Vila, M.; Sancho, M.-R.; Teniente, E. XYZ Monitor: IoT monitoring of infrastructures using microservices. A: Workshop on Smart Data Integration and Processing on Service Based Environments. "Service-Oriented Computing, ICSOC 2020 Workshops: AIOps, CFTIC, STRAPS, AI-PA, AI-IOTS, and satellite events: Dubai, United Arab Emirates, December 14-17, 2020: proceedings". Berlín: Springer, 2020, p. 472-484. ISBN 978-3-030-76352-7. The final authenticated version is available online at https://doi.org/10.1007/978-3-030-76352-7_43

XYZ Monitor: IoT Monitoring of Infrastructures using Microservices

Marc Vila[1,2], Maria-Ribera Sancho[1,3], and Ernest Teniente[1]

[1] Universitat Politècnica de Catalunya, Barcelona, Spain
{marc.vila.gomez,maria.ribera.sancho,ernest.teniente}@upc.edu
[2] Worldsensing, Barcelona, Spain
mvila@worldsensing.com
[3] Barcelona Supercomputing Center, Barcelona, Spain
maria.ribera@bsc.es

Abstract. One of the main features of the Internet of Things (IoT) is the ability to collect data from everywhere, convert this data into knowledge, and then use this knowledge to monitor about an undesirable situation. Monitoring needs to be done automatically to be practical and should be related to the ontological structure of the information being processed to be useful. However, current solutions do not allow to properly handle this information from a wide range of IoT devices and also to be able to react if a certain value threshold is exceeded. This is the main purpose of XYZ Monitor, the system we propose here: to monitor IoT devices so that it can automatically react and notify when a given alarm is detected. We deal with alarms defined by means of business rules and allow setting ontological requirements over the information handled.

Keywords: IoT · Monitoring · API · Microservices · Framework

1 Introduction

Several authors recognize the IoT to be one of the most important developments of the 21st century [14]. According to them, the IoT represents the most exciting technological revolution since the Internet because it brings endless opportunities and impact in every corner of our planet. IoT devices are used as human consumables such as wearables or health trackers; but they are also key to the success of industrial applications such as Smart Cities, Industry 4.0, Smart Energy, Connected Cars or Healthcare. I.e., almost all industrial environments are currently highly dependant from the IoT.

IoT devices and systems are intended to collect and process data from the least expected places, and its expansion is allowing to operate sensors in a wide range of applications; energy management, mobility, manufacturing, Smart Cities [18] or healthcare, where there is the need of services able to monitor the medical condition of a patient [7]; or even operated in private use at home, for example to monitor the home safety.

Thus, one of the inherent capabilities of such IoT systems is the ability to automatically monitor the information associated to the raw data they are pro-

cessing. This has to be achieved by transforming raw data into relevant knowledge of the system domain, and then specifying conditions over this knowledge, referred to as alarms, that allow identifying and handling undesirable situations.

Microservices are in the core of providing a solution for such monitoring systems. It is an architectural style which promotes developing an application "as a suite of small services, each running in its own process and communicating with lightweight mechanisms with the others" [6]. There is a need for managing IoT platforms through microservices-based architectures to facilitate IoT development in itself, improving scalability, interoperability and extensibility [17].

This is particularly important when automatic IoT monitoring is concerned, since microservices and IoT share a lot in common in terms of architectural goals [3]. However, to our knowledge, previous work used to assume monolithic architectures [7,8,11], without taking into account all the benefits of a microservices orientation. An exception can be found in [4], which implements a service-oriented architecture (SOA) for monitoring in agriculture; and in [10] that defines microservices architecture about security monitoring in public buildings.

Summarizing, we can see that previous proposals dealing with microservices, monitoring and IoT consider only very specific domains, and are intended to monitor a certain data, from a particular sensor to achieve a single given response. I.e., they are tailor-made and there is no general purpose domain independent proposal.

Our work is related to the industrial research and innovation at Worldsensing (www.worldsensing.com), which focuses on the monitoring of industrial environments through IoT systems. In this sense, one of the company's main goals is to develop a generic environment (assuming different device types and data from different providers) able to monitor systems and alarms, as defined through customizable business rules, and based on a microservices architecture. This is, in fact, the main contribution of this paper. The system we have developed, called XYZ Monitor, is Open Source, easy to use and applicable to different domains.

2 Related Work

We distinguish between IoT in monolithic and microservice architectures, and monitoring in IoT in both architectures.

