

DevOps for IoT Systems: Fast and Continuous Monitoring Feedback of System Availability

Miguel A. López-Peña¹, Jessica Díaz¹, Jorge E. Pérez¹, and Héctor Humanes

Abstract—Current Internet-of-Things (IoT) systems are highly distributed systems, which integrate cloud, fog, and edge computing approaches. Accelerating their maintenance and continuous improvement, while ensuring their availability, is complex. DevOps promotes fast and continuous feedback from operations to development to detect problems before customers are impacted, among other benefits. However, there is not any formal definition of how to do this. This article defines the “fast and continuous monitoring feedback of system availability” activity (F&CF availability) that supports automatic and continuous monitoring feedback from operations to the development of the IoT system availability. This activity has been formalized through the software and systems process engineering metamodel (SPEM). Its implementation is demonstrated in a real scenario that provides evidence that the formalization of the F&CF availability activity helps teams in better diagnosing and fixing outage problems. The result is a distributed and configurable monitoring component developed through code [monitoring as code (MaC)]. This component is embedded in the IoT infrastructure. MaC enables DevOps team to configure their own metrics and indicators at runtime, i.e., monitoring on demand. The formalization of this activity, based on an MaC technique, enables the automation, versioning, and replication of monitoring elements.

Index Terms—Availability, continuous monitoring, DevOps, fast and continuous feedback activity, Internet-of-Things (IoT) systems.

I. INTRODUCTION

NOWADAYS, it is a fact that there are a multitude of Internet-of-Things (IoT) applications in nearly all fields (especially in industry, utility management, health, transportation, sports, etc.). Many of them are deployed in environments, such as smart cities, manufacturing plants, energy distribution sites, hospitals, etc. [1], [2]. Many of those applications are

actually IoT systems that integrate sensors, actuators, computing resources, and communication components to control physical processes or monitor critical tasks [3].

IoT systems (see Fig. 1) can be understood as the integration of: a hardware infrastructure (a set of connected physical devices, such as servers, physical objects, sensors, actuators, etc., and communication networks); a software infrastructure (commonly named the IoT platform or framework) that supports the development, deployment, and execution of end-to-end IoT applications; and a set of applications that offer vertical solutions over those infrastructures.

Many IoT systems provide very important services, sometimes critical services that require an uninterrupted operation or very high levels of availability (in e-health or energy for instance), but their complexity makes them difficult to monitor in depth in order to prevent anomalies and failures [4]. In addition, rapid and continuous feedback to instrument maintenance, updates, and ultimately, meet customer satisfaction requires a multidisciplinary collaboration between development and operations teams, which have to test the systems in production, build new releases, and deploy them [5]. For that purpose, DevOps culture [6] is an approach that facilitates the collaboration between the IT teams [7] and accelerates and improves the cycles of maintenance [8]. DevOps promotes fast and continuous feedback from operations to development to detect problems long before customers are impacted [9]. However, *continuous monitoring* is not an easy task and is even more complex for IoT [10], [11].

In this work, the “fast and continuous monitoring feedback of system availability” activity (F&CF availability activity) is formally specified. The objective of this activity is early monitoring and detecting the availability of IoT systems working in production and to continuously inform about it (feedback) to the DevOps teams—feedback from operations to development. This specification is made using the software and systems process engineering metamodel 2.0 (SPEM 2.0) [12].

One important feature of the “F&CF availability” activity is that it defines the use of a versioned and repeatable configuration of an IoT software infrastructure that we named “monitoring as code” (MaC). MaC is defined as a specialization of infrastructure as code (IaC) for monitoring purposes that enable the implementation, deployment, and execution of customized pieces of code. With this MaC infrastructure, DevOps teams are able to configure their own metrics and indicators at runtime and receive detailed information about the availability of an IoT system in execution, in other words, monitoring on demand.

Manuscript received November 12, 2019; revised April 10, 2020 and June 17, 2020; accepted July 24, 2020. Date of publication July 29, 2020; date of current version October 9, 2020. This work was supported in part by the European Commission’s Horizon 2020 Research and Innovation Programme under Grant 732667 (RECAP, <http://www.recap-project.eu>) and CROWDSAVING under Grant TIN2016-79726-C2-1-R. (Corresponding author: Miguel A. López-Peña.)

Miguel A. López-Peña is with the Department of Innovation and Development, Sistemas Avanzados de Tecnología, S.A., 28023 Madrid, Spain, and also with the Department of Sistemas Informáticos, Universidad Politécnica de Madrid, 28031 Madrid, Spain (e-mail: miguelangel.lopez.pena@alumnos.upm.es).

Jessica Díaz and Jorge E. Pérez are with the Departamento de Sistemas Informáticos, Universidad Politécnica de Madrid, 28040 Madrid, Spain (e-mail: yesica.diaz@upm.es; jorgeenrique.perez@upm.es).

Héctor Humanes is with Department of Innovation, Sistemas Avanzados de Tecnología, S.A., 28023 Madrid, Spain (e-mail: hector.humanes@satec.es).

This article has supplementary downloadable material available at <http://ieeexplore.ieee.org>, provided by the authors.

Digital Object Identifier 10.1109/JIOT.2020.3012763

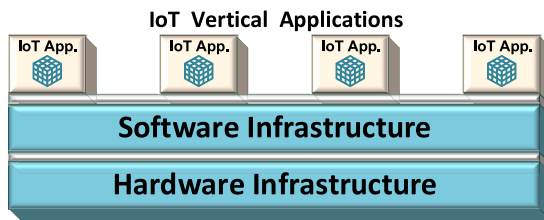


Fig. 1. IoT system structure.

The F&CF availability activity is specified while conforming to the fundamentals of DevOps [13], [14] and enables the quick detection of availability problems and obtains the information to fix them, ideally long before customers are impacted. This work was completed with the implementation of the F&CF availability activity in a real scenario in order to obtain evidence to verify the validity of the activity and its results in practice.

The remainder of this article is structured as follows. Section II briefly introduces the concepts of IoT and DevOps. Section III describes related work. Section IV presents the specification of the F&CF availability activity in SPEM 2.0. An implementation of the activity is described through a case study in Section V. Finally, the results, conclusions, and future works are described in Sections VI–VIII, respectively.

II. BACKGROUND

Standard ISO/IEC 20924 (on the IoT vocabulary) [15] defines the most important terms and concepts of IoT. Standard ISO/IEC 30141 (on the IoT reference architecture) [16] is the first international standard that “specifies a general IoT reference architecture in terms of defining system characteristics, a conceptual model, a reference model, and architecture views for IoT.” ISO/IEC 30141 also defines the concept of “trustworthiness” as one of the main characteristics of an IoT system and specifies that availability is an essential property to support the trustworthiness of IoT systems.

A. IoT Platforms: Cloud, Fog, and Edge Computing

There are many software IoT infrastructures in the market (commonly referred to as IoT platforms or frameworks) [17], [18] that facilitate the development of these systems. Some examples of IoT platforms are FIWARE,¹ SOFIA,² and ThingsBoard,³ which offer services and application programming interfaces (APIs) for the integration and interoperability of multiple devices, as well as the storage, processing, and analysis of a huge volume of data (data analytics). Together with these platforms, a set of IoT services offered by well-known cloud providers (for example, Amazon Web Services IoT,⁴ Google Cloud IoT,⁵ IBM Watson IoT,⁶ and Microsoft Azure IoT)⁷ have emerged in the last few

years. However, the adoption of the fog/edge computing paradigm [19], [20] is also growing, and most authors accept that fog/edge computing features should also be included in the definition of new IoT architectures [21]–[25]. There are already commercial examples of such infrastructures as, for example, Cisco Kinetic.⁸ These new IoT infrastructures that integrate cloud, edge, and fog computing are improving productivity and are addressing important aspects, such as big data management and high computational requirements [26]. However, they are much more complex and distributed and, therefore, are also much more difficult to monitor and ensure their availability.

