A Platform for Lightweight Deployment of IoT Applications Based on a Function-as-a-Service Model

S. Sansó, C. Guerrero, I. Lera, and C. Juiz, *Senior Member, IEEE*

***Abstract*—The use of Cloud computing for the development of Internet of Things (IoT) applications has emerged during the last years. But there is a lack of a platform which facilitates the deployment and the interoperability of this type of applications. This paper presents a platform to facilitate the deployment of Cloud-based applications to devices in IoT domains. The platform allows the programmers to use a Function-as-a-Service programming paradigm which is managed and configured in a Platform-as-a-Service web tool. The tool also allows establishing interoperability between the functions of the applications. The platform is validated by developing a Cloud application that orchestrates two IoT devices, one with a movement sensor and another one with a camera. Finally, a performance study is also performed. The proposed platform obtains faster and easier deployments of the applications. The resource usages of the IoT devices are also lower with regard to a deployment process based on Docker containers.**

***Index Terms*—Cloud computing, Function-as-a-Service, Internet of Things, Service orchestration.**

I. Introducción

DURANTE los últimos años, se ha producido un incremento importante del número de aplicaciones desarrolladas para ecosistemas basados en el Internet de las Cosas o IoT (*Internet of Things*). En este tipo de aplicaciones, se considera que cualquier dispositivo, por pequeño que sea, es capaz de conectarse a Internet y de monitorizar o controlar elementos físicos. Estos dispositivos recogen, procesan y comparten datos. Uno de los principales problemas de estos entornos es la heterogeneidad de los dispositivos IoT, que provoca que el despliegue de estos entornos sea un trabajo laborioso y costoso [1].

Ha habido algunas experiencias previas que han solucionado parcialmente el problema de la heterogeneidad. Este es el caso de, por ejemplo, el trabajo de Yacchimera y Palau [2] que propusieron un Gateway para solucionar los problemas de heterogeneidad de los dispositivos IoT, mediante la definición de un protocolo flexible que transforma los datos heterogéneos en un formato común.

Con la adopción de las arquitecturas IoT en entornos tan diversos como los industriales (*Industrial IoT, IIoT*), ciudades inteligentes (*Smart cities*), de transporte (*Internet of Vehicles*), o domésticos (*Smart homes*), se ha incrementado la necesidad de ofrecer plataformas para un despliegue rápido y efectivo de aplicaciones IoT. Existen propuestas de este tipo de plataformas, pero estas están, por un lado, basadas en modelos de desarrollo que requieren de herramientas muy específicas y que no están disponibles para todos los tipos de dispositivos IoT y, por otro lado, requieren dispositivos con gran cantidad de recursos, tanto de espacio como de computación. Este hecho también eleva la utilización de los recursos de red cuando se lleva a cabo el despliegue de las aplicaciones. Se han propuesto soluciones que facilitan el despliegue de las aplicaciones, pero que para ello requieren un consumo elevado de recursos. Dentro de este grupo entran las soluciones que se basan en el uso de Microservicos y virtualización de contenedores [3]. Por ejemplo, Krylovskiy et al. [4] propusieron la definición de aplicaciones IoT para Smart City mediante el uso de Microservicios. Pahl y Lee [5] exploraron el uso de contenedores para gestionar las arquitecturas Edge. Un trabajo similar a este último, de Ismail et al. [6], también utilizó Docker para implementar una plataforma de este tipo. En la mayoría de casos, estas soluciones se basan en modelos que generan un mayor consumo de recursos en los dispositivos donde se despliegan y además la disponibilidad de dichas herramientas es más limitada que otras tecnologías más tradicionales.

Estos inconvenientes relacionados con el alto uso de recursos han de ser solucionados mediante la adopción de un modelo de desarrollo que sea capaz de reducir el tamaño y los requerimientos de las aplicaciones ejecutadas en los dispositivos IoT. Las aplicaciones IoT que se ejecutan en los dispositivos suelen ser aplicaciones muy simples que se encargan únicamente de leer datos (monitorizar sensores) o escribir información (accionar actuadores) con ciertas reglas o condiciones. El modelo de desarrollo que se ha dado en llamar *Function-as-a-Service* (FaaS) satisface estos requerimientos, ya que las aplicaciones son desarrolladas como funciones que se ejecutan en cualquier elemento de proceso disponible [7].