2.1 IoT in Monolithic Architectures

In these architectures, systems are built, tested, and deployed as one large body of code, as a unique solution [12]. This is the classical way of building applications in software deployments. Everything is unified and, thus, there is no modularity.

Among the relevant work in this area we may find [5], which focuses on a Service Oriented Middleware; [2] which designs a Service Oriented Architecture (SOA) for wireless sensor networks; and [15] that expands the SOA concept with cloud-based Publish/Subscribe Middleware and also implements the Web of Things (WoT) concept.

The main drawback of these solutions is that of successfully handling IoT environments which are increasingly complex, with many kinds of devices that are heterogeneous as far as their use and operation. Monolithic systems have their main limitations here. Everything is linked, if there is a change, improvement or correction to make, even if it is minimal, the whole system has to be deployed, tested and restarted. If one part of the system stops working, it is very likely that the whole system will stop working. Moreover, their reusability is very limited.

2.2 IoT in Microservice Architecture

The microservice architecture emerged as a solution to overcome previous drawbacks [12]. In this architecture, systems are developed as a set of self-contained components, or loosely coupled services, also called microservices. Each microservice encapsulates its logic to implement a single business function, and communication is done through web interfaces (APIs). This approach has contributed to improved fault isolation, simplicity in understanding the system, technology flexibility, faster technical deployments, scalability, and reusability [13].

Several works are intended to provide IoT solutions through a microservice architecture. [9] explores how the service-oriented architecture paradigm may be revisited to address challenges posed by the IoT for the development of distributed applications. [3] investigates patterns and best practices used in microservices and analizes how they can be used in the IoT. [17] proposed an architecture of a microservice based middleware, to ensure cohesion between different types of devices, services and communication protocols. [1] proposes a modular and scalable architecture based on lightweight virtualization, with Docker. [16] proposes an open microservice system framework for IoT applications. [19] provides an environment to transform automatically functionalities from IoT devices to a Service Oriented Architecture based IoT services.

2.3 Monitoring in IoT

Some proposals have also been devoted to monitoring on IoT. [7] discusses the integration of IoT devices for health monitoring. [8] extends the previous concept to include safety protocols for data transmission also in healthcare. [11] proposes an IoT-based solution to monitor Smart Cities environments. However, all these proposals are based on monolithic architectures.

Moving to non-monolithic architectures, [4] implements a SOA for monitoring in agriculture; while [10] develops a microservice based architecture for a monitoring system to improve the safety of public buildings. It is worth noting that both applications are very domain-specific and that they handle only very concrete devices.

Summarizing, the use cases are very specific. Input elements to the systems are very limited, only certain devices are available. And once the data is in the system, it follows a closed monitoring flow. They include a business rules with notification system. But, this is also closed to modifications and cannot be changed externally.

3 XYZ Monitor System Overview

Our goal is aimed at overcoming the limitations of previous proposals. With this purpose, we have built the *XYZ Monitor* system, which is able to monitor data from different IoT devices and reporting if certain conditions over this data are met. Input data is generic and extensible to changes to support different elements. IoT devices communicate with the system via HTTP calls, through APIs. Once the data is in the system, it is analyzed and monitored. If the user has specified an alarm by means of a business rule, it will be monitored by sending notifications to an email address indicated in the system. XYZ Monitor relies on a microservices architecture, with the advantages that this entails.

XYZ Monitor gets the input of data from the devices themselves since we naturally assume that the real-world objects, sensors and devices can autonomously infer their state and submit this information to the service. This is a feasible assumption in the context of the IoT, where environmental data can be collected by the objects, which can then infer their own state.

Once in the system, this data will serve us to monitor the behaviour of the devices and also to activate alarms if certain conditions over the data are met.

3.1 Conceptual Overview

The workflow of our proposal is summarized in Figure 1. Initially, the *Data Collection* component receives information inputs from the different devices. These devices are onboarded in the *Data Management* system by the user. These data can be visualised on the platform itself as passive monitoring, *Monitoring* component. At the same time that the data is being received, the system, performs *Data Analysis*. This is done through business rules, predefined by the platform user, it is also called alarm monitoring. When an alarm occurs, i.e. a business rule detects that something is not right, like a value out of range, the system warns through the *Notification* component, automatically.