B. Why Is DevOps Important for IoT?

IT operations monitoring is an essential activity to guarantee reliable and compelling IoT systems. So far, the teams that develop IoT systems and IT operations work in silos, i.e., each one has its own objectives (usually different of business objectives), processes, methodology, tools, managers, etc. These silos and the lack of collaboration hamper rapid and continuous feedback to instrument maintenance, updates, and ultimately, meet customer satisfaction and impact on the business. In 2008, Debois was one of the first voices on highlighting the need of resolving the conflict between development and operations teams when they have to collaborate to provide quick response time to customer demands [27]. As a response to this need, DevOps emerged as a cultural approach that promotes the development team to work closely and efficiently with the operations team [7]. To create this culture, specific practices are needed. According to [9], *fast and continuous feedback from operations to development* is one of these fundamental practices. This means to create short feedback loops that enable detect errors or weaknesses of systems in production and correct them quickly in a continuous improvement cycle [28], [29]. Balaile *et al.* [30] identified “continuous monitoring” as a critical DevOps practice that provides developers with performance-related feedback and facilitates detecting any operational anomalies. This helps to bridge the gap between development and operations and change the team structures [30]. In a previous work [31] we focused on self-service cybersecurity monitoring as an enabler to introduce security practices in a DevOps environment.

However, continuous monitoring is not an easy task and is even more complex for IoT. Some researchers [10], [11] applied systematic literature review and multiple case study, respectively, to identify the main challenges to adopt DevOps to the embedded systems domain, in contrast to its widespread use in the Web domain. On the one hand, there are technical challenges as the high distribution of IoT systems (cloud, fog, and edge approaches). On the other hand, people who are not used to work together have to do it, being necessary to define new processes, activities, and automate them. Dealing with these challenges, we are enabling developers to monitor the systems they build, on demand, self-service, and thus, enabling fast and continuous feedback. In the related work section, we mention some initial advances on DevOps in IoT systems,

¹<https://www.fiware.org>

²<http://sofia2.com/>

³<https://thingsboard.io/>

⁴<https://www.amazonaws.cn/en/iot-core/>

⁵<https://cloud.google.com/solutions/iot/>

⁶<https://www.ibm.com/internet-of-things>

⁷<https://azure.microsoft.com/en-us/product-categories/iot/>

⁸<https://www.cisco.com/c/en/us/solutions/internet-of-things/iot-kinetic.html>










Icon	Name	Description
	Activity	It is a Work Breakdown Element and Work Definition that defines basic units of work within a Process as well as a Process itself.
	Composite Role	It is a grouping of Role Definitions that can be used in an Activity or Process to reduce the number of Roles defined in Method Content.
	Metric	It is a special Describable Element that contains one or more constraints that provide measurements for any Describable Element
	Step	It is a Section and Work Definition that is used to organize a Task Definition's Content Description into parts or subunits of work.
	Task Definition	It is a Method Content Element and a Work Definition that defines work being performed by Roles Definition instances
	Task Use	It is a Method Content Use and Work Breakdown Element that represents a proxy for a Task Definition in the context of one specific Activity
	Tool Definition	It is a special Method Content Element that can be used to specify a tool's participation in a Task Definition.
	Work Product Definition	It is Method Content Element that is used, modified, and produced by Task Definitions.
	Work Product Use	It is a special Breakdown Element that either represents an input and/or output type for an Activity or represents a general participant of the Activity.

Fig. 2. SPEM 2.0 graphical notation.

although most of the work focuses on automating the deployment and delivery, being continuous monitoring a challenge today. In this direction, Ferry *et al.* [32] presented challenges related to the development, operation, and quality assurance of trustworthy smart IoT systems. They present a research roadmap to enable DevOps in smart IoT systems and envision a DevOps framework to support them, although without specific details about how to support continuous monitoring.

To deal with this challenge, this article formalizes the activity for *fast and continuous feedback* to detect problems in advance, facilitating better-informed decisions in order to fix them. We provide the basis for automating this activity through the MaC so that developers can easily configure monitoring infrastructure, on demand and thus, obtain fast feedback to develop system more reliable.

III. RELATED WORK

Recently, some works approach the development of IoT systems based on DevOps practices. Most of them focus on automating the deployment and delivery of systems. For example, Karapantelakis *et al.* [33] described a system for automated lifecycle management of IoT applications requiring cellular network access. This system supports DevOps by automating the deployment pipeline of IoT applications, in other words, the system automates allocation and deallocation of network and cloud resources based on the information provided by a monitoring infrastructure of the network, CPU, and memory status. Similarly, Kim and Kim [34] described a system for data operation visibility. Bae *et al.* [35] focused on automated continuous integration and deployment of IoT cloud services using containers. Syed and Fernandez [36] described a cloud/fog ecosystem to support both IoT and DevOps in

the automated management of infrastructure but not explicitly together.

As far as we know, only [33] and [34] address monitoring infrastructures for IoT systems to convey feedback from operations to development. Nevertheless, our contribution differs from these previous ones in some aspects: 1) the monitoring infrastructure is created and configured automatically, it is versioned and repeatable, and easily adaptable to changes in production (monitoring as code, or MaC) and 2) we introduce the concept of custom monitoring on demand (managed as MaC components) as pieces of code for monitoring purposes that can be coded, deployed, and executed in the IoT system in production to improve the feedback from operations to development.

IV. FAST AND CONTINUOUS FEEDBACK OF SYSTEM AVAILABILITY (F&CF AVAILABILITY)

A. “F&CF Availability” Activity Definition

A key practice in DevOps is the fast and continuous feedback from operations to development. In this work, we have promoted this practice in large IoT systems in production for availability monitoring. For this purpose, we provide a formal specification of the F&CF availability activity using SPEM 2.0 [12] (see some SPEM 2.0 graphical notations in Fig. 2).

Fig. 3 shows the SPEM specification for the activity of the principle “fast and continuous feedback from operations to development” and the instantiation for the case of system availability monitoring. The activity specification is composed of three *TaskUse* elements, each of which is described in its corresponding *TaskDefinition* (through the relation $\langle\langle\text{content trace}\rangle\rangle$). Each *TaskDefinition*, as a *method content element*,

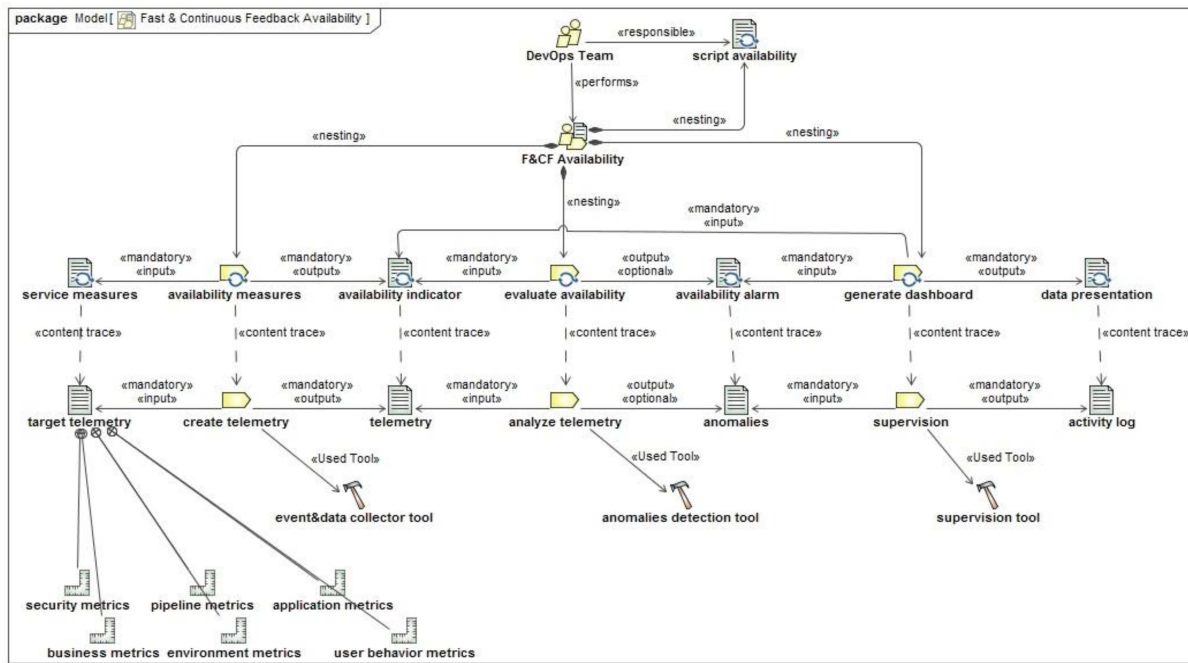


Fig. 3. Specification (SPEM 2.0) of the “F&CF availability” activity.

provides step-by-step explanations that describe how specific development goals are achieved independent of the placement of these steps within a development lifecycle. Each *TaskUse*, as a *process element*, takes these *method content elements* and relates them to partially ordered sequences that are customized to specific types of project.

