Continuous Delivery of Customized SaaS Edge Applications in Highly Distributed IoT Systems

Ramón López-Viana, Jessica Díaz, *Member, IEEE*, Vicente Hernández Díaz, and José-Fernán Martínez

***Abstract*—Edge computing is a reality for the current IoT systems that need fast processing and quick response time to make real-time decisions and IoT systems without permanent connectivity to the cloud (e.g., car manufacturing, precision agriculture, or cattle raising). Additionally, these industries are facing the need for rapid and continuous innovation by accelerating the delivery of over-the-air (OTA) software updates in edge devices. DevOps promotes collaboration between development and operation teams and automation at all steps of software construction to achieve continuous delivery (CD) of business value. Although DevOps has demonstrated numerous successful cases in the Web domain, in the IoT domain and, more specifically, at the edge, there are few reported cases. This work presents a success case of CD of customized software as a service software as a service (SaaS) updates at the IoT Edge. This may enable new business models at the IoT Edge. This article presents an architectural model of a highly distributed (cloud and edge) IoT system and a CD process flow for customized SaaS applications in edge nodes. Both the architectural model and the CD process flow are instantiated in a case study for precision agriculture.**

***Index Terms*—Continuous delivery (CD), DevOps, edge computing, IoT systems, software as a service (SaaS).**

I. Introduction

Internet of Things is revolutionizing traditional businesses, such as precision agriculture, cattle raising, car manufacturing, etc. IoT systems that support such revolution are highly distributed, which often integrate *cloud*, *edge*, and *fog* computing approaches depending on where intelligence and processing capabilities are allocated. Initially, cloud computing was the natural candidate to support the exponential growth of data generated by thousands or even millions of "things" [1], [2]. However, in some domains, these systems also require fast processing and quick response time to make real-time decisions [3], which is difficult due to cloud latency or is not possible due to connectivity problems. To deal with these challenges, edge computing brings the services and utilities of cloud computing closer to the devices and, thus, to end users [4] (*edge intelligence*). Edge computing provides fast processing, quick application response time, and low latency to delay-sensitive applications [4]. Furthermore, it does not require permanent connectivity to the cloud [5].

Unlike the cloud paradigm in which only a few players gained the market share, edge computing is generating opportunities for a wide variety of industries and is having an important economic and societal impact [3]. In the same way, as software as a service (SaaS) is the cloud business and the operating,[1] edge computing may adopt this successful model to deliver managed applications by third-party vendors to their users. In this sense, edge computing can be considered a way to extend cloud as hybrid clouds in which public clouds and on-premise infrastructure work together.

One of the keys to the success of the SaaS model is the ability to frequently deliver the value, much reliably and quickly, and DevOps promotes this. DevOps can be defined as an organizational and cultural approach to improve and accelerate the delivery of the business value by making dev and ops teams' collaboration effective and automating all steps of software construction [6]. DevOps is usually adopted by Web companies [7]; however, for IoT and embedded systems, the domain is harder [8]. The IoT industry is facing this challenge, specifically the need to deliver innovation fast, which means decrease time to market and deliver software updates to edge devices continuously [9]. A lot of IoT companies—e.g., Bosch Software Innovations and the Toyota Research Institute, among others—are approaching how to deliver over-the-air (OTA) software updates in edge devices, which is a starting point for an automated and continuous delivery (CD), and thus, *continuous experimentation*. To deal with these challenges, leading software cloud providers, such as Microsoft Azure and Amazon Web Services, are facing the adoption of these DevOps practices and principles at the edge. However, as this industry grows, more and more complex issues emerge, such as how to deliver large-scale software updates to these edge devices—e.g., the case of car manufacturing and the autonomous vehicle [9]—and how to customize SaaS for different users of these edge devices (e.g., based on different levels of subscription or preconfigured user preferences).

This work faces the challenge of applying CD of customized SaaS updates at the IoT Edge. In this article, CD at the IoT edge means that, on top of having automated building and testing of software modules for edge devices (versus field

Manuscript received October 14, 2019; revised February 13, 2020 and June 18, 2020; accepted July 13, 2020. Date of publication July 16, 2020; date of current version October 9, 2020. This work was supported in part by the AFARCLOUD European Project (Aggregate Farming in the Cloud), the latter supported in part by the ECSEL JU under Grant 783221-AFarCloud-H2020-ECSEL-2017-2, and in part by the Spanish Ministry of Science, Innovation and Universities under Grant PCI2018-092965; and in part by the crowdsaving under Grant TIN2016-79726-C2-1-R. *(Corresponding author: Jessica Díaz.)*

The authors are with the Universidad Politécnica de Madrid, 28031 Madrid, Spain (e-mail: ramon.lopez.viana@alumnos.upm.es; yesica.diaz@upm.es; vicente.hernandez@upm.es; jf.martinez@upm.es).

Digital Object Identifier 10.1109/JIOT.2020.3009633

[1]Source: Gartner (April 2019) https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g.

devices such as sensors), we also have automated the release process and can deploy these modules at any point of time by clicking on a button. To that end, this article presents an architectural model of a highly distributed (cloud and edge) IoT system and a CD process flow for customized SaaS applications in edge nodes. In this way, software vendors can offer their own software services on demand (SaaS) through cloud providers, and both installations and configurations of software at the edge are managed through a set of pipelines we have defined.

Both the architectural model and the CD flow are instantiated in a case study for precision agriculture in which various technologies are integrated: low-power devices, LoRaWAN platform, ARM-based edge devices, AMD-based edge devices, containers, Node-RED, and Azure cloud services, such as Azure DevOps, Azure IoT, Azure IoT Edge, device provisioning, functions, queues, storage, and Power BI.

This article is structured as follows. Section II introduces the concepts of edge computing and DevOps. Section III describes related work. Section IV presents an architecture model for highly distributed IoT systems and a CD process flow for customized SaaS applications in edge nodes. Its instantiation is described through a case study in Section V. Finally, conclusions are described in Section VI.

II. Edge Computing and DevOps

Under the umbrella of edge computing, there is a plethora of paradigms related to the processing of data out of the cloud, such as fog computing and mobile-edge computing, that brings some of the processing capabilities previously provided by the cloud closer to the data sources [10], [11]. This new paradigm is the response to the emerging problems related to the adoption of IoT. Some of these problems are the increasing number of IoT devices, the bandwidth required to transmit the generated data, the network latency, the network availability, and the security threats associated with the data travelling from the IoT devices to the cloud. There are many platforms available for building edge solutions [12], both commercial (AWS Greengrass, Microsoft Azure IoT Edge, etc.) and opensource (Fiware Fogflow, Eclipse Kura, etc.). The main contenders are AWS Greengrass and Microsoft Azure IoT as stated in the 2018 Gartner Magic Quadrant for IaaS.[2]

As the IoT edge industry grows, it demands fast innovation and requires the adoption of new organizational capabilities to develop, release, and learn from software in rapid cycles (rapid software development [13]). The European cluster SE4SA emphasizes that "managing the development complexity and risks in both design and runtime phases is considered crucial, in order to increase QoS, reduce the time needed to move new releases in the operation environment and enable the constitution of new and more effective cooperation processes between the development and the operation team" [14]. This assertion was made in 2016 in a context defined to identify new challenges of software engineering for smart systems and applications, specifically in domains such as IoT, cyber–physical systems (CPSs), cloud, and big data.

