperienced users – e.g., high-school students, hobbyists. The resulting *tdmimage* is prepackaged with Edge Gateway's docker-compose definition file, all the Docker-related components required to run it, and a number of utility scripts to help manage the Edge Station. The Edge Gateway is programmed to automatically configure its internal WiFi device as an Access Point on the first boot after installation to facilitate the configuration of the system by the user. The Edge Gateway is accessed via SSH and settings must be edited in a single configuration file. After the initial configuration – which includes appropriately configuring the device to connect to the domestic network – the system forces the user to change the default password and automatically disables the WiFi Access Point mode.

SoC	**CPU**	**ISA**	**Cores**	**Clock**	**Memory**
Broadcom BCM2837B0	64-bit ARM Cortex-A53	ARMv8-A AArch64	4	1.4 GHz	1 GB (shared)
		Networking			
Ethernet 1Gbps		WiFi 802.11.b/g/n/ac		Bluetooth 4.2	
		Storage			
		microSD card			
		Peripherals			
4 x USB 2.0		40 x GPIO	HDMI	I^2S	CSI

Table 1: Raspberry Pi 3 Model B+

4.2 Handlers

The handler microservices in the Edge Gateway act as the interface between the myriad of possible sensors and the data processing platform. Our handlers on the Edge Gateway translate the various incoming message formats to a common Fiware-compliant model. Once converted, the Fiware messages are published on the Edge Gateway's internal MQTT bus. The handler microservices are implemented in *Python* and use few other external libraries for communication and local storage.

The current Edge Gateway implementation includes the following handler microservices:

SFDS handler: for the *Stuttgart Fine Dust Sensor* [23];
IotaWatt handler: for the *IotaWatt Energy Monitor* [16];
Device handler: for Edge Gateway telemetry and directly connected sensors;

These are better described in the following paragraphs.

SFDS station handler. The SFDS station handler microservice implements communications and data acquisition from the *Stuttgart Fine Dust Sensor* – hence the name. SFDS stations are equipped with a battery of sensors, including temperature, humidity, barometric pressure, wind and rain, and PM2.5 and PM10 particulate levels. These stations can be configured to `HTTP POST` data to an arbitrary InfluxDB database. We leverage this feature by implementing a compatible interface in the SFDS handler. Thus, the SFDS station is configured to send its data to the Edge Gateway as if the latter was an InfluxDB instance; the SDFS station handler running on the Edge Gateway accepts the `POST` requests in InfluxDB format, acquires the data and then does it processing. SFDS messages on the Edge Gateway are published on the MQTT bus with the topic

```
WeatherObserved/esp8266-XXXX.DHT22
```

where *WeatherObserved* is the message type, *esp8266-XXXX* is the unique identifier of the transmitting station and *DHT22* is the ID of the sensor that generated the data.

IotaWatt station handler. The IotaWatt microservice handles interaction with the *IotaWatt Energy Monitor* [16]. This station can measure voltage, current and power, along with other derived electrical measures, on up to 14 different electrical channels. Since the IotaWatt station supports sending its data to an InfluxDB instance, the data transmission mechanism between the station and the Edge Gateway analogous to the one used for the SFDS station: the stations is configured to sent its data to a remote InfluxDb database, which is actually our Edge Gateway. Internal IotaWatt messages are published with the topic

```
EnergyMonitor/IOTAWATT-1.HVAC
```

where *EnergyMonitor* is the type the message, *IOTAWATT-1* is the unique identifier of the transmitting station and *HVAC* is the channel (HVAC) monitored.

Device handler. The Device handler generates messages with internal telemetry from the Edge Gateway system and handles sensors that are physically connected to our device. Thus, this microservice can produce multiple message types. The *Housekeeping* message is generated to send internal telemetry data, such as the kernel running on the device, internal memory, storage and processing resources available and occupied, etc. In addition, the handler reads data from a HTU21D temperature/humidity sensor that was physically attached to the Edge Gateway (through the I^2C electric bus). An message of type `HTU21D` is generated to communicate the data read from this sensor. As with all handlers, the data is stored to the internal InfluxDB Database and published to the MQTT broker. The two topics for the device handler messages are:

```
DeviceStatus/EDGE.HOUSEKEEPING
DeviceStatus/EDGE.HTU21D
```