The specification of the F&CF availability activity defines the following set of TaskDefinition elements (see Fig. 3).

1) *TaskDefinition “Create Telemetry”*: This task reads the values of the monitoring metrics specified in the WorkProductDefinition *target telemetry* and generates a new WorkProductDefinition named *telemetry* with the collected data. Table I shows the summarized formal description of this task in SPEM 2.0.

2) *TaskDefinition “Analyze Telemetry”*: This task is responsible for detecting anomalies and failures from the values of the metrics that the WorkProductDefinition *telemetry* collects. The set of detected anomalies is sent to the *anomalies* artifact.

3) *TaskDefinition “Supervision”*: The function of this task is to diagnose problems (from the *anomalies* artifact) and send the feedback to the development team with details of the anomalies and possible solutions to fix them (e.g., deploying a new version of the software for some of the monitored elements). This task will generate appropriate events and messages to help plan corrective actions. These events are described in the *activity log* artifact.

The main input for the F&CF availability activity through the TaskDefinition *create telemetry* is the WorkProductDefinition called *target telemetry*, which contains a complete definition of the elements of the IoT system that must be monitored and how to accomplish this. The artifact *target telemetry* is defined as associations of several *metric elements* (environment metrics, application metrics, pipeline

metrics, etc.) (Fig. 3). Table II shows a summary of the “environment metrics” specification in SPEM 2.0.

The F&CF availability activity is carried out by the DevOps Team, which performs both development and operation works. This role has been specified as an instance of *CompositeRole* whose main responsibility, in this case, is the generation of the WorkProductUse *script availability* that implements the MaC in this proposal.

Finally, the SPEM specification (Fig. 3) outlines a set of tools that support the TaskDefinition elements. These tools are part of a monitoring infrastructure (*ToolDefinition*) that is flexible enough so that DevOps teams can develop their own monitoring and alerting services according to their criteria (monitoring on demand). In order to automatize the deployment of the monitoring tools and the monitoring on demand components (management of code and configurations of monitoring, alarms, and alerts/events), we have defined a set of configurable scripts which we call MaC as a specialization of the well-known IaC. All those code and configuration components are versioned and configured, and they are repeatable. Virtualization and containerization technologies are used for the automation of the monitoring infrastructure. This automation is described in detail in the implementation of the F&CF availability activity.

B. “F&CF Availability” Instantiation

The instantiation of F&CF availability is specified by defining the TaskUse and WorkProductUse elements as shown in Fig. 3, and selecting the tools for the *event&data collector*, *anomalies detection*, and *supervision*. There are different monitoring tools (Prometheus,⁹ Pandora FMS,¹⁰ Netdata,¹¹

⁹<https://prometheus.io/>

¹⁰<https://pandorafms.com/>

¹¹<https://www.netdata.cloud/>

TABLE I

SPEM SPECIFICATION OF THE “CREATE TELEMETRY” TASK DEFINITION

Attribute	Value
name (from UML 2 Infrastructure)	Create Telemetry
presentationName (from Describable Element.ContentDescription)	create telemetry
Kind (from Extensible Element.Kind)	Discipline: feedback Ops2Dev
briefDescription (from Describable Element.ContentDescription)	This task defines how to get the metrics, and how these metrics are stored.
mainDescription (from Describable Element.ContentDescription)	This task is responsible for reading the values of those elements that have to be monitored and that are indicated in the WorkProductDefinition <i>target telemetry</i> . It generates a new <i>telemetry</i> artifact in which the read values appear (at the indicated intervals).
purpose (from Describable Element.ContentDescription)	To gather metrics from elements of an IoT system.
ownedTaskDefinitionParameter (Default_TaskDefinitionParameter)	This is a composition association that references the WorkProductDefinitions related with this task. In this case, <i>target telemetry (mandatory, input)</i> and <i>telemetry (mandatory, output)</i> .
usedTool: ToolDefinition	event&data collector tool
/step: Step	
Precondition (from Work Definition)	“target telemetry” artifact exists
Postcondition (from Work Definition)	“telemetry” artifact has been created

TABLE II

SPEM SPECIFICATION OF THE “ENVIRONMENT METRICS” METRIC

Attribute	Value
name	environment metrics
presentationName	Environment Metrics
Kind	Metric
briefDescription (from Describable Element.ContentDescription)	Defines the family of metrics related to the IoT physical infrastructure on which the system is deployed.
mainDescription (from Describable Element.ContentDescription)	<p>The taxonomy of environment metrics and their measures follow the schema:</p> <ul style="list-style-type: none"> ➤ Infrastructure metrics: refer to hardware and data links: <ul style="list-style-type: none"> • CPU Metrics: <ul style="list-style-type: none"> ○ CPU Usage (%) ○ ... • Memory Metrics: <ul style="list-style-type: none"> ○ RAM usage (Gb) ○ ... • Data Network Metrics: <ul style="list-style-type: none"> ○ Physical Network Interfaces ○ Aggregated Bandwidth (Kb/s) ○ Bandwidth (Kb/s) ○ Packets (packets/s) ○ Interface Errors (errors/s) ○ Interface Drops (drops/s) ○ ... • Disk Metrics: <ul style="list-style-type: none"> ○ I/O operations (operations/s): ○ Space usage (bytes) ○ I/O bandwidth/s (Kb/s) ○ Utilization (%) ○ ... ➤ IoT service metrics: the IoT infrastructure is designed as an architecture of microservices, all of them encapsulated in containers: <ul style="list-style-type: none"> • http.server.requests (#) • process.start.time (s) • process.uptime (s) • process.cpu.usage (%) • process.files.open (#) • system.load.average.1m (#) • system.cpu.count (#) • system.cpu.usage (%) • ... ➤ Device metrics: <ul style="list-style-type: none"> • Device status (On/Off) • Device error (Error Code) • ... ➤ ...

Zenoss,¹² Azure,¹³ AWS¹⁴ monitoring tools, etc.) and any of them could have been selected and instantiated. In this work, open-source Netdata tool has been selected because it is distributed and light, enables deployment and monitoring in both cloud, edge, and virtualized environments, and supports monitoring and event management as code (MaC).

The instantiation of F&CF availability activity includes the following elements.

¹²<https://www.zenoss.com/>

¹³<https://docs.microsoft.com/azure/azure-monitor/>

¹⁴<https://aws.amazon.com/cloudwatch/>

1) *Service Measures:* It defines the availability metrics to be collected in the instantiated scenario (IoT system). This instance defines the “environment metrics” related to availability in three scopes: 1) infrastructure metrics; 2) IoT service metrics; and 3) device metrics.

2) *Availability Measures:* This TaskUse collects the measures related to the availability specified in *service measures* and calculates availability indicators by means of functions that combine sets of measures applying predefined logic. It leaves the value of those measures and indicators in the WorkProductUse *availability indicator*. The steps are as follows.

- 1) Develop monitoring components (code in the Netdata plugin files) that collect the measures defined in *service measures* following the design model defined by the monitoring tool.
- 2) Configure the monitoring components in the monitoring tool by editing and modifying the configuration file `netdata.conf` defined for this purpose.
- 3) Restart the monitoring tool services (warm restart).
- 3) *Availability Indicator*: This `WorkProductUse` receives and stores the entire set of values of the measures and indicators.
- 4) *Evaluate Availability*: This `TaskUse` receives the values of measures and indicators stored in the `WorkProductUse` *availability indicator* and applies a set of rules and logic that define the grade of availability of the IoT system, generates availability alerts and events, and produces the detailed information that will be sent to the DevOps team. This detailed information will be the feedback of the availability status of the IoT system in production (measures and indicator values, alerts and events generated, and their severity levels) to be sent to the DevOps team to anticipate anomalies and fix failures. The results of this task are sent to *the availability alarm* `WorkProductUse`. The steps are as follows.
 - 1) Develop the alarm components (code in the Netdata alarm definition files) that evaluate the IoT system availability, following the design model defined by the monitoring tool.
 - 2) Configure the alarm components in the monitoring tool by editing and modifying the configuration file `netdata.conf`.
 - 3) Restart the monitoring tool services (warm restart).
 - 5) *Availability Alarm*: This `WorkProductUse` is the repository that receives detailed information (indicators and evaluation results) from the *evaluate availability* task.
 - 6) *Generate Dashboard*: This `TaskUse` implements two actions: 1) generate the dashboard Web application retrieving measures, indicators, and their analysis stored in “availability indicator” and *availability alarm*, respectively and 2) send detailed feedback to the DevOps team through direct messages (e-mail, instant messaging, etc.) and/or as tickets to other event monitoring systems (third-party applications). The steps are as follows.
 - 1) Configure the dashboard in the monitoring tool by editing and modifying the configuration file defined for this purpose. This configuration includes the URL to show the dashboard in a Web browser.
 - 2) Configure the event management in the monitoring tool by editing and modifying the configuration file `netdata.conf` or programming *ad hoc* components to send messages and events that are called from the alarm components.
 - 3) Restart the monitoring tool services (warm restart).
 - 7) *Data Presentation*: This `WorkProductUse` is the Web application generated by the monitoring tool that shows the IoT system availability dashboard. This application is accessible through a Web browser in a specific URL previously configured in the tool.