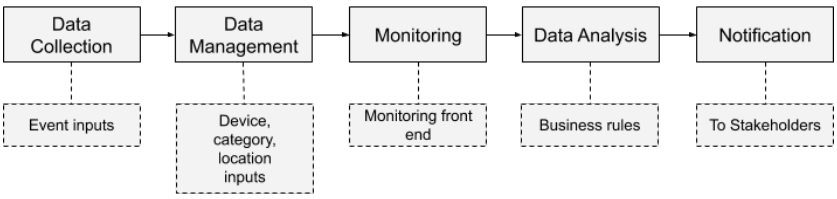

Fig. 1. Workflow of *our proposal*

One of the main features of our system is its ability to handle alarms that are defined by means of business rules, specified over the conceptual ontology of the information handled. This conceptual ontology is defined in Figure 2. We use UML in the figure because it is a widespread language and because it allows abstracting the concepts of interest from its technological implementation.

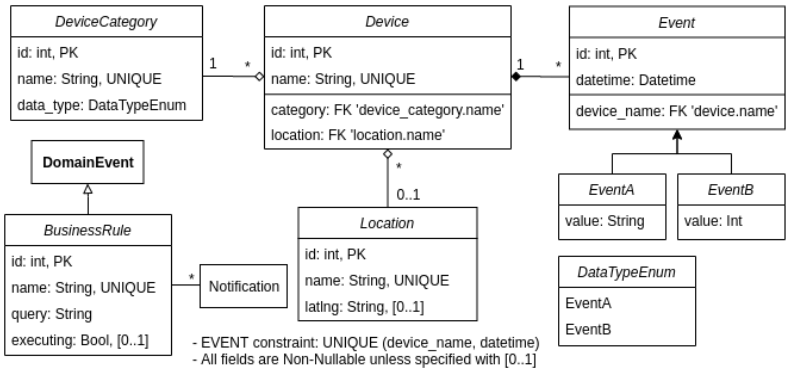

Fig. 2. Ontology of the information handled by XYZ

An *event* is an input of data sent by an IoT device to our system. We assume two different types of *events*, although our proposal is easily extensible by assuming additional subclasses of *events* with different types. A *device* is the smart object intercepting the *events* (such as a sensor or a thermometer). Each *device* has a location in coordinates format or it is named with a label. A *device* belongs to a *device category* which manages it and serves to indicate what type of *events* that *device* is sending. Finally, a *business rule* (also called alarm) allows stating a complex condition to be monitored over the data stored by means of a query and also notify to whoever has been determined in the system.

3.2 Architectural Overview

Our system has been designed to operate through a microservices architecture. Each functional module is isolated and communicates with each other over HTTP interfaces. This architecture is specified in Figure 3.

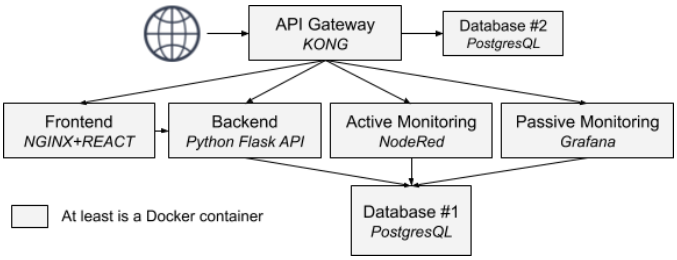

Fig. 3. Microservice architecture of *our proposal*

When a user or an IoT device wants to communicate with our system, our API Gateway will be in the front-line. An API Gateway takes all the HTTP requests from a client determines which services are needed and then routes them to the appropriate microservice. It translates between web protocols and web-unfriendly protocols, used internally. Among the existing, Netflix Zuul,

Kong, Apache Apisix, etc., we have chosen Kong (https://konghq.com/kong/), an Open Source Software, simple to configure, it works well and meets our needs.

Among the existing OSS data visualization systems for monitoring, *e.g.* Kibana, Prometheus, Grafana or Chronograf, we decided to use Grafana (https://grafana.com) because it is easy to use and fits well with most types of IoT data we are dealing with. We are able to have a monitoring and visualization system with the data reported by the devices.

Our system incorporates an element that allows it to react to certain circumstances. Node-Red (https://nodered.org), its OSS, and permits creating flows for actions through custom JavaScript functions. Within this system we have created a flow to check these circumstances or business rules.