Adicionalmente, el aumento de los requisitos, tanto de almacenamiento de los datos monitorizados por los sensores, como de su procesado para la toma automática de decisiones, ha provocado que sea necesario disponer de elementos con mayores capacidades de proceso que los propios dispositivos IoT. Los servicios basados en la nube, o *Cloud Computing*, cubren estas necesidades. Gracias a la interrelación del binomio IoT y Cloud, se pueden desarrollar aplicaciones que

Sesbastià Sansó is with the Universitat de les Illes Balears (UIB), Illes Balears, España, sanso.barcelo94@gmail.com.

Carlos Guerrero is with the Universitat de les Illes Balears (UIB), Illes Balears, España, carlos.guerrero@uib.es.

Isaac Lera is with the Universitat de les Illes Balears (UIB), Illes Balears, España, isaac.lera@uib.es.

Carlos Juiz is with the Universitat de les Illes Balears (UIB), Illes Balears, España, cjuiz@uib.es.

interaccionen con dispositivos reales, gracias al IoT, y que al mismo tiempo tengan capacidades de almacenamiento y procesado ilimitadas, gracias al Cloud. No es nueva la idea de soportar aplicaciones que interaccionan con el mundo real mediante el uso de servicios en la nube y ya ha sido previamente utilizada en otros campos, como por ejemplo la computación móvil o la robótica [8]. O el trabajo de Pradilla et al. [9], en el que presentaron SOSFul (*Sensor Observation Service RESTFul*), una plataforma para recoger y analizar datos de sensores en entornos IoT.

Con la incorporación de las tecnologías Cloud, se pueden definir plataformas para aplicaciones IoT que faciliten y simplifiquen el proceso de despliegue de las mismas. Por eso, en este artículo presentamos una plataforma para el despliegue de aplicaciones IoT desarrolladas usando un modelo FaaS, y cuya orquestación se realiza mediante servicios en Cloud que utilizan un modelo de plataforma como servicio (*Platform-as-a-Service, PaaS*).

II. Planteamiento del Problema

El problema que afrontamos en este artículo es la definición y desarrollo de una plataforma de gestión y despliegue de aplicaciones para entornos IoT basados en el cloud. Los problemas de despliegue de aplicaciones IoT que queremos abordar con nuestro trabajo son los siguientes:

Recursos limitados. Los dispositivos IoT tienen pocos recursos hardware. Por tanto las aplicaciones que se desarrollen para ellos, como las plataformas para el despliegue de aplicaciones, han de consumir pocos recursos. Es por ello que hemos de definir una solución que produzca una baja sobrecarga adicional a la propia aplicación sobre los dispositivos IoT.

Heterogeneidad de dispositivos. Los dispositivos IoT, además de pocos recursos hardware, tienen unas características muy heterogéneas, y los sistemas operativos que incluyen están muy limitados y el software que incluyen puedo no estar estandarizado. Para ello, la solución que se desarrolle ha de estar basada en tecnologías lo más extendidas posible y que se encuentren implementadas en el mayor número de plataformas posible.

Despliegue remoto y automatizado. Los entornos IoT están formados por un número muy elevado de dispositivos. Por lo que, para facilitar la gestión de sus dispositivos, es necesario que sus aplicaciones puedan ser gestionadas de forma centralizada y que sean desplegadas de forma remota y automatizada.

Interoperabilidad entre aplicaciones. Dado que los sistemas IoT son entornos distribuidos, y que las aplicaciones serán habitualmente sencillas para generar un consumo de recursos bajos, será necesario poder definir reglas de interoperabilidad entre aplicaciones. De esta forma se conseguirán flujos de trabajo que incluyan dispositivos de distinto tipo, y que la tarea global realizada sea de mayor complejidad que la que pudiera hacer individualmente una sola aplicación en un único dispositivo.