V. CASE STUDY: IMPLEMENTATION OF “F&CF AVAILABILITY” ACTIVITY

The SPEM specification of the F&CF availability activity has been validated by implementing it in a real IoT scenario. The activity implementation and the results of the case study are detailed as follows.

A. Case Study Description

The scenario for the case study was a drinkable water supply system. This system consisted of three flow meters that could be opened/closed on demand and that send the state of the current metering to the IoT platform.

The flow meters were installed in the water entry points of three different sets of buildings. Each set of buildings was a closed workplace where all the workers had everything they needed to work and live. The workplace consisted of offices, factories, canteens, houses, and warehouses, and could be seen as a sort of small smart city, where systems such as the drinkable water supply could be monitored using IoT capabilities. The IoT capabilities of the system helped the facility management team to provide a quality service since they had fine-grained information in real time. With the information provided by the system, they could forecast system failures even before they happened and act in advance to avoid supply interruptions among other benefits.

The flow meters measured the amount of water flowing through them and sent this data to third-party application through the IoT infrastructure. The flow meters also had the capability to stop and start the water flow and report this status in conjunction with the water flow measures.

B. Case Study Technical Environment

For the development of this case study, we deployed RECAP’s SAT-IoT platform [37]. This platform was designed and built using a distributed microservice architecture composed of multiple small components working together.

The flow meter devices were configured to send their data to the SAT-IoT collector (one of the main entities of the platform built as a microservice and deployed in a container) using the MQTT protocol over TCP. This IoT collector also provided an endpoint where the user could obtain metrics, such as data received, errors in the data, and data acquisition performance.

This case study focused on the monitoring of the availability of this IoT system. The availability was studied from the hardware infrastructure level that includes the status of servers and devices, going through the IoT software infrastructure level, and to the application level. To perform this study, the Netdata monitoring tool was chosen as an implementation of the *event&data collector*, *anomalies detection*, and *supervision* tools defined in the F&CF availability activity specification (Fig. 3). This tool was designed as highly modular software to perform all-in-one metrics analysis. The Netdata configurations, plugins, and event/alarm managers can be created, extended, or modified to fit the DevOps team needs since almost behavior in Netdata can be coded. This is key for enabling MaC and supporting the automation that

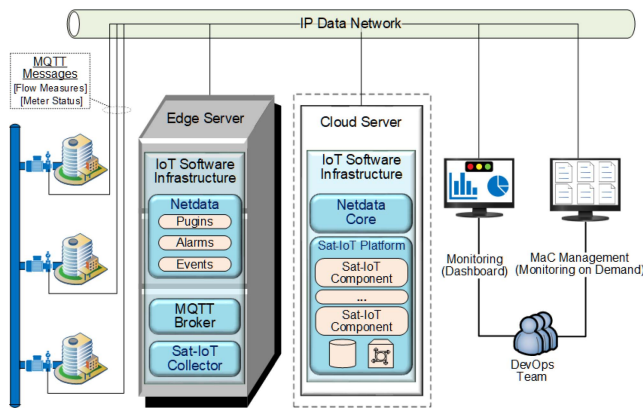


Fig. 4. Case study IoT system overview.

the F&CF activity aims. These plugins and configurations are managed via a version control system and are deployed using an automation tool like Ansible¹⁵ among others. Fig. 4 shows an overview of the IoT distributed system that was deployed in this case study, in which the SAT-IoT collector, Netdata plugins, alarm managers, and event managers were deployed in an edge node, while the Netdata core and the other IoT software infrastructure components were in cloud servers.

C. Implementation of “F&CF Availability”

Once the IoT infrastructure was deployed, the next step was the development of the F&CF availability activity and its deployment and execution on the IoT system. To that end, the *TaskUse* elements specified in Section IV-B were implemented *ad hoc* as MaC components and configured to be installed and executed in Netdata. The implementation consists of a set of configuration files, scripts, and python code files, all of them under GitLab¹⁶ CI/CD pipeline, configuration, and version control. The management of deployments and executions is carried out using playbooks that Ansible executes in the selected servers. All this is the *WorkProductUse script availability* generated by DevOps team to implement the MaC. Fig. 5 shows the sequence of steps performed by *script availability*.

The components we developed are described as follows.

1) *TaskUse “Availability Measures”*: This *TaskUse* was implemented by deploying the Netdata tool in the server, specifying Netdata configuration files for measuring availability, developing the plugins, and executing them with Ansible. Code 1 shows an excerpt from the configurations file. When the Netdata tool was configured and was running, custom plugins were coded using the Python programming language. In this case, two *ad hoc* plugins were developed (*flow_meters_collector.py* and *API_check_collector.py*) to retrieve information from the SAT-IoT collector [37] via the API REST endpoint. The information retrieved included the flow meter measures, the state of the valve, the amount of data sent, etc. Some code sequences of the plugins implemented are shown in Codes 2 and 3.

Code 1 Netdata Configuration File

```
...
[global]
...
plugins directory =
    "/usr/libexec/netdata/plugins.d"
"/etc/netdata/custom-plugins.d"
...
[web]
...
[plugins]
PATH environment variable =
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
r/sbin:/usr/local/bin:/usr/local/sbin:
PYTHONPATH environment variable = /opt/python/bin
proc = yes
enable running new plugins = yes
check for new plugins every = 60
apps = yes
charts.d = yes
python.d = yes
[health]
...
[registry]
...
[backend]
...
[statsd]
...
...
```

Since this case study aimed to evaluate the availability of the IoT system, it was necessary to provide a set of indicators needed to calculate the availability. The calculation of these indicators was coded and deployed in the same way as the plugins, and they were also managed as MaC components and coded in Python. In the case study, several indicators were created.

- 1) System availability indicators combined CPU, RAM, disk, and network usage metrics of hardware infrastructure. In addition, multiple indicators related to devices and data sent from them were developed to reflect availability. The first one was the error indicator. This indicator was used to raise an alarm if a valve was supposed to be closed, but the water was flowing through the flow meter. The second one was a request indicator to alert that the number of responses sent from device is less than the number of requests received. The last one was the flow threshold, which indicates that the current water flow is exceeding the manufacturer's recommended limits.
- 2) Software infrastructure (Sat-IoT platform) availability indicators relied on the responses (or the lack of them if a component was down) from the IoT software infrastructure.
- 2) *TaskUse “Evaluate Availability”*: This *TaskUse* examines indicators, calculates the availability at different levels, and generates feedback about the behavior of the system and raise alarms.

In this case study, the availability is calculated at two levels: 1) component level (system environment, SW infrastructure, and data/device) and 2) indicator level (for example, error, request, and threshold are the indicators of data/device component), and they are aggregated to calculate availability of

¹⁵<https://www.ansible.com/>

¹⁶<https://about.gitlab.com>

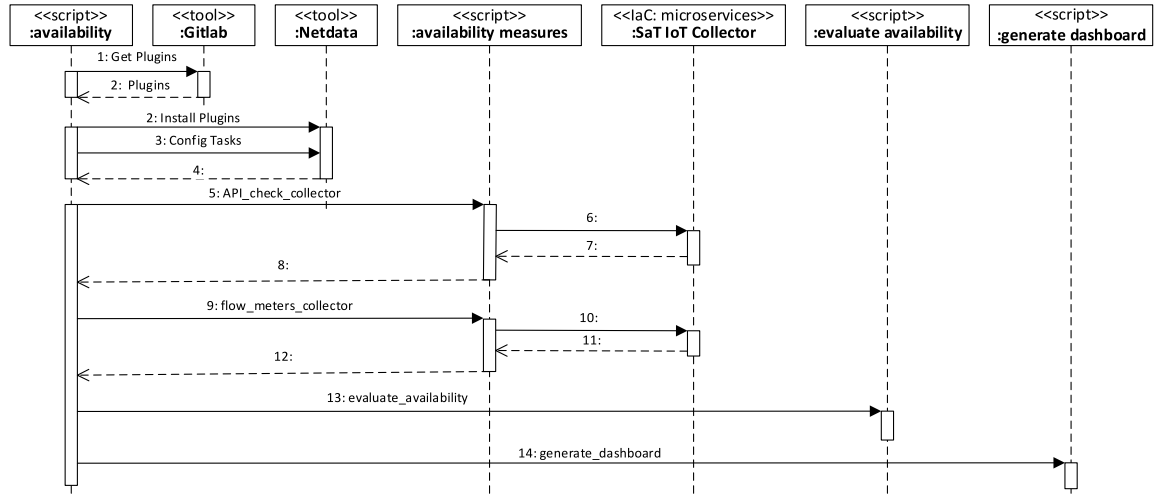


Fig. 5. Sequence diagram of “script availability.”