Also a frontend has been developed in React (https://reactjs.org), which is an OSS JavaScript framework. The backend is done with Flask (https://flask.palletsprojects.com), an OSS micro framework in Python. As database, PostgreSQL (https://postgresql.org) is used, an OSS object-relational database.

The system architecture has been deployed under microservices by the means of Docker (https://docker.com) and Docker-compose (https://docs.docker.com/compose/). Docker is an open source project providing a systematic way to automate the faster deployment of Linux applications inside portable containers.

4 Proposal in a nutshell

4.1 Data Collection

Our system is easily extensible. Therefore, to exemplify its input we have chosen to distinguish between two types of inputs available for the system. If the user has the need to add another type of input, it can be done easily. This is why all input elements have a superior element, which generalizes their values. More precisely, XYZ Monitor is able to handle two types of possible elements: *EventA* and *EventB*, respectively. Both kind of inputs are specializations of input type *Event*, which is a communication trigger that is the base type for a device reading.

Events An *Event* includes the information of the device to which the data is being input (*device_name*), and also the date of sample collection (*datetime*); but it does not have the ability to know what value that sample has. The value of the sample is obtained from the classes inherited from *Event*. Which can be *EventA* or *EventB*. For this, an API (`/api/events`) has been developed to gather *Events* sent to the system. It is defined following the JSON Schema [4] specification:

Listing 1.1. Event JSON Schema simplification

```
"event": {
  "type": "object",
  "properties": {
    "device_name": {
      "type": "string",
      "$ref": "#/definitions/device/properties/name"
    },
```

[4] JSON Schema - A Media Type for Describing JSON Documents: https://tools.ietf.org/html/draft-handrews-json-schema-01

```
    "datetime": {
      "type": "string",
      "format": "date-time",
      "example": "2020-04-01T10:54:03+00:00"
    }
  },
  "required": [ "device_name", "datetime" ] }
```

EventA is an input value in the form of a string, i.e. it can be any alphanumeric element. For example, "heat", "1.0", "1", etc.

Listing 1.2. EventA JSON Schema simplification

```
"event_a": {
  "type": "object",
  "allOf": [  { "$ref": "#/definitions/event" },
              { "properties": {
                  "value": {
                    "type": "string",
                    "example": "1"
                  }
                },
                "required": [ "value" ] } ] }
```

EventB events are defined in a similar way, but the input value that is in the form of a number, so it can have numerical values, even with decimal values.

XYZ Monitor shows the events that have been correctly received by the system (see Figure 4). In particular, XYZ monitor provides for each event: its identifier, device to which it reports, value and date of reception.

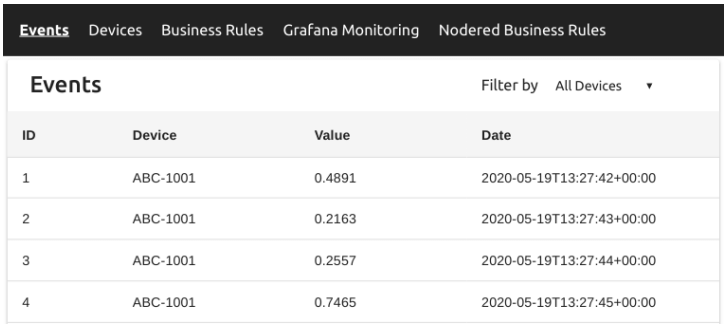

Fig. 4. Frontend showing the Events yet in our system

4.2 Data Management

We have seen how to send an Event to the system. Next we will see how to indicate that a certain event is from a specific device. And how the other attributes of a device are handled.

Devices The system is prepared to handle an infinite amount of devices. A device can be an element of the world as well as virtual. Infinite elements can coexist in the system. Since the same user is in charge of registering them in the system.

A *Device* can have only one point of measuring, which is why it is associated with a type of reading, the *category* field. But also a *Device* can be located into an *Area*, this is field is optional and its named *Location*. For this purpose, an API (`/api/devices`) has been developed that acts as an on-boarder to the system. This is achieved by means of a JSON schema similar to the previous ones.

XYZ Monitor shows the existing devices as illustrated in Figure 5. For each device, our system provides its name, its category, the type of event it reports and its location. Moreover, on the right side of the figure, we show how Grafana is reporting in real time the events that are being received in that device.