4.3 Dispatcher

As described in Sec. 3, the Dispatcher microservice relays the data acquired by the Edge Gateway to the data collection platform. The external transmission to the platform is authenticated by the remote MQTT broker and encrypted with SSL/TLS certificates. The dispatcher implements dynamic throttling, adjusting the transmission rate and policy based on the available network bandwidth. This feature implemented at the level of the Edge Gateway allows the overall platform to cope with temporary network outages and support widely differing uplink technologies – e.g., from LTE to ADSL or even GSM – even when using sensors with very simple transmission logic. Finally, the Dispatcher also adds Edge-Gateway-related metadata to the outgoing messages, like Edge Gateway ID, timestamp and position. Further, the message topics are modified to comply with the requirements of the IoT Fiware component *IoTAgent*. Specifically, an API key that identifies the type of devices and the Edge Gateway identifier is prepended to the internal MQTT topic, */Type/Edge.Station.Sensor*. The string '`attrs`' is appended as per Fiware IoT protocol requirements. As an example,

```
WeatherObserved/esp8266-YYYYYYY.BME280
```

becomes

```
/<APIKEY>/Edge-XXXXXXXX.esp8266-YYYYYYY.BME280/attrs
```

Like the Handlers, the Dispatcher is also implemented in Python.

4.4 Dashboard

The Grafana ancillary service on the Edge Gateway is used to provide a user-accessible interface to the data recorded data on the local InfluxDB instance. A convenient web dashboard that summarizes data collected from the standard handlers described in this Section is provided and preinstalled. Moreover, users can be easily generate their own views of the data leveraging Grafana's functionality to plot graphs, charts and create gauge widgets that visualize collected data. In addition, the Grafana service can also be configured also to send email alerts in the case of critical events – e.g., measured temperature too high or too low, unavailability of some sensor. Fig. 4 shows a screenshot of Grafana running on the Edge Gateway.

4.5 Creating small Docker container images

The size of software container images used in the Edge Gateway implementation directly affects the time to download and launch the images [8] – which entails effects on the time to first start the device and to deploy updates. To minimize their size we have chosen the Alpine Linux image for the ARM64 architecture [2] as the base for all our container images (about 5 MB in size). The image grows quickly with the installation of the Python interpreter and external libraries.

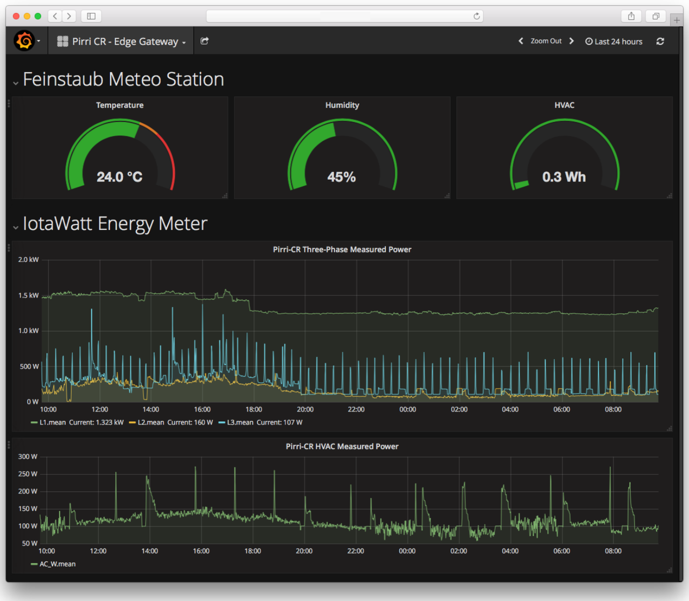


Fig. 4: Screenshot of the Grafana dashboard running on the Edge Gateway.

However, the layered storage approach used by Docker gives us the opportunity to contain the total size of all the images used by our system by ensuring they re-use the base layers – which are therefore downloaded and stored only once – while all the application-specific files are added at the top of the stack. Our overall layer stack is illustrated in Fig. 5.

To produce efficiently compact top layers for our images a multi-stage Dockerfile is used. The building process is split in a *building* stage and a *final* stage. The first image is used to build the application and libraries, and thus contains all build-time dependencies like compilers, header files, and so on. On the other hand, the latter image includes only the resulting executables (copied from the build image) and the runtime dependencies.