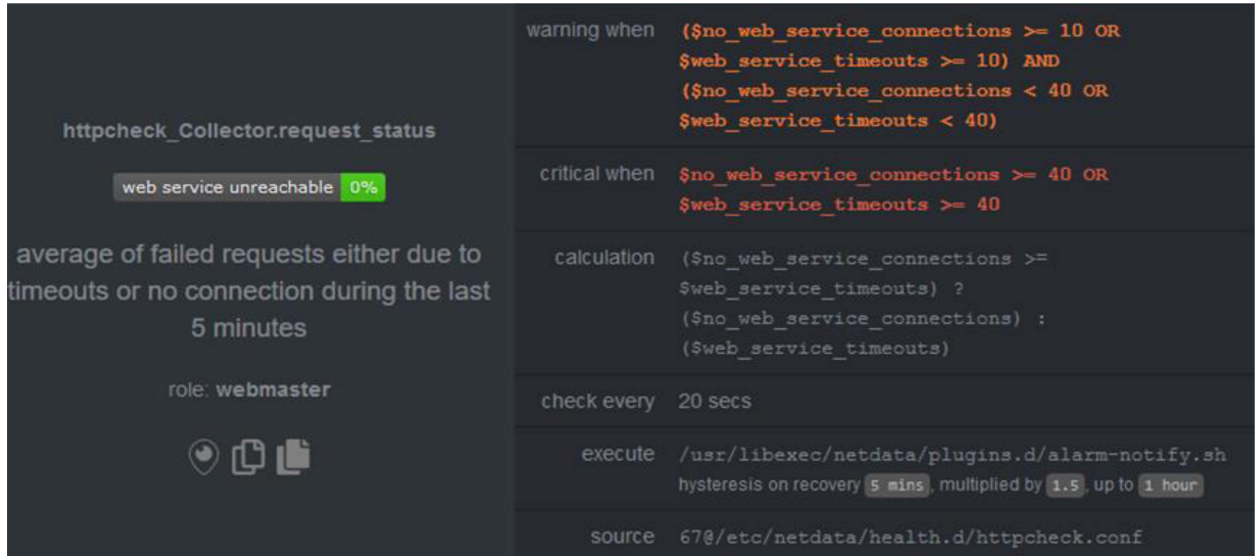


Fig. 6. IoT software infrastructure alarm.

the whole IoT system. The availability $A_I(C_i, I_j)$, where I_j is an indicator that belongs to the component C_i , is calculated as in (1), the availability of each component $A_C(C_i)$ is calculated as in (2), and the availability of the system $A_{\text{system}}(C_i)$ is calculated as in (3). This model, expresses the percentage of time that a component/system is working properly and it is computed by well-known equation which relates mean time to repair (MTTR) and mean time to failure (MTTF) [38]

$$A_I(C_i, I_j) = \frac{\text{MTTF}(C_i, I_j)}{\text{MTTF}(C_i, I_j) + \text{MTTR}(C_i, I_j)} = \frac{\text{Uptime}(C_i, I_j)}{\text{Uptime}(C_i, I_j) + \text{Downtime}(C_i, I_j)} \quad (1)$$

$$A_C(C_i) = \prod_j A_I(C_i, j) \quad (2)$$

$$A_{\text{system}}(C_i) = \prod_i A_C(C_i). \quad (3)$$

This TaskUse was coded in Netdata using the graphic alarm configuration system shown in Fig. 6. This system provides a programmable rule system that enables make calculations, define rules to raise alarms, set the alarm severity, and also execute a process associate to the alarm—in this use case, a feedback generator that sends detailed information about indicators and behavior of the system.

3) *TaskUse “Generate Dashboard”*: This TaskUse is the key piece of the activity for the de DevOps team since this TaskUse provides a rich and tailored feedback from the system in production, including measures, indicators, alarms, and detailed notification messages about the behavior of the system. The TaskUse *generate dashboard* is the customized configuration of the dashboard and feedback system in this case study. This customization was made using the Netdata configuration file which contained the charts definition and location, the order of the elements, the scale of the data, and all the parameters that are shown in the dashboard. The notification actions were also defined through the alarm



Fig. 7. Dashboard panel with flows and threshold indicators.

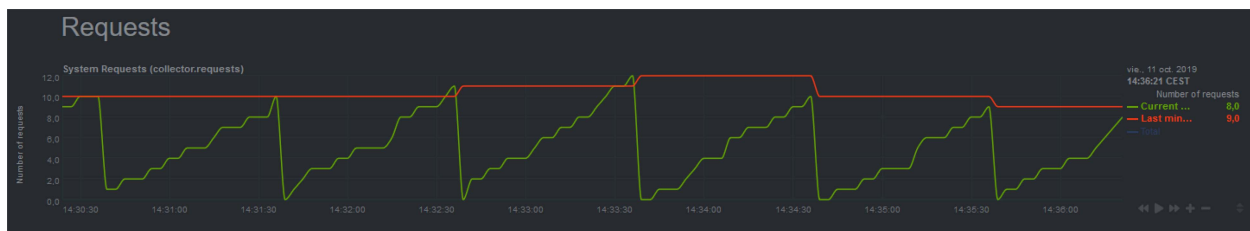


Fig. 8. Request indicator graph.

configuration system described in the configuration of *evaluate availability*. The customization of the dashboard is specified and designed by the DevOps team so that it is adapted to the specific environment monitored and helps the team to interpret the information shown and to take the necessary actions in the case of system anomalies or failures. Code 4 shows a fragment of the configuration code for generating the dashboard.

D. Case Study Results

The result of this case study is a complete implementation of the F&CF availability activity that support the DevOps team in the process of continuous improvement. DevOps team manages both monitoring infrastructure and monitoring components under a CI/CD model and the deploy of new versions is fully automated.

This implementation is an example of MaC that consists of both the installation of the tool (Netdata) and a set of configurations and customized pieces of code that were developed specifically for the IoT system to be monitored. All these components are controlled, tested, and versioned under a CI/CD by the DevOps team, which is able to change them throughout the IoT application life cycle. In addition, any new version of the whole monitoring system can be deployed and run automatically through virtualization and containerization technology as it is specified in *script availability* component in the F&CF availability activity specification.

Figs. 7–9 show only some views of the very large and detailed dashboard designed. For example, Fig. 7 is a view of the panel that shows the time series of water flows through

Code 2 Flow_Meters_Collector.py Plugin

```
...
# the main loop
count = 0
last_run = next_run = now = get_millis()
while True:
    if next_run <= now:
        count += 1
        # DATA GATHERING
        resp = requests.get('http://localhost:8086/actuator/info')
        if resp.status_code != 200:
            # This means something went wrong.
            raise ApiError('GET /actuator/info
            {}'.format(resp.status_code))
        metrics = json.loads(resp.content)
        #print "Stats:
"+str(metrics["UsageMetrics"]["CurrentMinuteRequests"])
        value1 = metrics["UsageMetrics"]["CurrentMinuteRequests"]
        value2 = metrics["UsageMetrics"]["LastMinuteRequests"]
        value3 = metrics["UsageMetrics"]["TotalRequests"]
        value4 = metrics["UsageMetrics"]["FlowMeter1"]
        value5 = metrics["UsageMetrics"]["FlowMeter2"]
        value6 = metrics["UsageMetrics"]["FlowMeter3"]
        value7 = metrics["UsageMetrics"]["DirtyData1"]
        value8 = metrics["UsageMetrics"]["DirtyData2"]
        value9 = metrics["UsageMetrics"]["DirtyData3"]
...