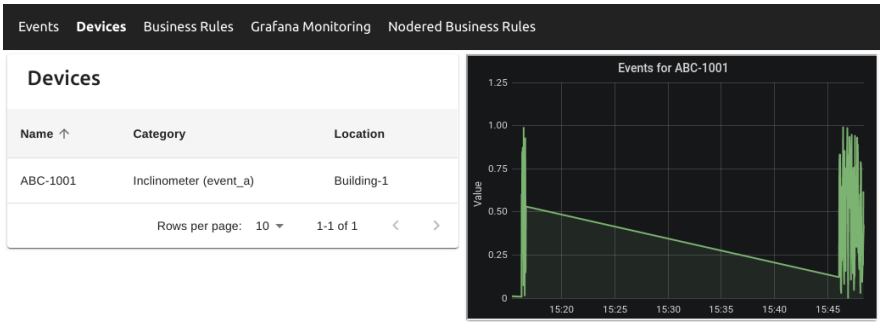

Fig. 5. Frontend with a Device and its Events, showing Grafana to monitor it

Device Categories At this point we already know how to insert a *Device* and its *Events* into our system. However, we have not yet defined how to differentiate *Events* from one *Device* to another. The *category* field inside the *device* component gives us a small clue. A *Category* is a set of possible types of elements defined by its *name* and the *data_type* field, which is an string-enumeration of the possible types of *Events* available in the system (*EventA* or *EventB*). This is achieved by means of a JSON schema similar to the previous ones and it is accessible through the endpoint (`/api/device-categories`).

Locations As mentioned, a *Device* has the possibility of being located in one area. The location of a device is given by a *name*, in addition, there is the possibility of including coordinates, *latlng*, in order to have a precise location, in *latlng* format. Can be accessed in the API endpoint (`/api/locations`). This is achieved by means of a JSON schema similar to the previous ones.

4.3 Monitoring

At this point, our system is able of receiving external, sensor-based, information and classify it accordingly, by type of data, category and also by its location. To observe the data, the system has been equipped with a data visualization environment for the sake of monitoring.

Figure 6 shows the visualization system, built with Grafana `/grafana`. It is displaying all the *Events* that have been received from the ABC-1001 *device*, which are of type *EventA* on the date from 14:49 to 15:49 of 19/05/2020.

It can be observed that the data has arrived well between 14:49 - 14:50, 15:16 - 15:18 and 15:48 - 15:49, while no data has been received for the other ranges. The system therefore draws a continuous line with no variation between them to note that there is no data there.

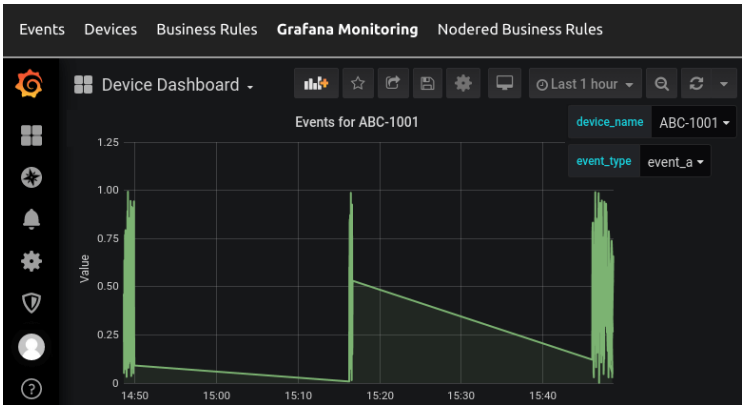

Fig. 6. Grafana panel showing the Device' Events in our system

The graph in Figure 6 is generated from an *SQL query*, specified in Listing 1.3. This SQL query is a traditional query against a PostgreSQL database. The advantage of running sql queries in Grafana is that the text starting with $ are the variables, and these can be sent by the browser to the system, at the time of accessing the URL, which allows to dynamically generate these kind of graphs.
`/grafana/device-dashboard?var-device_name=ABC-1001&var-event_type=event_a`

Listing 1.3. SQL query to show Device's data in Grafana

```
SELECT event.datetime as "time",
       $event_type.value,
       event.device_name
FROM event
INNER JOIN device ON event.device_name=device.name
INNER JOIN $event_type ON event.id=$event_type.id
WHERE event.device_name = '$device_name'
```

4.4 Business Rules

Figure 7 shows the flow of the *Business Rules* to achieve notifications. Its implemented with the Node-Red tool, which works by execution flows. The whole processing of *Business Rules* is based in a single flow, `/nodered` endpoint.