[2]Magic Quadrant for Cloud Infrastructure as a Service, Worldwide.

Silos between development and IT operation, which currently exist in most technology companies, make early and frequent releases in production more complex, which means less business innovation and lower capability to compete in the market [14]. DevOps is an emerging approach that promotes collaboration to break silos and automation for fast speed in releases and quick response time to customer demands [6]. The automation of the building, testing, and deploying processes, enables good practices such as continuous integration (CI) and CD, and thus, speeds release up. The CD of a SaaS model based on preconfigured customer preferences may enable new business models at the IoT edge. However, the adoption of such a model at the edge requires to deal with some challenges [8], [9].

1) *Customization:* More and more customers require to adopt IoT solutions to their needs, being necessary the ability to customize and release software updates at any time by automating software integration and delivery processes.
2) *Connectivity:* The Internet connectivity can vary depending on the environment where the solution is deployed. Internet connectivity is not the same in all countries or in all areas in the same country. Even when the available bandwidth or the network quality is poor, the edge devices must be able to receive updates and provide most of the expected services.
3) *Resilience:* The downtime due to updates and after errors should be as short as possible, by implementing the required procedures to recover the status of the system.

III. Related Work

This section reports the related work about CD and other (DevOps) practices concerning the automation of software delivery in the IoT. As the application of these practices to IoT is relatively new and recently supported by IT companies, few related works were found. Specifically, few implementations or successful cases and experiences have been reported. Some researches try to explain this fact. Hence, Lwakatare *et al.* [8] and Mattos *et al.* [9] 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.

The few IoT use cases implementing DevOps that were found [15]–[20] were analyzed with the aim of identifying if CD was applied for edge devices and if the above-mentioned challenges were addressed. Karapantelakis *et al.* [15] 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, i.e., the system automates allocation and deallocation of network and cloud resources based on the information provided by a monitoring infrastructure—network, CPU, and memory status. Bae *et al.* [16] described the automation of the CI and the deployment of IoT cloud services using containers. Syed and Fernandez [17] described a cloud ecosystem to support both IoT—fog computing—and

DevOps—automated management of infrastructure—but not explicitly together. All those works aim to automate the delivery of IoT applications, but none of them focuses on edge devices.

Moore *et al.* [18] presented a system to collect and examine the use of stored solar energy within a home and an electric vehicle. This work focuses on applying containerization technologies to build systems that scale from one user to many, i.e., deploying software in a scalable way, ostensibly, on devices. However, as this is a short article, neither CI nor delivery workflows are described.

Bae *et al.* [16] described the CI and automated deployment of a microservice-based IoT cloud solution for a smart lab energy management. Recently, Banijamali *et al.* [19] described a specific solution for OTA CD of software updates in the automotive domain. Bae *et al.* and Banijamali *et al.* show the growing concern in the IoT community about CI/CD and both are similar to the contribution presented here, i.e., automated delivery of software. However, these papers do not describe an architectural model and a CD process flow for a generic IoT highly distributed solution (cloud + edge) as we do. Banijamali *et al.* [19] described the CD of software updates in an automotive cloud platform but it does not describe the automated deployment in the in-vehicle platform. We explicitly describe in detail the services and the process flow for deploying new software in edge devices. Bae *et al.* [16] described the automated deployment of containers on Raspberry Pi devices. However, none of them address the above-mentioned challenges that edge faces as this article does: 1) the customization of SaaS at the edge for readily deploying new building blocks for providing new capabilities on farmer's demand, which may enable new business models at the IoT Edge; 2) connectivity restrictions typical of the precision agriculture domain; and 3) resilience in terms of low downtime and status recovery (prevent message loss).

IV. Architectural Model and CD Flow for Customized SaaS Applications in Edge Devices

The architectural model and CD flow for highly distributed IoT systems we present aims: 1) to deliver customizable IoT solutions to user requirements based on the concept of SaaS; 2) to provide CD of business value through the automated deployment of software updates in edge devices based on preconfigured delivery policies; 3) to implement most of the intelligence required by the system without requiring a constant connection to the Internet (edge computing); and 4) to provide the required execution environment in edge devices to deliver and execute resilient services.

Fig. 1 shows a three-tier architecture model for highly distributed systems: 1) *the cloud layer* is composed by IoT-related cloud services, such as ingestion, hot and batch processing, storing, visualization, etc., as well as DevOps-related services and tools for automating the building and deployment of software updates both in the cloud and the edge; 2) *the edge layer* is composed of edge devices that provide the capability of processing data, managing IoT devices, and communicating devices with the cloud layer through a *runtime*; and 3) *the sensor network layer* is composed of IoT

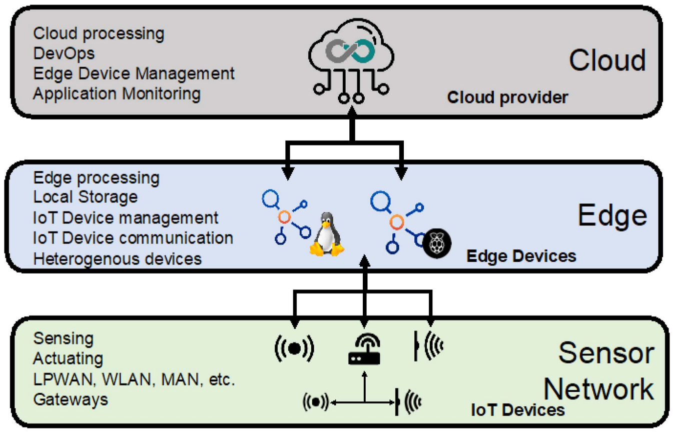

Fig. 1. Three-tier architectural model for highly distributed IoT systems.

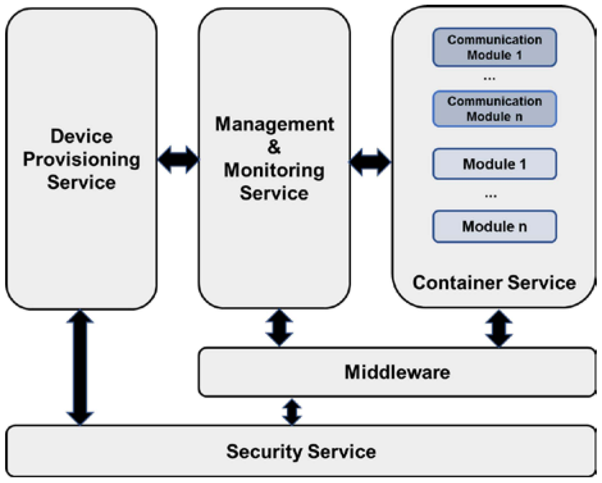

Fig. 2. Edge device runtime.

devices, which are responsible for sensing and/or actuating the surrounding environment. In most cases, these elements do not have processing capabilities and often may require a gateway to translate the communication protocol from IoT devices to edge devices. Especially, solutions that require energy efficiency also require low power or ultralow-power wireless sensor networks (WSNs [19]) for wide areas through the use of networking technologies, such as Sigfox, LoRa, and NB-IoT. This article focuses on the description of the first two top layers.