```

the flowmeters and the threshold indicator (green in normal conditions a red when the evaluation of this indicator raises an alarm). Fig. 8 shows a real-time detailed graph with current (green line) and cumulative values (red line) of requests sent from the system to flowmeters, with which the monitoring system calculates request indicators and raises the corresponding alarms. Fig. 9 displays one of the views of the

Code 3 API_Check_Collector.py Plugin

```

class Service(UrlService):
    def __init__(self, configuration = None, name = None):
        UrlService.__init__(self, configuration = configuration, name = name)
        self.order = ORDER
        self.definitions = CHARTS
        pattern = self.configuration.get('regex')
        self.regex = re.compile(pattern) if pattern else None
        self.status_codes_accepted =
self.configuration.get('status_accepted', [200])
        self.follow_redirect = self.configuration.get('redirect', True)
    def _get_data(self):
        data = dict()
        data[HTTP_SUCCESS] = 0
        data[HTTP_BAD_CONTENT] = 0
        data[HTTP_BAD_STATUS] = 0
        data[HTTP_TIMEOUT] = 0
        data[HTTP_NO_CONNECTION] = 0
        url = self.url
        try:
            start = time()
            status, content = self._get_raw_data_with_status
                (retries = 1 if self.follow_redirect else False,
                 redirect = self.follow_redirect)
            diff = time() - start
            data[HTTP_RESPONSE_TIME] = max(round(diff * 10000), 0)
            self.debug('Url: {url}. Host responded with status code {code} in
                {diff} s'.format(url = url, code = status, diff
= diff))
            self.process_response(content, data, status)
            except urllib3.exceptions.NewConnectionError as error:
                self.debug('Connection failed: {url}. Error:
{error}'.format(url = url, error = error))
                data[HTTP_NO_CONNECTION] = 1
            except (urllib3.exceptions.TimeoutError,
urllib3.exceptions.PoolError) as error:
                self.debug('Connection timed out: {url}. Error:
{error}'.format(url = url, error = error))
                data[HTTP_TIMEOUT] = 1
            except urllib3.exceptions.HTTPError as error:
                self.debug('Connection failed: {url}. Error:
{error}'.format(url = url, error = error))
                data[HTTP_NO_CONNECTION] = 1
            except (TypeError, AttributeError) as error:
                self.error('Url: {url}. Error: {error}'.format(url = url, error = error))
                return None
            return data
    def process_response(self, content, data, status):
        data[HTTP_RESPONSE_LENGTH] = len(content)
        self.debug('Content: \n\n{content}\n'.format(content = content))
        if status in self.status_codes_accepted:
            if self.regex and self.regex.search(content) is None:
                self.debug('No match for regex "{regex}"
                    found'.format(regex = self.regex.pattern))
                data[HTTP_BAD_CONTENT] = 1
            else:
                data[HTTP_SUCCESS] = 1
            else:
                data[HTTP_BAD_STATUS] = 1
        ...

```

dashboard panel that monitors the flowmeters error indicator, this panel uses three graphical gauges that change their color to red when the evaluation of the error level raises an alarm.

In addition to the graphical dashboard, the monitoring system sends detailed feedback messages systematically. Table III shows some of these detailed messages containing indicators, alerts, availability values, and text messages that allow the DevOps team to know the cause and severity of the failures and make decisions fast.

Code 4 Dashboard Generation Code

```

...
parser = argparse.ArgumentParser(description = 'Dashboard for the collector
demo')
parser.add_argument('update_every', type = int, nargs = '?',
                    help = 'update frequency in seconds')
args = parser.parse_args()
# internal defaults for the command line arguments
update_every = 1
# evaluate the command line arguments
if args.update_every != None:
    update_every = args.update_every
# various preparations
update_every * = 3000
get_millis = lambda: int(round(time.time() * 1000))
# generate the charts in dashboard
try:
    # type.id name title units family context? chartType priority updateFre-
quency
    db_conf ('CHART collector.requests requests "System Requests"
        "Number of requests" "Requests" "collector.requests"
        line 1001 %s\n' % int(update_every / 1000))
    db_conf ('DIMENSION value1 "Current minute" absolute 1 1\n')
    db_conf ('DIMENSION value2 "Last minute" absolute 1 1\n')
    db_conf ('DIMENSION value3 "Total" absolute 1 1\n')
    db_conf ('CHART collector.flowmeter1 flowmeter1 "Flow meter 1" "m3/h"
        "Flow meters" "collector.flowmeter"
        area 100000 %s\n' % int(update_every / 1000))
    db_conf ('DIMENSION value4 "Current flow" absolute 1 1\n')
    db_conf ('CHART collector.flowmeter2 flowmeter2 "Flow meter 2" "m3/h"
        "Flow meters" "collector.flowmeter"
        area 100000 %s\n' % int(update_every / 1000))
    db_conf ('DIMENSION value5 "Current flow" absolute 1 1\n')
    ...
    db_conf ('CHART collector.dirtydata1 dirtydata1 "Dirty data 1" "Failures"
        "Dirty data" "collector.dirtydata1"
        area 100000 %s\n' % int(update_every / 1000))
    db_conf ('DIMENSION value7 "Dirty data 1" absolute 1 1\n')
    db_conf ('CHART collector.dirtydata2 dirtydata2 "Dirty data 2" "Failures"
        "Dirty data" "collector.dirtydata2"
        area 100000 %s\n' % int(update_every / 1000))
    db_conf ('DIMENSION value8 "Dirty data 2" absolute 1 1\n')
    ...
    ...
    db_conf ("flush")
except IOError as e:
    db_conf(err = 'Failed to send data to netdata\n')
    sys.exit(0)
...

```

Fig. 10 shows an additional panel that was built *ad hoc* to monitor one of the flow meters during a maintenance process. The upper part of the panel shows two combined graphs (one line graph represented in the ordinate left axis and several bar graphs represented in the ordinate right axis). The line graph shows the water flow measurements sent by the flow meter device during the process in which anomalies, such as missing data, erroneous values, etc. are seen. The bar graphs enrich the previous information since it represents in the same time interval different types of events and failure alarms detected by the monitoring system, as well as the evolution of the availability of the device. The bottom of the panel shows the console that receives and displays periodically the feedback messages which improve even more the interpretation of the system dashboard, and the analysis tasks.

As a result, DevOps team can focus the analysis on the key factors of the system with the *ad hoc* availability monitoring implementation because they had more detailed information and multiple general and specific views. In parallel, DevOps

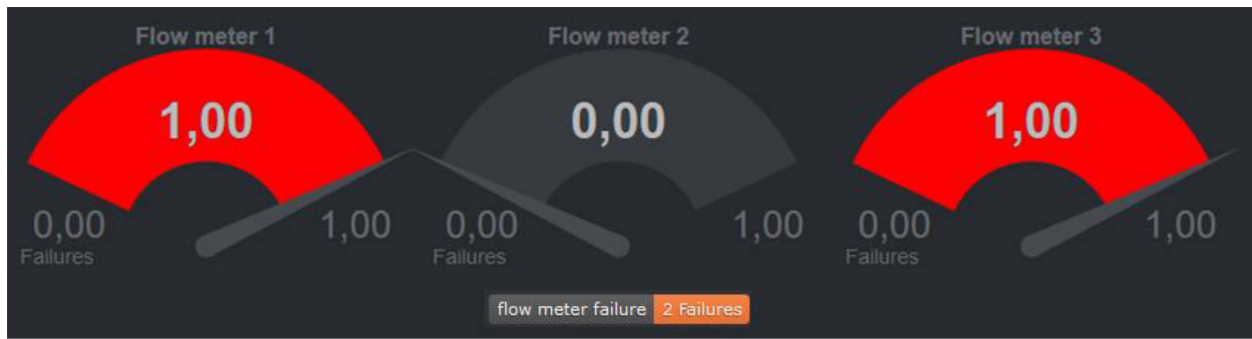


Fig. 9. Flowmeter error indicator panel in the dashboard.