En la Tabla I, se muestran los requisitos de la solución que cubriría el problema planteado (columna *Solución propuesta*). Además, hemos incluido información de algunos trabajos anteriores que han cubierto parcialmente esta problemática. Bellavista y Zanni [10] proponen una solución basada en Docker, con posibilidades de interoperabilidad y facilidad de gestión. Roca et al. [11] proponen una solución basada en virtualización de funciones propia para arquitecturas Fog. Finalmente, STORm (*Solución inteligente para la TOma de decisiones en un ambiente Residencial*) [12] es también una arquitectura para la gestión de aplicaciones en entornos Fog aplicados a casas inteligentes.

Por tanto, a modo de resumen, una propuesta de solución de la problemática planteada debería de cumplir los siguientes requisitos: (a) Despliegue automatizado de las aplicaciones en los dispositivos IoT que forman parte del sistema; (b) Plataforma genérica y aplicable a dispositivos IoT heterogéneos, basada en servicios y tecnologías estandarizados; (c) Plataforma adaptable tanto a entornos IoT, como basados en computación Fog; (d) Solución basada en una tecnología que requiera de pocos recursos y que mejore el rendimiento en los dispositivos de bajo coste; (e) Interoperabilidad definida por el usuario entre las aplicaciones que se desplieguen en los distintos dispositivos que forman parte del sistema IoT.

III. Solución Propuesta

La solución propuesta soporta la definición de las aplicaciones IoT usando un modelo de FaaS. Esta definición se hace de forma centralizada en un servicio de tipo PaaS, para posteriormente desplegar las funciones necesarias entre los dispositivos IoT. Más concretamente, la arquitectura ha de ser capaz, desde un punto de vista del despliegue de aplicaciones, de: (a) descubrir automáticamente los nuevos dispositivos IoT conectados en el sistema; (b) asociar los dispositivos con las funciones de la aplicación que necesitan ejecutar; y (c) desplegar de forma automatizada las funciones en los dispositivos. Desde un punto de vista del administrador del sistema, la arquitectura ha de ser capaz de: (d) permitir al programador definir funciones que se ejecuten en los dispositivos; (e) permitir la definición de funciones de forma genérica y configurable en cada dispositivo; (f) establecer reglas para la interoperabilidad de los dispositivos.

Nuestra plataforma está dividida en dos capas diferenciadas: la capa de gestión y configuración, tanto de funciones como de dispositivos IoT, y basada en un modelo de PaaS; y la capa de dispositivos IoT, donde se despliegan las aplicaciones IoT basadas en un modelo FaaS que, de forma general, se encargar de los procesos de monitorización o actuación. Dos tipos distintos de comunicación se establecen entre ambas capas: comunicaciones basadas en RESTful (*Representational State Transfer*), que se utilizan para la interoperabilidad entre las aplicaciones y para el descubrimiento de dispositivos; comunicación basada en el protocolo SSH (*Secure SHell*), que se utiliza para el despliegue de las aplicaciones. La Fig. 1 muestra de forma gráfica todos las capas, componentes y elementos de comunicación del sistema.

El código fuente de la plataforma desarrollada ha sido liberado de forma pública y puede ser descargado y utilizado de forma libre por cualquier usuario[1].

[1] https://github.com/awvaarua/servidor_central

TABLA I
COMPARACIÓN ENTRE SOLUCIONES AL PROBLEMA PROPUESTO

	Solución propuesta	Bellavista et al. [10]	Roca et al. [11]	Pereira et al. [12]
Ámbito	IoT	Fog	Fog	Fog
Integración con Cloud	Sí	Sí	Sí	Sí
Tecnología	FaaS	Docker	Fog Function Virtualization	Propia
Despliegue	Remoto y desasistido	Docker Swarm	Orquestador sin decisión del usuario	Manual
Compatibilidad con dispositivos IoT	Amplia	Limitada	Limitada	Limitada
Interoperabilidad	Sí	Sí	Sí	Sí
Uso de recursos	Bajo	Alto	No definido	No definido
Código fuente disponible	Sí	No	No	No