A. Edge Layer

Edge devices require a collection of services to run code at the edge (also known as modules), receive modules to run at the edge, manage and monitor the health of the device, and enable security and communication. This collection of services and programs that turns a device into an IoT edge device is commonly named an *IoT edge runtime*. It is especially important for this article that the runtime supports software updates to leverage a SaaS model using a DevOps lifecycle. Fig. 2 shows the services that an edge device runtime must provide the following.

1) *The management and monitoring services* monitor the status of the devices, provide metrics about devices'

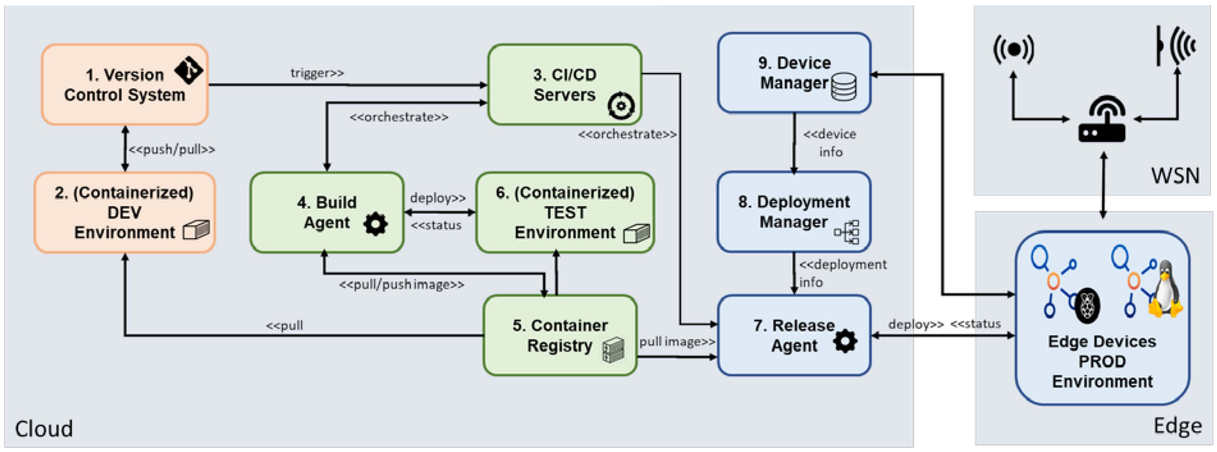

Fig. 3. CD process flow: cloud components and tools.

behavior and performance, and apply configuration changes to the devices.

2) *The security service* guarantees a trust relationship between the edge devices and the cloud platform by providing the required components for ensuring the security of the communications, the devices' identity, and the provenance of the modules to be deployed on the edge devices. This may require using the hardware security model (HSM), signature access service (SAS), or digital certificates and the required protocols.

3) *The device provisioning service* automatically enrolls edge devices without requiring user intervention, which is crucial due to the number and heterogeneity of the devices involved in this type of systems. So, the device must be preconfigured with the required information.

4) *The container service* enables the deployment of the modules that implement the business logic to be executed at the edge. Containerization provides a way to package and isolate the modules that implement the logic in the edge with all their dependencies. Each module has a defined responsibility that can be stopped, started, or modified as required independently of other modules.

5) *The middleware* enables the communication between modules, a device to cloud, and cloud to device. Edge devices should be able to work disconnected from the cloud (e.g., in areas with connectivity restrictions). Thus, it is necessary to consider protocols, like MQTT, that support devices can be disconnected most of the time, event-driven behavior, and the capability of storing events until these are processed.

B. Cloud Layer

The cloud layer provides, additional to ingestion, processing, and storing services, the required infrastructure to implement IoT DevOps lifecycles. Thus, it supports CI and deployment, with special emphasis devices' scalability, connectivity, and platform heterogeneity, typical of highly distributed IoT systems. Fig. 3 shows the components and tools that support the DevOps lifecycle and their relation.

1) *Version control systems* manage code changes and configuration management practices.

2) *Development environments* ensure that the code (packaged as modules) works in production-like environments.

3) *CI/CD servers* mainly implement build, test, and release pipelines for committed code changes and report the results.

4) *Build agents* are used during build pipelines to compile, test, and create container images of modules.

5) *Container registries* store modules images that are the result of build/test pipelines and provide a way to access the modules images as part of a distribution/release.

6) *Testing environments* ensure quality, performance, availability, security, etc., of code through acceptance tests.

7) *Release agents* are used during release pipelines to deploy container images into edge devices. The cloud platform must support build/release agents for different devices' platforms that may make up an IoT system. If the cloud is not able to host the agent platform of a target device, then the cloud must provide a mechanism to create and connect *external agents* in the pipeline. For example, the cloud provider may only host agents for AMD (Ubuntu or Windows), whereas the target device platform is ARM. In this case, the cloud provider has to offer a mechanism to connect with a remote agent for ARM to be executed during the build/release pipeline.

8) *Deployment managers* apply delivery policies for deploying new modules into edge devices. These policies are described in a *deployment descriptor* and allow release managers to define the policies to customize the services to be deployed in a device. These policies include: a) the conditions that an edge device must meet to be a target of a new deployment; b) the device target properties to check the deployment status; c) the modules (version, configuration, etc.) to be deployed on the edge devices; d) the communication definition for both module-to-module and module-to-cloud and the retention time applied to the messages processed in the device that enables to recover the state of the service after a restart or when the device recovers the online condition after a disconnection from the cloud; and e) the metrics to monitor the edge device.

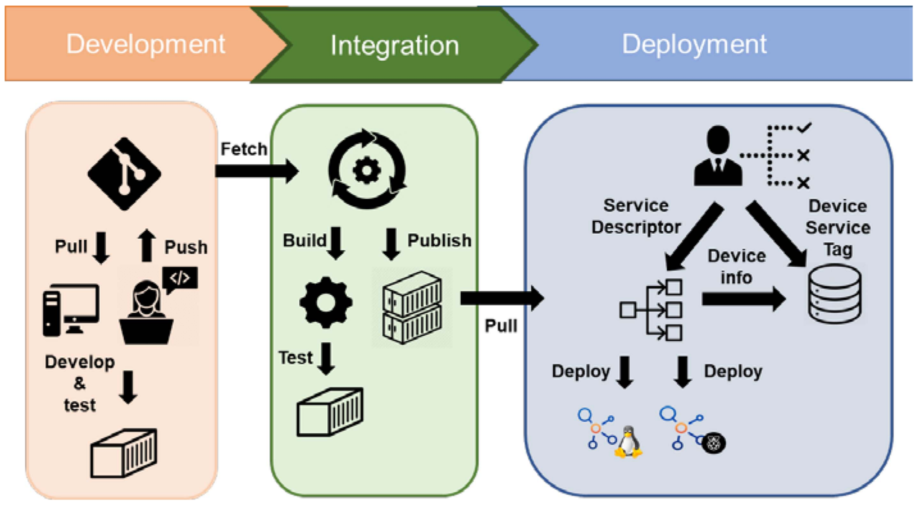

Fig. 4. CD process flow for customized SaaS applications at the edge.