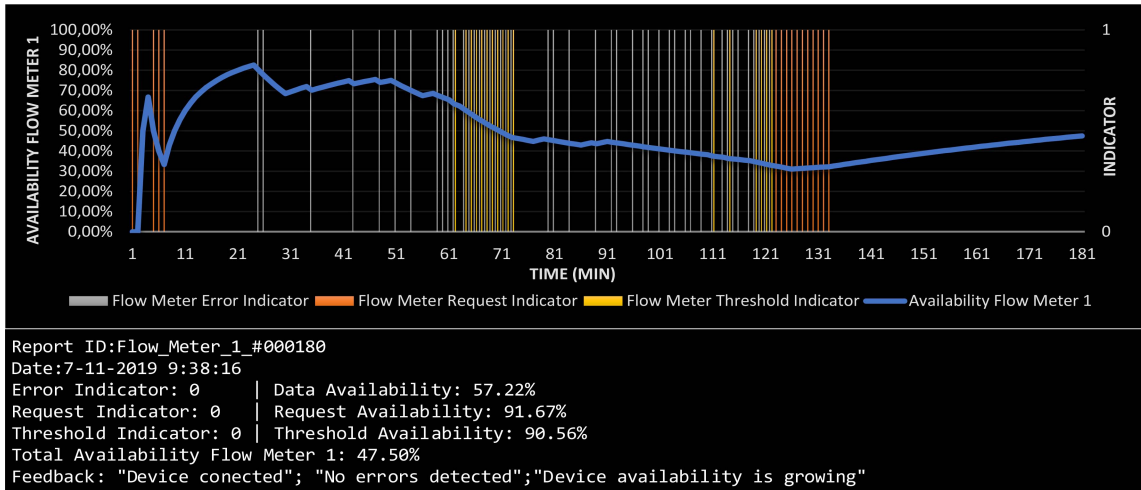


Fig. 10. Custom dashboard for a maintenance process.

team is receiving a detailed feedback of the system in production.

VI. DISCUSSION

The results achieved at the end of this work and the main contributions provided are the following.

A. Definition and Formal Specification of the “F&CF Availability” Activity for IoT Systems

It promotes the practice “fast and continuous feedback from operations to development”. Specifically, we focused on monitoring availability, as this is one of the main features of IoT systems. This formal specification has been made using SPEM 2.0.

B. Definition of MaC and Its Management

It provides a set of predefined components (configuration files, code components, and scripts corresponding to the “TaskUses” defined in the F&CF availability activity specification) under an extension of the IaC model described in this work. With these components, DevOps teams can create monitoring and analysis executable components on demand by code that are versioned, replicable, and adaptable, instead of requiring the manual configuration provided by the monitoring tools. With these customized pieces of code, the team

TABLE III
DETAILED FEEDBACK FROM OPERATIONS TO DEVELOPMENT (EXTRACT)

```

...
{"Report ID": "Flow_Meter_1_#000120",
 "Date": "7-11-19 8:38:16",
 "Error Indicator": 8, "Data Availability": "40%",
 "Request Indicator": 0, "Request Availability": "97%",
 "Threshold Indicator": 4, "Threshold Availability": "87%",
 "Total Availability Flow Meter 1": "34%",
 "Feedback": "\"Device connected\" \"Data errors detected (Critical)\" \"Data out of range (Critical)\" \"Device availability is growing\""}
...
{"Report ID": "Flow_Meter_1_#000130",
 "Date": "7-11-19 8:48:16",
 "Error Indicator": 1, "Data Availability": "41%",
 "Request Indicator": 9, "Request Availability": "90%",
 "Threshold Indicator": 1, "Threshold Availability": "87%",
 "Total Availability Flow Meter 1": "32%",
 "Feedback": "\"Device not connected\" \"Device manually Switched off\" \"Device is not available (Critical)\""}
...
    
```

can define their own metrics and indicators at runtime and receive detailed information about the availability of an IoT system in execution. Therefore, DevOps team members share the responsibility of releasing and deploying reliable software and have the necessary infrastructure and information to do it, which is key to enable a DevOps culture.

C. Implementation of the “F&CF Availability” Activity in Real Scenario

It enables the definition of specific availability tests adapted to the IoT system (defining specific indicators and alarm levels) and the running of these tests over the system in production minimizing the risks by detecting faults very quickly.

VII. CONCLUSION

This work is focused on the need of apply DevOps in the development of reliable and compelling IoT systems. The article presented and formalized through SPEM 2.0 a key activity named F&CF availability that supports feedback from operations to development in the process of maintenance and continuous improvement for IoT systems in operation. The formal specification provides reusability and flexibility by separating good practices (method content) from their use in different modalities of software development (process).

The F&CF availability activity has been designed to allow DevOps teams to receive detailed feedback from operations. This monitoring is based on the detection of anomalies or failures of availability from IoT hardware infrastructure, devices, IoT software infrastructure, and IoT applications. The feedback obtained from the detection of availability anomalies allows DevOps members in operations to make more informed decisions to fix problems or support IoT system evolution.

The monitoring system and the monitoring on demand components were developed using the techniques IaC and MaC, which enables two DevOps good practices: all the components of the monitoring infrastructure are versioned (GitLab CI) and repeatable as their deployment had been automated through virtualization and containerization technology.

The case study provides evidence of how this monitoring infrastructure for a potable water supply system enabled the detection of system availability status and helped to better anticipate anomalies that could produce unavailability in the midterm when a production deployment was performed.

The validity of the approach was performed through a case study. All case studies are qualitative in nature. That is, they are, in general terms, very difficult to judge objectively [39]. However, they are often used in software engineering due to the difficulty in producing multiple experiments or experiments with large populations. In fact, the major limitation in case study research concerns external validity, “the generality of the results with respect to a specific population” [40], in that only one case is studied. In our examination, though only one case was studied, in the humble opinion of the authors, it was sufficient to validate the contributions claimed. In fact, this case study allowed the research team to evaluate the designed activity and the process practices in a real setting. This is an important aspect in software engineering, in which a multitude of external factors may affect the validation results [41].

VIII. FUTURE WORK

We plan to address the formal specification (using SPEM 2.0) of the whole DevOps process for IoT. This process will define the phases and activities needed to represent the

three main DevOps practices [9]: 1) the fast flow of work from development into operation; 2) fast and continuous feedback from operations to development; and 3) continuous improvement.

We also plan to address new *trustworthiness* dimensions in IoT systems (integrity, reliability, etc.) [32], and we will pose the evolution from “monitoring” to “supervision” to improve the feedback and provide developers not only fine grain information but recommendations and procedures to solve system anomalies and failures. Along this line, we will extend the instantiation of the F&CF availability activity to *supervision as code*. The specification of the activities for these new characteristics as points of variability in the SPEM specification will also be studied.

REFERENCES

- [1] L. Atzori, A. Iera, and G. Morabito, “Understanding the Internet of Things: Definition, potentials, and societal role of a fast evolving paradigm,” *Ad Hoc Netw.*, vol. 56, pp. 122–140, Mar. 2017.
- [2] T. Kramp, R. Kranenburg, and S. Lange, Introduction to the Internet of Things,” in *Enabling Things to Talk: Designing IoT solutions With the IoT Architectural Reference Model*. Berlin, Germany: Springer, 2013, pp. 1–10, doi: [10.1007/978-3-642-40403-0](https://doi.org/10.1007/978-3-642-40403-0).
- [3] E. Lee, “Cyber physical systems: Design challenges,” in *Proc. 11th IEEE Int. Symp. Object Orient. Real Time Distrib. Comput.*, Orlando, FL, USA, 2008, pp. 1189–1194.
- [4] J. Reason and A. Hobbs, *Managing Maintenance Error: A Practical Guide*. Hoboken, NJ, USA: CRC Press, 2017.
- [5] G. Casale *et al.*, “Current and future challenges of software engineering for services and applications,” *Procedia Comput. Sci.*, vol. 97, pp. 34–42, Oct. 2016.
- [6] L. Lwakatare, P. Kuvaja, and M. Oivo, “Relationship of DevOps to agile, lean and continuous deployment,” in *Proc. 17th Int. Conf. PROFES*, 2016, pp. 399–415.
- [7] L. Lwakatare, P. Kuvaja, and M. Oivo, “An exploratory study of DevOps extending the dimensions of DevOps with practices,” in *Proc. 11th Int. Conf. Softw. Eng. Adv.*, 2016, pp. 91–99.
- [8] A. Dyck, R. Penners, and H. Lichter, “Towards definitions for release engineering and DevOps,” in *Proc. IEEE/ACM 3rd Int. Workshop Release Eng.*, 2015, p. 3.
- [9] G. Kim, J. Willis, P. Debois, and J. Humble, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Portland, OR, USA: IT Revolution Press, 2016.
- [10] L. Lwakatare *et al.*, “Towards DevOps in the embedded systems domain: Why is it so hard?” in *Proc. 49th Hawaii Int. Conf. Syst. Sci. (HICSS)*, 2016, pp. 5437–5446.
- [11] D. Mattos, J. Bosch, and H. Holmstrom, “Agile processes: Challenges and strategies for undertaking continuous experimentation to embedded systems: Industry and research perspectives,” in *Proc. XP*, 2018, pp. 277–292.
- [12] OMG. (2008). *About the Software Systems Process Engineering Metamodel Specification (SPEM) Version 2.0*. [Online]. Available: <http://www.omg.org/spec/SPEM/>
- [13] F. Erich, C. Amrit, and M. Daneva, “A qualitative study of DevOps usage in practice,” *J. Softw. Evol. Process.*, vol. 29, no. 6, pp. 205–240, Jan. 2017.
- [14] W. Van Gremberen and S. De Haes, “Introduction to the minitrack on IT governance and its mechanisms,” in *Proc. 51st Hawaii Int. Conf. Syst. Sci.*, 2018, pp. 1–2.
- [15] *Information Technology—Internet of Things—Definition and Vocabulary*, Int. Standard 20924, 2018.
- [16] *Internet of Things (IoT)—Reference Architecture*, Int. Standard 30141, 2018.
- [17] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller, “A survey of commercial frameworks for the Internet of Things,” in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Autom. (ETFA)*, 2015, pp. 1–8.