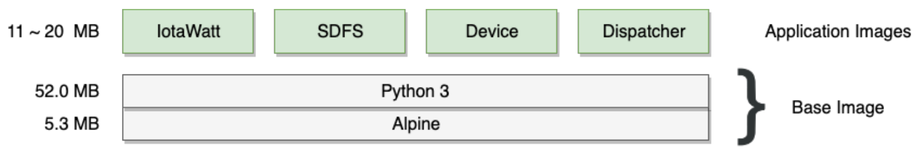


Fig. 5: Container layer stack

Table 2 summarize the different layer sizes as reported by the *Dive*[3] inspecting tool. Splitting handlers and dispatcher into simple microservices that only implement a single functionality results in code that is relatively short and readable, and facilitates debugging and correcting errors. By creating these microservices on images that share their base *alpine-python* image layers, the storage required by the images is reduced on average by 70%, thus avoiding the potential pitfall of multiplying the volume if images to transfer and store by the total number images to be used on the system.

Container	**Sensors**	**Lines**	**Layer Size**	**Container Size**
sdfs	Weather/Air	402	20 MB	77.7 MB
iotawatt	Energy	354	20 MB	77.7 MB
device	Onboard	476	16 MB	73.9 MB
dispatcher	Data Relaying	424	11 MB	68.9 MB
alpine-python		Base System and Python		58.0 MB

Table 2: Containers overview

4.6 Deployment

A number of Edge Gateways were deployed and tested, first in a laboratory setting and later in offices and private homes. Fig. 6 depicts the overall architecture of the project TDM infrastructure spanning from sensors to the cloud-based data processing platform. The Edge Gateways collect and relay to the processing platform the data generated by sensors and stations. The TDM platform also integrates data from other sources, like satellite images, weather radar images, and various other geo-referenced data. Data are archived and indexed by a system that combines different technologies (i.e., *HDFS*, *SQL* and *NOSQL* databases) storing data on the most appropriate system depending on its characteristics: e.g., point-like, images, time-series, etc. Various means are provided to access and process the collected data. *Jupyter*[4] notebooks are available for ad hoc queries and analyses. Event-driven and periodic unattended operations can be performed using the *Apache Airflow*[5] workflow engine or through *Grid Engine*[6] batch computing jobs. Finally, aggregated and processed data are published as Open Data on the TDM *CKAN*[7] portal from where they can be downloaded.

Two Open Source Open Hardware sensor and measurement stations platoforms were used as a starting sensor set: the SFDS weather and air station

[3] https://github.com/wagoodman/dive
[4] https://jupyter.org/
[5] https://airflow.apache.org/
[6] http://gridscheduler.sourceforge.net/
[7] https://ckan.org/

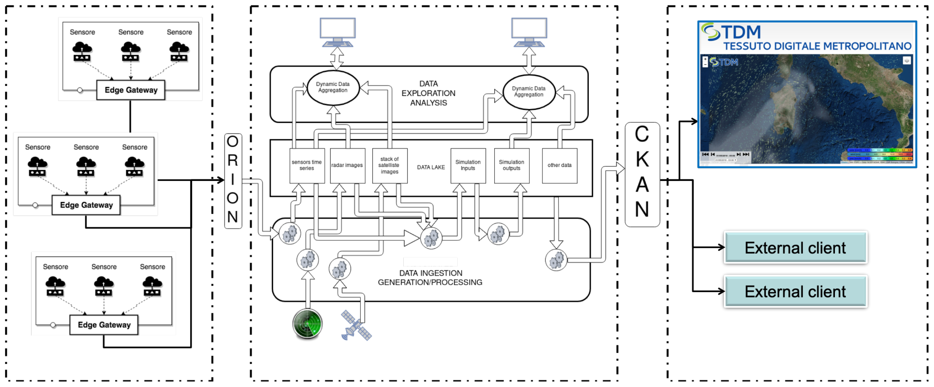
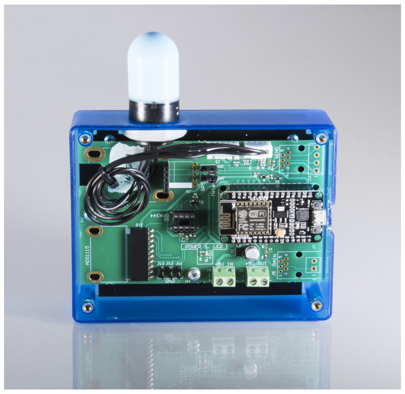