9) *Device managers* supervise the information and status of edge devices. A *device registry* enables automatic device provisioning (i.e., negotiation of device enrollment and identification) and provides a way to identify devices as targets of deployments through tags.
10) *Monitoring managers* monitor edge devices as well as the deployment process to guarantee the expected result based on preconfigured metrics.

C. CD Flow for Customized SaaS in Edge Devices

This section defines a CD process flow that implements the integration and delivery of customized SaaS applications at the edge. Fig. 4 shows this flow. Developers build and test the modules' code locally and manage code changes into a version control system (see the column named *development* in Fig. 4). Then, code changes are integrated with the existing modules in a CI server. The CI server can be configured to pull, build, and test the code every time a change is committed (see the column named *integration* in Fig. 4). This building process is performed by the build agents, which are based on the characteristics of the target edge devices. If the result is successful, the generated artifacts are published in a container registry. Finally, the build pipeline reports the result to the interested parties (e.g., release managers).

Once the build pipeline ends successfully, the release agent checks if the delivery of a release is required, e.g., if it is required to deploy modules updates into production or production-like edge devices based on the content of a deployment descriptor (see the column named *deployment* in Fig. 4). The release pipeline can be customized based on the target device platform, the preconfigured preferences in the deployment descriptor, and the status of the edge devices (e.g., a device temporarily unavailable). The agent that executes the release pipeline requests the edge devices to apply the changes when these are available. Finally, the edge devices can be monitored for obtaining information about their status and the status of the modules deployed into them to apply corrective actions if errors are detected. On the contrary, if a deployment is successful, the information regarding the new status is stored in the device registry for future processing.

This CD process flow supports the customization of SaaS applications in the edge through an additional mechanism that allows release managers to tag the edge devices. Then, the release manager can configure both the release pipeline and the deployment descriptor to specify which modules have to be deployed in which edge devices.

V. Case Study

The case study in which the architectural model and the CD process flow described in the previous section is instantiated is a prototype for a precision agriculture. Precision agriculture is a challenging domain due to the lack of a public communications network in some areas in the world, low-energy consumption constraints for deployed devices, and the heterogeneity of environmental conditions that impact crop raising. The case study focuses on the monitoring and management of crop fields in isolated areas and the main challenge to be addressed is to provide a customized solution according to farmers' requirements following a SaaS model. This case study allows us to validate if the architectural model and the CD flow satisfy the requirements to support the customization of SaaS applications for precision agriculture in isolated areas.

A. Technical Requirements

The solution for this prototype should be flexible enough to adapt to farmers' requirements' changes in the shortest possible time. Whenever a farmer switches the crop type, new software may be deployed in the edge devices (if the necessary sensors and actuators exist). Next, the technical requirements that have been used for validating the prototype are described as follows.

1) To provide a SaaS model in which new software modules can be added, increasing the catalog of services available for their deployment, and thus, increasing the value of the solution.

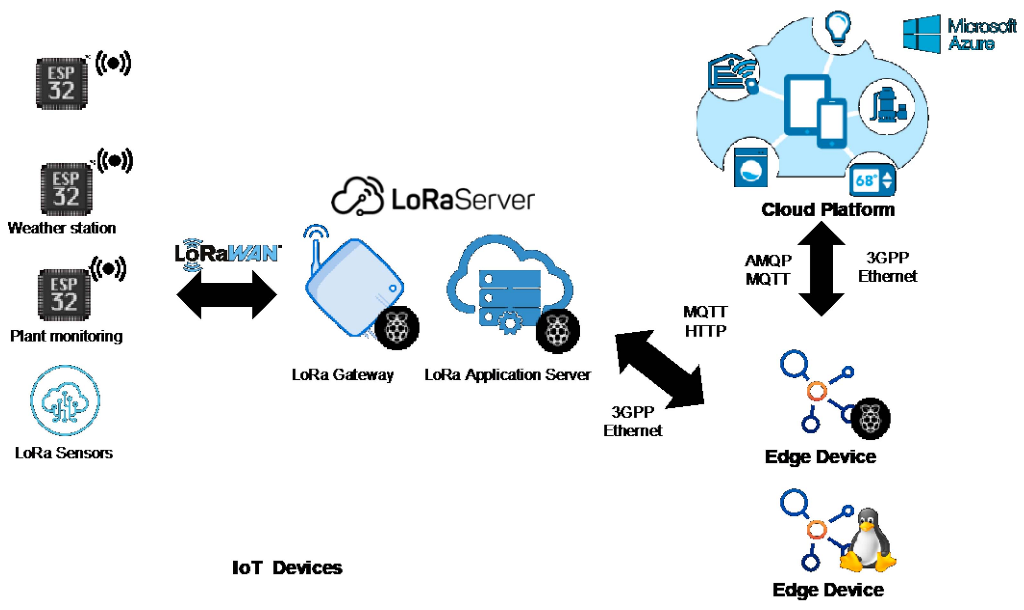

Fig. 5. Architecture deployment for a precision agriculture solution.

2) To update modules and deploy new ones automatically, without a disruptive effect on the operation (zero downtime).
3) To minimize the bandwidth required for a service update or upgrade.
4) To implement most of the intelligence required by the system without requiring a permanent connection to the cloud, thus optimizing the bandwidth consumption.
5) To supply advanced processing capabilities using self-managed cloud services (ingestion, batch processing, and visualization).

The research question that this case study is meant to answer is: do the architectural model and the CD process flow we defined, meet the requirements here described? To that end, the solution needs to consider at least the following elements: 1) the IoT devices to be deployed on the crop field; 2) the WSN to support the communication of the IoT devices; 3) the edge devices and the modules for processing the IoT device data; 4) the cloud services to manage and monitor the solution; and 5) the cloud services required for provisioning the SaaS model and the CD process flow.

The description of the case study focuses on the edge devices and the services that are needed to support the automatic deployment of customized SaaS applications at the edge, although a summary of other required elements is described too.