- [18] P. Ray, "A survey of IoT cloud platforms," *Future Comput. Informat. J.*, vol. 1, nos. 1–2, pp. 35–46, 2016.
- [19] *Industrial Internet Consortium (IIC)*. Accessed: Jun. 2020. [Online]. Available: <https://www.iiconsortium.org>
- [20] Y. C. Hu *et al.*, "Mobile edge computing—A key technology towards 5G," ETSI, Sophia Antipolis, France, White Paper, pp. 1–16, 2015.
- [21] P. Fremantle. (2015). *A Reference Architecture for the Internet of Things (White Paper)*. [Online]. Available: <https://wso2.com/whitepapers/a-reference-architecture-for-the-internet-of-things/>
- [22] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things characterization of fog computing," in *Proc. 1st MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–15.
- [23] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future Internet: The Internet of Things architecture, possible applications and key challenges," in *Proc. 10th Int. Conf. Front. Technol.*, 2012, pp. 257–260.
- [24] Y. Shanhe, H. Zijiang, Q. Zhengrui, and L. Qun, "Fog computing: Platform and applications," In *Proc. 3rd IEEE Workshop Hot Topics Web Syst. Technol. (HotWeb)*, 2015, pp. 73–78.
- [25] A. Munir, P. Kansakar, and S. U. Khan, "IFCIoT: Integrated fog cloud IoT: A novel architectural paradigm for the future Internet of Things," *IEEE Consum. Electron. Mag.*, vol. 6, no. 3, pp. 74–82, Jul. 2017.
- [26] S. Shekhar, A. D. Chhokra, A. Bhattacharjee, G. Aupy, and A. S. Gokhale, "INDICES: Exploiting edge resources for performance-aware cloud-hosted services," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput. (ICFEC)*, 2017, pp. 75–80.
- [27] P. Debois, "Agile infrastructure and operations: How infra-gile are you?" in *Proc. Agile Conf.*, 2008, pp. 202–207.
- [28] A. Wahaballa, O. Wahaballa, M. Abdellatif, H. Xiong, and Z. Qin, "Toward unified DevOps model," in *Proc. 6th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Beijing, China, 2015, pp. 211–214.
- [29] O. Krancher, P. Luther, and M. Jost, "Key affordances of platform-as-a-service: Self-organization and continuous feedback," *J. Manag. Inf. Syst.*, vol. 35, no. 3, pp. 776–812, 2018.
- [30] A. Balalaie, H. Abbas, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, Mar. 2016.
- [31] J. Díaz, J. Pérez, M. A. López-Peña, G. Mena, and A. Yagüe, "Self-service cybersecurity monitoring as enabler for DevSecOps," *IEEE Access*, vol. 7, pp. 100283–100295, 2019.
- [32] N. Ferry *et al.*, "ENACT: Development, operation, and quality assurance of trustworthy smart IoT systems," in *Proc. DEVOPS*, 2018, pp. 112–127.
- [33] A. Karapantelakis *et al.*, "DevOps for IoT applications using cellular networks and cloud," in *Proc. IEEE 4th Int. Conf. Future Internet Things Cloud (FiCloud)*, 2016, pp. 340–347.
- [34] S. Kim and J. Kim, "Enabling operation data visibility for SmartX-MINI IoT-cloud playground," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, 2016, pp. 428–430.
- [35] J. Bae, C. Kim, and J. Kim, "Automated deployment of SmartX IoT-cloud services based on continuous integration," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, 2016, pp. 684–700.
- [36] M. Syed and E. Fernandez, "Cloud ecosystems support for Internet of Things and DevOps using patterns," in *Proc. IEEE 1st Int. Conf. Internet Things Design Implement. (IoTDI)*, 2016, pp. 301–304.
- [37] M. A. Lopez-Peña and I. M. Fernández, *SAT-IoT: An Architectural Model for a High Performance Fog/Edge/Cloud IoT Platform*, Limericks County Library, Limerick, Ireland, 2019.
- [38] I. Silva, L. A. Guedes, P. Portugal, and F. Vasques, "Reliability and availability evaluation of wireless sensor networks for industrial applications," *Sensors*, vol. 12, no. 1, pp. 806–838, 2012.
- [39] R. K. Yin, *Case Study Research: Design and Methods*, 4th ed. London, U.K.: SAGE, 2009.
- [40] U. Van Heesch, P. Avgeriou, and R. Hilliard, "A documentation framework for architecture decisions," *J. Syst. Softw.*, vol. 85, no. 4, pp. 795–820, 2012.
- [41] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 557–572, Jul./Aug. 1999.



Miguel A. López-Peña received the B.S. degree in computer science from the Universidad Carlos III de Madrid, Getafe, Spain, and the master's degree from the Spanish Ministry of Education (EQF level 7), have postgraduate studies from the Universidad Complutense de Madrid (Physics Sciences Faculty Doctorate Courses), Madrid, Spain, Rey Juan Carlos de Madrid (ESCET Faculty Master's Programs), Madrid, and IESE Business School, Barcelona, Spain. He is currently pursuing the Ph.D. degree in "Science and Computer Technologies for Smart Cities" Program with the Universidad Politécnica de Madrid, Madrid.

Since 2005, he has been an Innovation and Development Manager with the Sistemas Avanzados de Tecnología, S.A., Madrid, and an ICT private multinational company with which he has participating in multitude of European and national Research and Development projects.



Jessica Díaz received the Ph.D. degree in computer science from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 2012 (best thesis Award).

She is currently an Associate Professor with the E.T.S. Ingeniería de Sistemas Informáticos, UPM, where she has been a Researcher with the System and Software Technology Research Group since April 2003 and is participating in several European and National projects related to software engineering on Internet of Things and smart systems. Her research interests include DevOps and rapid application development, cloud computing, software architectures, software product lines, and model-driven development.



Jorge E. Pérez received the B.S. degree in computer science from the Universidad Carlos III de Madrid, Madrid, Spain, in 1999, and the Ph.D. degree from the Universidad Politécnica de Madrid (UPM), Madrid, in 2004.

From 1993 to 1997, he was a Secretary of the Faculty of Computer Sciences, UPM. From 2007 to 2014, he was the Headmaster of the Department of Applied Computer Science, UPM, where he has been an Associate Professor since 1985. He is currently the Coordinator of the DMAE-DIA Educational Innovation Group and the Competencies and Active Learning in Engineering Education Research Group. He has taken part in the creation of several syllabuses for both grades and Postgraduates and has supervised many research projects. His research interests include software architectures and modeling.

Dr. Pérez is a recipient of several awards from the Rector of the UPM for educational innovation at university.



Héctor Humanes received the B.S. degree in software engineering and the master's degree in embedded and distributed system's software from the Universidad Politécnica de Madrid, Madrid, Spain, in 2017 and 2018, respectively.

Since 2018, he has been the Technical Leader of the Innovation Department, Sistemas Avanzados de Tecnología, S.A., Madrid. He has worked for the System and Software Technology Group and the Research Group, Universidad Politécnica of Madrid, Madrid, Spain, where he grew his interests in software architecture and IoT systems.