Fig. 6: TDM Cloud Architecture

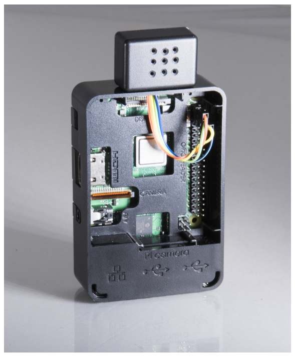
and the IotaWatt Energy Monitor. To support the three different scenarios for weather and air measurements – *outdoor*, *indoor*, and *mobile* – and to make station construction accessible to low-skill users, we have designed and produced a single PCB board to distribute to a number of early joiner volunteers. This board contains all the connectors to integrate the different sensors and modules to build the three types of station. No soldering is required and a board can be quickly reconfigured from one scenario to another by plugging and unplugging the desired modules. Fig. 7 shows the Indoor station with the board, the temperature, humidity and barometric pressure sensors, and the *ESP12-F* micro-controller. The same image also shows the IotaWatt module in our Computer Room, the TDM Edge Gateway with a directly attached sensor for indoor temperature and humidity measurements, and the mobile station boarded on a bike for mobile air quality data acquisition.

5 Related Work

Edge Computing is a hot topic in IoT and BigData analysis driven by the desire to leverage IoT devices for the wealth of valuable data they can collect and the fact that scalability, network latency and resiliency concerns make it unviable to rely solely on centralized computing to collect the data generated by large sensor networks. Pace et al. [24] identified Edge Computing as a way to overcome the issues that affect Cloud computing in Healthcare applications. Large-scale patients services can burden the network and thus deteriorate latencies and break the real-time constraints of critical applications. Moreover, patient data cannot always be store in Cloud due to privacy and data security concerns. In addition, speed of data analysis and response is crucial in autonomous and semi-autonomous decision systems. They propose the *BodyEdge* complete architecture for a mobile Edge Gateway for uses in the Healthcare industry. It is composed by two complementary components: a smartphone-hosted relay node for the Body Sensor Network devices that cannot directly talk with the Edge Gateway, and


(a) Indoor Station

(b) IotaWatt


(c) Edge Gateway

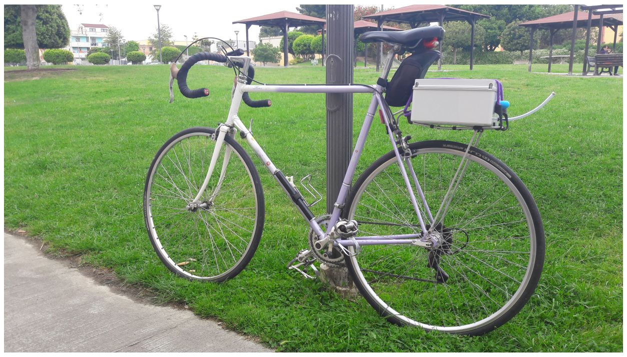

(d) Mobile Station on bike

Fig. 7: TDM Sensors and Edge Gateway

an Edge Gateway that actually provides the healthcare services and communications to the far Cloud. Unlike the architecture we propose, BodyEdge modules seem to be monolithic applications. Microservice-based design or containerization are not mentioned, and communication between the modules are described as client-server.

In comparison, the *AGILE* framework for IoT gateways is designed based on a microservices architecture [8] and its software components are delivered using Docker containers. Various IoT high-level functionality is provided, such as device and protocol management, UI and SDK for IoT development, gateway management UI. Similar our architecture, protocol handlers are implemented in individual containers and communicate with each other using DBus. Differing from the TDM Edge Gateway, AGILE specializes handler microservices on sensor communication protocol, instead of sensors types. The primary motivation behind this different decision is the fact that the TDM station handlers are primarily designed to translate sensors data to a common FIWARE format. It is not specified if AGILE uses D-Bus publish/subscribe or in one-to-one request-response mode.