B. Overall Architecture of the Precision Agriculture Prototype

Fig. 5 shows the architecture of the solution. The *IoT devices* are deployed in the farmer's facilities or crop fields for sensing and actuating on the surrounding environment. Two prototypes of IoT devices have been built using a cheap device such as the ESP32 with a LoRa communication module and a set of sensors (for a detailed description, see http://github.com/rlopezv/aaas/aaas-device-esp32). These

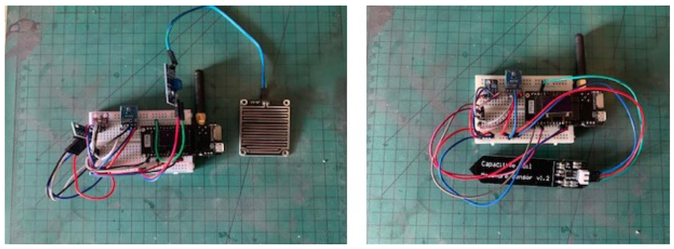

Fig. 6. IoT device prototypes.

prototypes are depicted in Fig. 6. The one on the left works as a weather station and the other on the right works as a plant monitoring station.

These devices have been programmed using the Arduino framework and the bidirectional communication has been achieved by piggybacking the LoRa ACK messages sent by the gateway. The LoRa server is a specialized IoT device that supports the use of LoRa communications. It is composed of a LoRa gateway and a LoRa app server, which are responsible for the communication between the IoT devices and the edge devices. Among the communication class types available for LoRa communications (*A*, *B*, and *C*—see section *LoRaWAN MAC layer* in [21]), the selected one has been *A*, which supports bidirectional communication between a device and a gateway with the lowest power requirements, very important in a context like the one selected. Nonetheless, the drawback of communication class type *A* is that the downlink communication from the gateway can only be done shortly after the end device has sent an uplink transmission. For enrolling the IoT devices, the selected mechanism has been OTA activation (OTAA), instead of activation by personalization (ABAP), which provides a more secure way when establishing the connection between the gateway and the IoT device since it requires to perform a join-procedure with the gateway, during which a dynamic DevAddr is assigned and security keys are negotiated with the device. Both OTAA

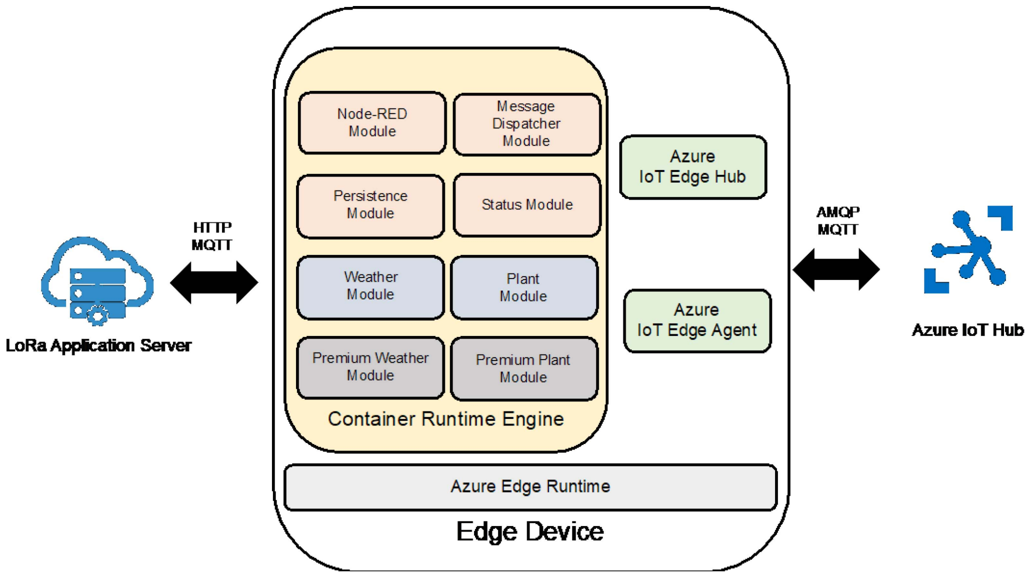

Fig. 7. Architecture design of edge devices.

and ABAP are technological solutions that provide means for deploying and undeploying software modules in wirelessly connected devices. The LoRa server is composed of the RAK831 Lora radio front-end module from RAK Wireless [22] running on a Raspberry Pi 3B with the open-source loraserver.io [23] platform installed that provides all the required services in a LoRaWAN network.

The *edge devices* (see Fig. 5) are deployed in the farmer's facilities and host the runtime described in Fig. 2 (see Section IV). In this prototype, we selected the *Azure Edge runtime environment* [24], which contains the software services that turn a computing device into an edge device using Microsoft Azure platform. The edge devices execute the modules to provide the services selected by the farmers on this runtime. The runtime also enables IoT edge devices to receive modules to be deployed or updated and executed at the edge as well as to communicate the results. The edge devices can be any kind of device able to run the Azure Edge runtime environment. For this prototype, we used a Raspberry Pi 3B with Raspbian Stretch and a virtualized AMD64.

Finally, the *cloud platform* (see Fig. 5) is mainly responsible for the management of the infrastructure required for the proposed solution that supports the required functionalities that guarantee the correct application of the CI/CD practices in order to provide a service based on the SaaS model.

The following sections describe in detail the architecture of the edge devices, the cloud services, and the CD flow.

C. Architecture Design of Edge Devices

Fig. 7 shows the most relevant components deployed and installed in the edge devices of the prototype. We selected the Azure SDK for Node.js, which fits well with a constrained device such as the Raspberry Pi and developed several modules for processing the messages from both the devices and the cloud. Each device offers different services based on the modules that are selected during the deployment. Since the same SDK has been used for developing the modules, the images to be deployed on the devices share some resources. This reduces the image's size because the container engine can detect the shared resources and reuse them without downloading them again. These modules are as follows.

1) *Message Dispatcher Module:* This module processes the messages received from the LoRa gateway by the integration module. It stores and routes the received message to a queue for further processing in the configured modules.
2) *Plant Module:* This is a basic module that reports to the cloud the last available measure from a plant for a configured period, by default every 15 min.
3) *Plant Premium Module:* This module is an alternative to the previous one. It continuously notifies the measures and applies a simple algorithm to notify the irrigation requirements depending on the defined properties.
4) *Weather Module:* This is a basic module that reports to the cloud the last measure available from a weather station for a configured period, by default every 15 min.
5) *Weather Premium Module:* This module is an alternative to the previous one. It continuously notifies weather-related measurements and applies a simple algorithm to foresee weather changes.
6) *Status Module:* This module processes the status messages received from the IoT edge devices concerning the sensors' status. In addition to these modules, two modules have been included without using the Azure SDK.
7) *Node-RED Module:* This module creates a Node-RED [25] engine that implements some customized flows to communicate with the LoRa gateway and generates some dashboards based on the data received from the cloud and the IoT devices (see https://github.com/rlopezv/aaas/tree/master/aaas-edge-nodered-module). Although the LoRa gateway supports several protocols, such as MQTT, HTTP,

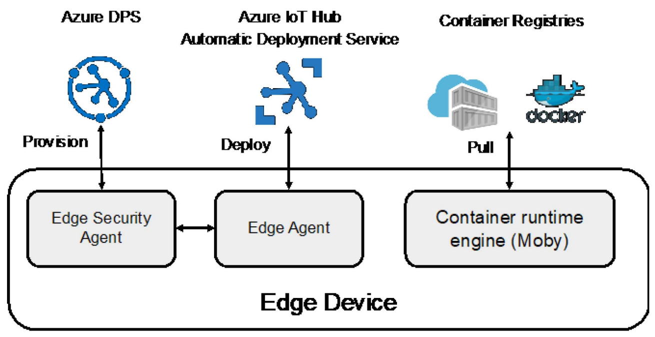

Fig. 8. Communication between the cloud layer and the edge layer to enable the automatic deployment of modules into edge devices.

and gRPC, the selected one for this case study has been MQTT, which allows defining the QoS to ensure the delivery of the messages in a simple way and is supported by both the edge device and the Node-RED framework.