The execution is as follows: every 5 seconds (timer box) a query is prepared (query box) and the database that contains all the data is queried (Postgres #1 box), to check if there exists any *Business Rules* in *executing* state. The result

is checked to see if there are any (prep array-loop and array-loop boxes). In the case that there is no *business rule* or its is not in *executing* state, the flow ends here. In the case of having *active Business Rules*, the system checks one by one (prepare BR query box) and executes the query against the database (Postgres #2 box) to check for alarms. If there is any, an email is prepared (generate email message box) and sent through the *Notification* system (notification box).

A *Business Rule* is identified by a *name* and it contains a *query* field where the designer specifies the SQL query identifying the alarm. For instance: `SELECT * FROM Event where device_name = 'ABC-1001'`. And the *executing* field is where the *Business Rule* can be started or stopped temporarily, and act as an *status* of it. The query system is an SQL call, because in this way, the options for the user are not limited. This is achieved by means of a JSON schema similar to the previous ones. Business rules can be specified on: *Device Categories*, *Device Locations*, *Devices*, *Events* and any kind of combinations between them.

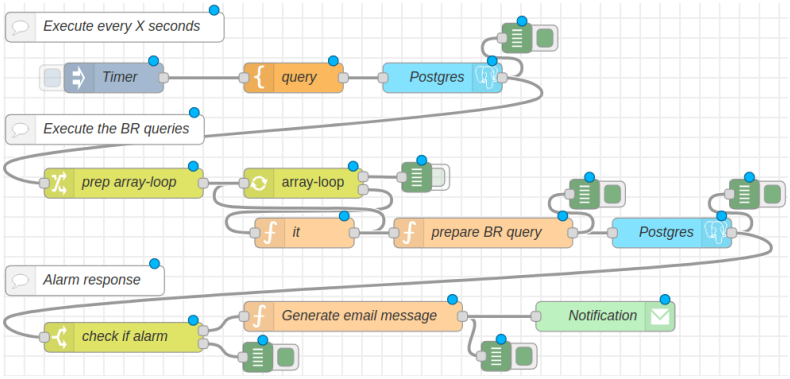

Fig. 7. Our flow to check Business Rules in Node-Red

4.5 Notifications

The notification system has been implemented in the Node-Red microservice, as an independent code block. It supports sending of notifications via email to a predefined recipient added in the system. These emails are sent via SMTP (Simple Mail Transfer Protocol), and its content warns about an alarm; informing regarding the device, the date and the values that provoked it.

Listing 1.4. SMTP Email sample sent from our Node-red

```
FROM: sender@example.com
TO: receiver@example.com
DATE: Wed., 29 Apr. 13:02
SUBJECT: New alarm from Node-Red
BODY:
  There is an alarm, the following elements are involved:
  Device: ABC-1001
  Value: 0.10
  Datetime: Tue Apr 29 2020 13:01:12 GMT+0200 (CEST)
```

5 Experimentation

The XYZ Monitor system has been implemented under a microservices architecture. Specifically on Docker and Docker-compose. An overview of the *Docker* recipe that makes the system work is available in Section 5.1.

The project code is Open Sourced and it is available in GitHub, at www.github.com/worldsensing/xyz-iot-monitoring. The project has been tested under Ubuntu 18, Linux. Although it should be compatible with any Linux operating system. Making small changes, it could be used in MacOS or even Windows, but this is out of the scope of the project since Worldsensing' main systems run on Linux. Inside the repository, the steps to follow to setup, initialize and run the whole system work are clearly stated. Everything is prepared so that the modifications required to make it work are minimal since the scripts we have prepared take care of most of the work.

5.1 Orchestration

A microservices architecture requires having, at least, a file with the information of the services that will be deployed. In our case, with *Docker* and *Docker-compose* we have written the orchestration file, in *.yaml* format, that contains the instructions to make the whole system work [5].

Listing 1.5. Docker-compose orchestration

```
services:
  # API Gateway - Base Image - Kong 2
  # Frontend - Custom - Nginx 1.17.9 + Alpine + React 16.13
  # Monitoring - Custom - Grafana 6.6.2
  # Business Rules - Base - Node-red
  # Backend API - Custom - Python 3.8.2 + Alpine 3.11 + Flask
  # Main Database - Base Image - PostgresQL 11.7
```

6 Conclusions and Further Work

We have presented the *XYZ Monitor* system as an extensible solution to successfully handle general purpose alarms defined over different kinds of devices in an IoT environment. In our system, alarms are defined by means of business rules specified over the ontological structure of the information handled by these devices. The solution we have developed is based on a microservices architecture, to facilitate the assignment of responsibilities among the components involved in alarm monitoring. Our solution is fully Open Source and it is publicly available.