Another architecture for general purpose Edge Gateways, similar to our TDM Edge Gateway, is the *LEGIoT – Lightweight Edge Gateway for the Internet of Things* [20]. The architecture proposed is based on microservices running on Docker containers and implemented using low-cost Single Board Computers like *Odroid* (C1+, C2) and Rasbperry Pi (RPi2, RPi3). It has a modular and flexible design similar to TDM Edge Gateway. Modules are divided in *Northbound*, in charge of communication to the remote end, and *Southbound* that deal to the local sensors. Northbound and Southbound modules are *activated on-demand* to limit power usage. Received data are saved to a local database and are made available to other containers. Unlike in our TDM Edge Gateway, the internal data exchange is performed by a dedicated module using a custom API, while the TDM design relies on standard asynchronous and agnostic publish/subscribe mechanisms. A deeper difference between the two designs is that the TDM Edge Gateway continuously transmits the acquired data, while LEGIoT implements a "pull" strategy, whereby data are retrieved by the remote end upon activation of a suitable Northbound module acting as protocol server. Finally, LEGIoT supports multi-tenancy while TDM Edge Gateway is single-tenant by design, responding to specific demands of TDM research.

6 Conclusions and Future Work

This paper presents a flexible and scalable microservice-based Edge Gateway architecture that facilitates integrating heterogeneous sensors in complex IoT data acquisition applications. The architecture is particularly well suited to research applications, given the possibility to quickly and easily add or substitute its on-board software components. Indeed, the TDM Edge Gateway architecture was developed and is being used in the context of the "Tessuto Digitale Metropolitano" project, where it is a component of a research-oriented urban technological fabric. The testing and deployment phases confirm the advantages of the microservice architecture when combined with publish/subscribe data exchange protocols.

In future work, as the installed base of the device expands, with deployments in public buildings, schools and private dwellings, a number of measurement and performance statistics will be available for future study. The introduction of auto-tuned transmission policies based on bandwidth statistics and availability is planned for the dispatcher. From the Edge Computing perspective, we are evaluating the integrating computational algorithms for the estimation and forecast of power consumption, such as those by Massidda et al. [17, 18], as containerized applications. Finally, feedback mechanisms for home appliances and actuators are under evaluation.

References

1. Al-Kuwari, M., Ramadan, A., Ismael, Y., Al-Sughair, L., Gastli, A., Benammar, M.: Smart-home automation using IoT-based sensing and monitoring platform. In: 2018 IEEE 12th International Conference on Compatibility, Power Electronics and Power Engineering (CPE-POWERENG 2018). pp. 1–6. IEEE, Doha (Apr 2018). https://doi.org/10.1109/CPE.2018.8372548, `https://ieeexplore.ieee.org/document/8372548/`
2. Alpine Linux: Alpine Linux Home Page. `https://alpinelinux.org/`, Online; accessed 18-April-2019
3. Arch Linux ARM: Raspberry Pi 3 — Arch Linux ARM. `https://archlinuxarm.org/platforms/armv8/broadcom/raspberry-pi-3`, Online; accessed 18-April-2019
4. Beserra, D., Moreno, E.D., Endo, P.T., Barreto, J., Sadok, D., Fernandes, S.: Performance analysis of lxc for hpc environments. In: 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems. p. 358–363. IEEE (Jul 2015). https://doi.org/10.1109/CISIS.2015.53
5. Cloutier, M., Paradis, C., Weaver, V.: A Raspberry Pi Cluster Instrumented for Fine-Grained Power Measurement. Electronics **5**(4), 61 (Sep 2016). https://doi.org/10.3390/electronics5040061, `http://www.mdpi.com/2079-9292/5/4/61`
6. Di Tommaso, P., Palumbo, E., Chatzou, M., Prieto, P., Heuer, M.L., Notredame, C.: The impact of docker containers on the performance of genomic pipelines. PeerJ **3**, e1273 (Sep 2015). https://doi.org/10.7717/peerj.1273
7. Docker Inc: Docker Documentation. `https://docs.docker.com/`, Online; accessed 20-May-2019
8. Dolui, K., Kiraly, C.: Towards Multi-Container Deployment on IoT Gateways. In: 2018 IEEE Global Communications Conference (GLOBECOM). pp. 1–7. IEEE, Abu Dhabi, United Arab Emirates (Dec 2018). https://doi.org/10.1109/GLOCOM.2018.8647688, `https://ieeexplore.ieee.org/document/8647688/`
9. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: Yesterday, Today, and Tomorrow. In: Mazzara, M., Meyer, B. (eds.) Present and Ulterior Software Engineering, pp. 195–216. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978 $-$ 3 $-$ 319 $-$ 67425 $-$ 4_12, `https://doi.org/10.1007/978-3-319-67425-4\_12`