8) *Persistence Module:* This module is based on the image provided by PostgreSQL that is used for storing the data received from the IoT devices and the cloud for further processing, such as the dashboard described previously. Since one of the objectives was to provide a SaaS service based on the farmer's selection, two service models were defined: a) a premium model, which includes the common modules and those named premium (plant and weather premium modules) and b) a basic model, which includes the common modules and the standard modules (weather and plant modules).

Finally, the services for provisioning, management, and communication described in Section IV are provided by the *edge agent* and the *edge runtime* which are part of the Azure IoT Edge runtime.

D. Cloud Services

The release of the modules to be deployed on the edge devices is automatically managed by the *Azure IoT hub* (see Fig. 8). The Azure IoT hub is the main service for connecting edge devices and the cloud. The main service it provides for this case study is the *automatic deployment service* (see Fig. 8) that automates deployments in a secure way and allows release engineers to define a set of conditions that devices must meet to receive the modules. In this case, the configuration of the deployments is based on a tag defined by the edge devices that identifies the service level, premium, or basic. Both conditions and modules to be deployed are specified in the deployment descriptor. This way, including a new module or changing a deployed one only requires modifying the deployment descriptor to configure the release. Once the device is enrolled, the Azure IoT hub deploys the modules in the container engine installed on the device.

Additionally, the *device provisioning service* (see Fig. 8) has been used since it supports zero-touch, just-in-time device provisioning to the right Azure IoT hub without requiring human intervention, enabling customers to provision devices in a secure and scalable way. Although this can be done using both X.509 and TPM-based identities, we used symmetric keys for prototyping purposes. It also allows release engineers to tag the devices for management and provides a way to monitor and configure the devices individually, which is used when selecting the modules to be deployed into the device.

The modules to be deployed in the edge devices are defined and configured in the deployment descriptor for the defined SaaS models (see Section IV). This file contains the following.

1) *The Modules to be Deployed Into the Device:* It includes the container image hosted in a container registry, such as Docker Hub or Azure Container Registry, along with the required credentials and the instructions for creating and managing every module. Some of the properties are related to the edge runtime environment and others are related to the customization of the docker image that contains the module.
2) *The Routes for Communicating the Modules Among Them or With the Cloud:* Their behavior is similar to a queue, allowing to define multiple routes for the same deployment. The retention policy applied for these messages can also be defined so that the messages cannot be lost whenever a module restarts or reconnects to the cloud platform.
3) *The Desired Properties for Each Module Deployed in the IoT Edge Device:* These properties are reported in the module twin and can be used for configuring and monitoring the module in the device.

Code 1 shows an excerpt of the deployment descriptor used for the premium service (in bold the most relevant elements).

E. Azure DevOps

The CD process flow described in Section IV-C is instantiated using the Azure DevOps platform. It requires connecting to any source control, such as GitHub, and can release changes continuously to edge devices. The created pipelines are based on *Azure IoT Edge for Azure Pipelines extension* that allows to setup CI/CD pipelines for edge devices on Azure DevOps. It provides "build" and "push" tasks for CI and a "deploy to IoT edge devices" task for CD. The pipelines extract the source code from a GitHub repository (http.//github.com/rlopezv/aaas-edge-modules), build the module images, and publish them to a container registry that hosts the images to be deployed in the edge devices (see Code 2). Finally, the deployment in the edge devices has been automatized with the IoT Edge automatic deployment service described previously.

Since one of the target platforms is ARM and Azure DevOps only provides hosted agents for amd64 and windows-amd64, a remote agent for building the modules hosted in a Raspberry Pi was created and configured (see https://github.com/rlopezv/aaas/aaa-azure). The agent executes the tasks in the hosted remote ARM agent, which has the characteristics described in Code 3.

Finally, a release pipeline for automatically deploying the modules in edge devices was defined (see Code 4). This pipeline mimics the functionality of the IoT hub deployment service looking for target devices and deploying the modules

Code 1 Deployment Descriptor in JSON

```
  "id": "aaas_premium_service,"
  "schemaVersion": null,
  "labels": {
    "service": "premium"
  },
  "content": {
    "modulesContent": {
[...]
            "postgres": {
              "settings": {
                "image": "postgres:9.6,"
                "createOptions": "{ HIDDEN }"
              },
            [...]
          "messagedispatchermodule": {
            "settings": {
              "image": "containerregistry.azurecr.io/ messagedispatcher
                       module:latest,"
              "createOptions": "{ HIDDEN }"
          },
            "type": "docker,"
            "status": "running,"
            "restartPolicy": "always,"
            "version": "1.0"
          },
          [...]
          "premiumweathermodule ": {
            "settings": {
              "image": "containerregistry.azurecr.io/ premiumweather
                       module:latest,"
              "createOptions": "{ HIDDEN }"
            },
            "type": "docker,"
            "status": "running,"
            "restartPolicy": "always,"
            "version": "1.0"
          },
          [...]
    "$edgeHub": {
      "properties.desired": {
        "routes": {
          "mesageDispatcherModuleToPremiumWeatherModule": "FROM
/messages/modules/messagedispatchermodule/outputs/weather       INTO
BrokeredEndpoint(\"/modules/premiumweathermodule/inputs/data\"),"
[...]
  "targetCondition": "service ='premium',"
[...]
```

Code 2 CI Pipeline Description in YAML

```
pool:
  name: ARM
steps:
- task: AzureIoTEdge@2
  displayName: 'Azure IoT Edge - Build module images'
  inputs:
    defaultPlatform: arm32v7
- task: AzureIoTEdge@2
  displayName: 'Azure IoT Edge - Push module images'
  inputs:
    action: 'Push module images'
    azureSubscriptionEndpoint: 'HIDDEN'
    azureContainerRegistry: '{HIDDEN}'
    defaultPlatform: arm32v7
```

based on the preconfigured target conditions and using a deployment descriptor like the one described previously.