As further work, we plan to enrich further the ontological structure of the information and to develop techniques to incrementally compute whether an alarm has been activated. In addition to observing how the system behaves when there are many devices sending information at the same time. Overall, the final goal of this work is to put this system into practice at Worldsensing.

[5] This listing is simplified a lot to avoid making it extremely long here. The actual one can be found in the root folder of the repository.

Acknowledgements: This work is partially funded by Industrial Doctorates Plan from Generalitat de Catalunya (DI-2019). Also with the support of inLab FIB at Universitat Politècnica de Catalunya and Worldsensing S.L. The REMEDiAL project, funded by Ministerio de Economia, Industria y Competitividad (TIN2017-87610-R); and the Generalitat de Catalunya (2017-SGR-1749) have also contributed.

References

1. Alam, M., Rufino, J., et al.: Orchestration of microservices for iot using docker and edge computing. IEEE Communications Magazine **56**(9), 118–123 (2018)
2. Avilés-López, E., García-Macías, J.: Tinysoa: A service-oriented architecture for wireless sensor networks. Service Oriented Comp. and App. **3**, 99–108 (2009)
3. Butzin, B., Golatowski, F., et al.: Microservices approach for the internet of things. In: 21st Int. Conf. on Emerging Tech. and Factory Aut. (ETFA). pp. 1–6 (2016)
4. Cambra, C., Sendra, S., et al.: An iot service-oriented system for agriculture monitoring. In: International Conference on Communications (ICC). pp. 1–6 (2017)
5. Caporuscio, M., Raverdy, P., et al.: ubisoap: A service-oriented middleware for ubiquitous networking. IEEE Transactions on Services Comp. **5**(1), 86–98 (2012)
6. Fowler, M., Lewis, J.: Microservices, a definition (2014), http://martinfowler.com/articles/microservices.html, [Last accessed 12 Aug 2020]
7. Hassanalieragh, M., et al.: Health monitoring and management using internet-of-things (iot) sensing with cloud-based processing: Opportunities and challenges. In: ICSOC 2015. pp. 285–292 (2015)
8. Hossain, M.S., Muhammad, G.: Cloud-assisted industrial internet of things (iiot) - enabled framework for health monitoring. Computer Networks **101**, 192–202 (2016)
9. Issarny, V., Bouloukakis, G., Georgantas, N., Billet, B.: Revisiting service-oriented architecture for the iot: A middleware perspective. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. vol. 9936, pp. 3–17. Springer, Cham (2016)
10. Mongiello, M., Nocera, F., et al.: A microservices-based iot monitoring system to improve the safety in public building. In: SpliTech. pp. 1–6 (2018)
11. Montori, F., Bedogni, L., et al.: A collaborative internet of things architecture for smart cities and environmental monitoring. IEEE Internet of Things Journal **5**(2), 592–605 (2018)
12. Namiot, D., Sneps-Sneppe, M.: On micro-services architecture. Interenational Journal of Open Information Technologies **2**(9), 24–27 (2014)
13. Newman, S.: Building Microservices. O'Reilly Media Inc., 1st edn. (2015)
14. SmartDataCollective: Iot is the most important development of the 21st century (2018), https://www.smartdatacollective.com/iot-most-important-development-of-21st-century, [Last accessed 06 Sep 2020]
15. Soldatos, J.e.a.: Openiot: Open source internet-of-things in the cloud. LNCS **9001**, 13–25 (2015)
16. Sun, L., Li, Y., et al.: An open iot framework based on microservices architecture. China Communications **14**(2), 154–162 (2017)
17. Vresk, T., Čavrak, I.: Architecture of an interoperable iot platform based on microservices. In: MIPRO 2016. pp. 1196–1201 (2016)
18. Zanella, A., Bui, N., et al.: Internet of things for smart cities. IEEE Internet of Things Journal **1**(1), 22–32 (2014)
19. Zhao, Y., Zou, Y., Ng, J., da Costa, D.A.: An automatic approach for transforming iot applications to restful services on the cloud. In: ICSOC 2017. vol. 10601, pp. 673–689. Springer (2017)