10. Eclipse Foundation: Eclipse Mosquitto. https://mosquitto.org/, Online; accessed 30-April-2019
11. Fiware Foundation: Home — FIWARE. https://www.fiware.org/, Online; accessed 14-May-2019
12. Fiware Foundation: Smart Cities — FIWARE Open Source Platform for Smart Cities. https://www.fiware.org/community/smart-cities/, Online; accessed 14-May-2019
13. Grafana Labs: Grafana — The open platform for analytics and monitoring. https://grafana.com/, Online; accessed 30-April-2019
14. Hajji, W., Tso, F.: Understanding the Performance of Low Power Raspberry Pi Cloud for Big Data. Electronics **5**(4), 29 (Jun 2016). https://doi.org/10.3390/electronics5020029, http://www.mdpi.com/2079-9292/5/2/29
15. InfluxData: InfluxDB 1.7 documentation — InfluxData Documentation. https://docs.influxdata.com/influxdb/v1.7/, Online; accessed 30-April-2019
16. IoTaWatt, Inc: IoTaWatt — Open WiFi Electricity Monitor. https://iotawatt.com/, Online; accessed 29-April-2019
17. Massidda, L., Marrocu, M.: Quantile regression post-processing of weather forecast for short-term solar power probabilistic forecasting. Energies **11**(7), 1763 (july 2018). https://doi.org/10.3390/en11071763, http://publications.crs4.it/pubdocs/2018/MM18a
18. Massidda, L., Marrocu, M.: Smart meter forecasting from one minute to one year horizons. Energies **11**(12) (december 2018). https://doi.org/10.3390/en11123520, http://publications.crs4.it/pubdocs/2018/MM18b
19. Merkel, D.: Docker: lightweight linux containers for consistent development and deployment. Linux Journal **2014** (03 2014)
20. Morabito, R., Petrolo, R., Loscrì, V., Mitton, N.: LEGIoT: A Lightweight Edge Gateway for the Internet of Things. Future Generation Computer Systems **81**, 1–15 (Apr 2018). https://doi.org/10.1016/j.future.2017.10.011, https://linkinghub.elsevier.com/retrieve/pii/S0167739X17306593
21. NodeMcu Team: NodeMcu — An open-source firmware based on ESP8266 wifi-soc. https://www.nodemcu.com/index_en.html, Online; accessed 29-April-2019
22. OASIS Open 2015: MQTT Version 3.1.1. https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html, Online; accessed 09-May-2019
23. OK Lab Stuttgart: Home — luftdaten.info — Feinstaub selber messen. https://luftdaten.info/en/home-en/, Online; accessed 29-April-2019
24. Pace, P., Aloi, G., Gravina, R., Caliciuri, G., Fortino, G., Liotta, A.: An Edge-Based Architecture to Support Efficient Applications for Healthcare Industry 4.0. IEEE Transactions on Industrial Informatics **15**(1), 481–489 (Jan 2019). https://doi.org/10.1109/TII.2018.2843169, https://ieeexplore.ieee.org/document/8370750/
25. Saha, S., Majumdar, A.: Data centre temperature monitoring with ESP8266 based Wireless Sensor Network and cloud based dashboard with real time alert system. In: 2017 Devices for Integrated Circuit (DevIC). pp. 307–310. IEEE, Kalyani, India (Mar 2017). https://doi.org/10.1109/DEVIC.2017.8073958, http://ieeexplore.ieee.org/document/8073958/
26. Tzounis, A., Katsoulas, N., Bartzanas, T., Kittas, C.: Internet of Things in agriculture, recent advances and future challenges. Biosystems Engineering **164**, 31–48 (Dec 2017). https://doi.org/10.1016/j.biosystemseng.2017.09.007, https://linkinghub.elsevier.com/retrieve/pii/S1537511017302544

27. Wikipedia: Publish–subscribe pattern — Wikipedia. `https://en.wikipedia.org/wiki/Publish-subscribe_pattern`, Online; accessed 20-May-2019