F. Results & Lessons learnt

This section includes the aggregated results of executing the release pipeline on the agriculture prototype for a first deployment of all modules and for a second deployment of a two-modules update. The steps to be executed are: 1) a release manager executes the build pipeline; 2) the build process is executed in an ARM agent node; 3) the generated images of the modules are pushed in an Azure container registry; 4) an edge device is associated with an Azure IoT hub; 5) the IoT hub pulls the images of the modules and automatically deploys the modules in the edge device as specified in the deployment descriptor. Next, two modules are updated (code is changed), so the following steps are executed: 6) the build pipeline is automatically executed and the new version for the modules images are uploaded to the container registry; 7) the IoT hub pulls the images of the modules that have changed and automatically deploys and replaces the modules with the new versions; and 8) the new modules start working by retrieving pending messages from the queue.

Code 3 ARM Agent Properties

```
Agent.Name
rpi-agent
Agent.Version
2.153.2
_
/home/pi/devops/bin/Agent.Listener
Agent.ComputerName
rpi-azure-basic
Agent.HomeDirectory
/home/pi/devops
Agent.OS
Linux
Agent.OSArchitecture
ARM
```

Code 4 CD Pipeline Description in YAML

```
pool:
  name: ARM
steps:
- task: AzureIoTEdge@2
  displayName: 'Azure IoT Edge - Generate deployment manifest'
  inputs:
    action: 'Generate deployment manifest'
    defaultPlatform: arm32v7
steps:
- task: AzureIoTEdge@2
  displayName: 'Azure IoT Edge - Deploy to IoT Edge devices'
  inputs:
    action: 'Deploy to IoT Edge devices'
    azureSubscription: 'HIDDEN'
    iothubname: 'aaas-iothub'
    deviceOption: 'Multiple Devices'
    targetcondition: 'service ="premium"'
```

TABLE I
CI PIPELINE TIMES

Register (minutes)	Pipeline First Execution (minutes)	Pipeline Second Execution - Update (minutes)
1.5	15	4

The collected performance data are about CI pipeline execution time, module image size, and deployment and starting/restarting time. Hence, Table I shows execution time for the CI pipelines, specifically the first time that the pipeline is executed, and subsequent executions after updates. Although the CI pipeline requires some time to detect the available agents, the pipeline is almost instantaneous. The first execution of the CI pipeline requires much more time than later executions because the first execution populates a cache used for

TABLE II
MODULES IMAGE SIZE & START/RESTART TIMES

Module	Size (Mb)	First Run Start Time (seconds)	Update Restart Time (seconds)
weathermodule	132,58	1	3
statusmodule	146,27	1	3
plantmodule	133,28	1	3
messagedispatcherm	143,61	1	3
premiumplantmodule	132,58	1	3
premiumweathermod	143,16	1	3
Node-RED	800,80	3	
postgres-9.6	238,05	2	

TABLE III
IMAGES LAYERS

Module	No. of layers per image	No. of layers in the cache	No. of new or changing layers	Size of non-cached layers (Kb)
weathermodule	9	6	3	50
Node-RED	16	15	1	40

building the module images. This cache stores images composed of *layers*. As the modules share many layers among them, the process of building later images is quite much faster due to the use of this cache.

Table II shows the size of modules deployed in the edge devices and the time for starting them up (the first time) and for restarting the modules after an update which is almost instantaneous. During the first deployment, the most demanding modules were those that are not developed using the Azure SDK (i.e., the Node-RED and Postgres modules). Their images were also the largest and the most complex, although response times of almost all modules were quite fast, as Table II shows.

Table III shows the pieces of codes (layers) that make up a module and the layers that a module shares with other modules. In an upgrade, not all the layers are updated. The last columns show the number of layers to be updated and the size of these noncached layers, and thus, the kb to be deployed due to an upgrade of that module. The *weathermodule* and the *Node-RED* modules were changed, thus only four layers were built, cached, and then deployed. Hence, the time required is shorter since only the layers not previously cached are rebuilt, which speeds up the integration and the delivery processes.

It can be concluded that the prototype for precision agriculture based on the architecture model and the CD process flow here described show the feasibility of releasing resilient software updates at any time, in a readily way, fostering the modernization of that economical sector. The case study showed changes (code updates, shutdowns, and restarts) at the edge without affecting other services and without farmer's participation. Moreover, due to the selected protocol for communicating the LoRa gateway and the edge device, as well as the retention policy defined in the edge device, no messages were lost during the tests, even when connectivity outages were forced.

G. Limitations

CD in the IoT edge shows some limitations regarding testing. Testing is a critical part of the CD pipelines, but edge poses a significant challenge as it sounds unlikely to have a staging environment (i.e., a second farm). Replicating production-like environments is not an easy and cheap task in edge computing, even in certain Web domains that depend on large on-premise infrastructure. Some solutions could address the configuration of CD pipelines to deploy in some segments of the production environment using some well-known deployment patterns, such as canary releasing, feature switches or feature toggling, dark feature, and A/B testing. More case studies are necessary to evaluate the feasibility and effectiveness of these patterns in the edge.

The devices used in the above-described prototype have been selected considering not only they meet the required capabilities but also their low cost. Thus, the deployment of the hardware infrastructure of this approach could be affordable for farmers or for farming solution providers, who could even plan to hire such infrastructure to farmers. This approach enables a flexible infrastructure that can run different applications for the different farming activities changing over the seasons. Nonetheless, the cost effectiveness of this solution has not been measured or compared to others.

VI. CONCLUSION

Although DevOps has demonstrated numerous successful cases in the Web domain, in the IoT domain, there are few reported cases that adopt CI/CD practices in the edge. This work presented a success case of *CD in the IoT edge* by applying a SaaS model that allows release engineers to customize these edge applications.

This article presented an architectural model and a CD flow for the customized deployment of SaaS solutions in the edge and demonstrated its feasibility in an IoT-based agriculture prototype. The instantiation is based on the Microsoft Azure platform and validated the adoption of CI/CD practices in highly distributed IoT systems using edge devices with a runtime as defined in this article. The architecture and the CD flow defined the services and components required for; 1) fetching software updates from version control systems; 2) building software and creation of the container images; 3) publishing the images to a container repository, and finally; and 4) the deployment of these images in the edge devices. A key issue is that we enable the customization of SaaS applications—packaged as modules—based on a set of defined parameters —e.g., user preferences, payment, etc.

The prototype for precision agriculture provides a solution for the edge devices that is flexible enough to support different communication protocols with IoT devices and to operate in areas without requiring the existence of a public communication network, solving the issue with the help of a gateway and using the nonlicensed spectrum for creating its own private network.

The architectural model and the CD flow may enable the development of user-friendly applications for nontechnical users of the prototype. Farmers could eventually decide which services are required and deploy them in edge devices, once the edge devices are connected to the cloud. In this way, farmers configure the most suitable solution for the crop type at each moment of a season. As future work, we also plan to extend our solution to provide firmware

OTA (FOTA) updates in field devices (sensors). One approach to this could be the development of the edge modules capable of managing FOTA updates at field devices.

APPENDIX

The source code of the prototype is available in a public git repository located in https://github.com/rlopezv/aaas that contains the components used for the prototype with information about their content. Additionally, a video demonstrating part of the work done is available on https://youtu.be/gCkv7YQ9RoQ.

REFERENCES

[1] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and Internet of Things: A survey," *Future Gener. Comput. Syst.*, vol. 56, pp. 684–700, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.future.2015.09.021

[2] E. Cavalcante *et al.*, "On the interplay of Internet of Things and cloud computing: A systematic mapping study," *Comput. Commun.*, vols. 89–90, pp. 17–33, Sep. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2016.03.012

[3] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob, and M. Imran, "The role of edge computing in Internet of Things," *IEEE Commun. Mag.*, vol. 56, no. 11, pp. 110–115, Nov. 2018.

[4] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Gener. Comput. Syst.*, vol. 97, pp. 219–235, Aug. 2019, [Online]. Available: https://doi.org/10.1016/j.future.2019.02.050

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[6] A. Dyck, R. Penners, and H. Lichter, "Towards definitions for release engineering and DevOps," in *Proc. IEEE/ACM 3rd Int. Workshop Release Eng.*, Florence, Italy, 2015, p. 3.

[7] J. Díaz, J. Pérez, A. Yague, A. de Antona, and A. Villegas, "DevOps in practice—A preliminary analysis of two multinational companies," in *Proc. 20th Int. Conf. Product Focused Softw. Process Improvement*, 2019, p. 8. [Online]. Available: https://easychair.org/publications/preprint/Z7Zv

[8] L. E. Lwakatare *et al.*, "Towards DevOps in the embedded systems domain: Why is it so hard?" in *Proc. 49th Hawaii Int. Conf. Syst. Sci. (HICSS)*, Koloa, HI, USA, Jan. 2016, pp. 5437–5446.

[9] D. Mattos, J. Bosch, and H. Holmstrom, "Challenges and strategies for undertaking continuous experimentation to embedded systems: Industry and research perspectives," in *Agile Processes in Software Engineering and Extreme Programming*. Cham, Siwitzerland: Springer, May 2018. doi: 10.1007/978-3-319-91602-6_20

[10] H. Bangui, S. Rakrak, S. Raghay, and B. Buhnova, "Moving to the edge-cloud-of-things: Recent advances and future research directions," *Electronics*, vol. 7, no. 11, p. 309, 2018. [Online]. Available: https://doi.org/10.3390/electronics7110309

[11] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of Things applications," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 439–449, Feb. 2018.

[12] A. Baktayan and A. Zahary, "A review on cloud and fog computing integration for IoT: Platforms perspective," *EAI Endorsed Trans. Internet Things*, vol. 4, no. 14, p. e5, 2018. [Online]. Available: http://dx.doi.org/10.4108/eai.20-12-2018.156084

[13] B. Fitzgerald and K. J. Stol, "Continuous software engineering: A roadmap and agenda," *J. Syst. Softw.*, vol. 123, pp. 176–189, Jan. 2017.

[14] G. Casale *et al.*, "Current and future challenges of software engineering for services and applications," *Procedia Comput. Sci.*, vol. 97, pp. 34–42, Jan. 2016, doi: 10.1016/j.procs.2016.08.278.

[15] A. Karapantelakis *et al.*, "DevOps for IoT applications using cellular networks and cloud," in *Proc. IEEE 4th Int. Conf. Future Internet Things Cloud (FiCloud)*, Vienna, Austria, 2016, pp. 340–347.

[16] 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)*, Jeju, South Korea, 2016, pp. 1076–1081.

[17] 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)*, Berlin, Germany, 2016, pp. 301–304.

[18] J. Moore, G. Kortuem, A. Smith, N. Chowdhury, J. Cavero, and D. Gooch, "DevOps for the urban IoT," in *Proc. 2nd Int. Conf. IoT Urban Space (Urb-IoT)*, New York, NY, USA, 2016, pp. 78–81.

[19] A. Banijamali, P. Jamshidi, P. Kuvaja, and M. Oivo, "Kuksa: A cloud-native architecture for enabling continuous delivery in the automotive domain," in *Product-Focused Software Process Improvement (PROFES)* (Lecture Notes in Computer Science), vol. 11915. Cham, Switzerland: Springer, 2019.

[20] T. Abirami and B. Bhuvaneswari, "Energy efficient wireless sensor network for precision agriculture," *Int. J. Sci. Res. Sci. Eng. Technol.*, vol. 6, no. 2, pp. 98–105, 2019. [Online]. Available: https://doi.org/10.32628/IJSRSET196220

[21] P. Lea, *Internet of Things for Architects*. Birmingham, U.K.: Packt Publ., 2018.

[22] *RAK831—RAK The Middleware from RAK Enable IoT*. Accessed: Jul. 02, 2019. [Online]. Available: https://www.rakwireless.com/en/WisKeyOSH/RAK831

[23] *LoRa Server, Open-Source LoRaWAN Network-Server*. Accessed: Jul. 02, 2019. [Online]. Available: https://www.loraserver.io/

[24] *IoT Edge Microsoft Azure*. Accessed: Jul. 02, 2019. [Online]. Available: https://azure.microsoft.com/enus/services/iot-edge/

[25] *Node-RED*. Accessed: Jul. 02, 2019. [Online]. Available: https://nodered.org/

Ramón López-Viana received the B.S. and M.S. degrees in telecommunication engineering from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 1995 and 2000, respectively, and the M.S. degree in Internet of Things from UPM in 2019.

Since 2000, he has been working in software development in different areas mainly in the context of Enterprise Applications. His current interests are focused in Internet of Things, edge computing, cloud computing, and microservices and DevOps.

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

She is an Associate Professor with UPM, E.T.S. Ingeniería de Sistemas Informáticos). Since April 2003, she has been a Researcher with the System and Software Technology Research Group and is participating in several European and national projects related to Software Engineering on Internet of Things and Smart Systems. Her research interests are focused on DevOps and rapid application development, cloud computing, software architectures, software product lines, and model-driven development.

Dr. Díaz was received the best thesis Award from UPM.

Vicente Hernández Díaz received the M.Sc. degree in electronic engineering from UAH, Huntsville, AL, USA, in 2013.

He is an Associate Professor with the Universidad Politécnica de Madrid, E.T.S. de Ingeniería y Sistemas de Telecomunicacion, Madrid, Spain. Since 2005, he has been a Member of the Research Group of Next-Generation Networks and Services. He has participated in several European research projects, and recently in AFARCloud (Precision Agriculture) and SWARMS (underwater collaborative robots). His research activities are focused on ubiquitous computing, Internet of Things, and network resilience.

José-Fernán Martínez received the Ph.D. degree in telematics engineering from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 2001.

He is an Associate Professor with UPM, E.T.S. de Ingeniería y Sistemas de Telecomunicacion. He is the Head of the Research Group of Next-Generation Networks and Services. He has participated in several European research projects, he is leading AFARCloud (Precision Agriculture) and has led SWARMS (underwater collaborative robots), both of them large European research projects. His research activities are focused on ubiquitous computing, Internet of Things, underwater cooperating robotics, new advanced services for WSAN, and resilient systems.