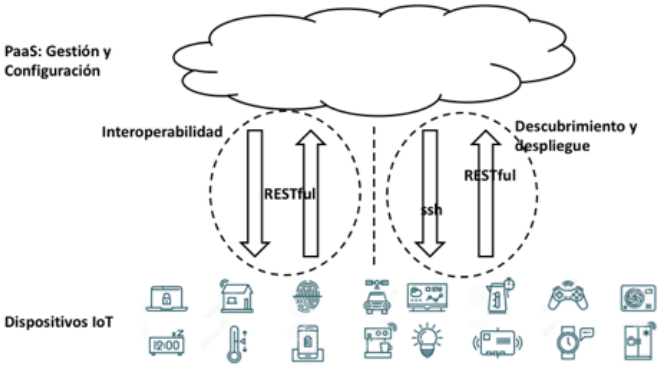

Fig. 1. Arquitectura de la plataforma.

A. Capa PaaS para la Gestión y Configuración

Esta capa es el elemento central de la plataforma, que se encarga de gestionar y configurar todo el sistema. Esta capa interacciona, por un lado, con el administrador del sistema para la gestión de las aplicaciones y, por otro lado, con los dispositivos IoT para el despliegue e interoperabilidad. Típicamente se encuentra disponible en un servidor central del entorno IoT, o incluso, en un proveedor de servicios Cloud. Su diseño se ha planteado como un entorno de PaaS de forma que sobre el usuario administrador solo recae la responsabilidad de gestionar el código de las aplicaciones IoT. Para el diseño de esas aplicaciones se ha optado por un patrón de desarrollo basado en definición de funciones, normalmente llamado FaaS [13].

La gestión y configuración de las aplicaciones IoT conlleva las siguientes necesidades:

Gestionar las funciones que forman la aplicación IoT. Los administradores de la plataforma, o programadores, podrán gestionar las funciones de la aplicación a través de las típicas operaciones CRUD (*Create, Read, Update and Delete*). Además del código fuente del script de la función, también es necesario indicar el intérprete que se ha de utilizar para su ejecución en el dispositivo IoT, indicando el comando que se ha de ejecutar para invocar a la función.

En la Fig. 2 se muestra la pantalla de gestión de funciones, para el caso de una función programada como un script de Python.

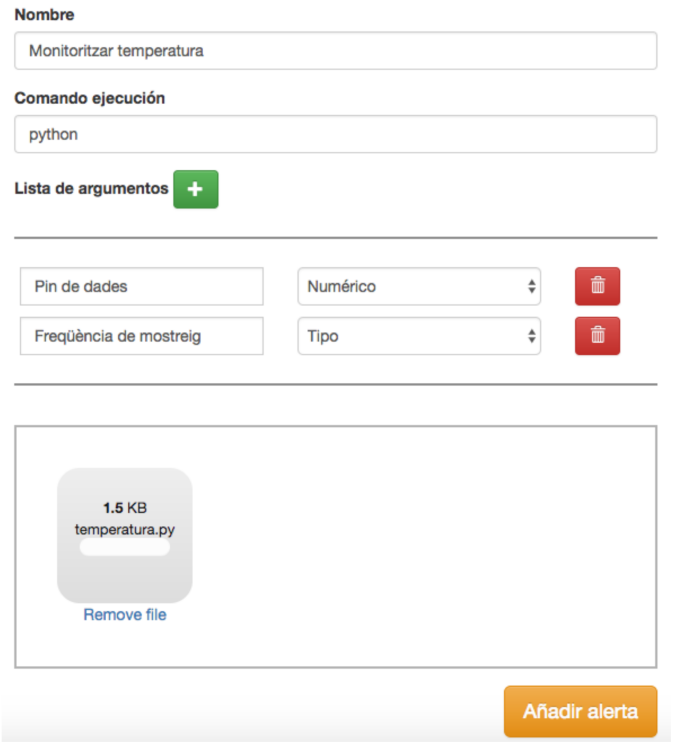

Fig. 2. Pantalla de gestión de funciones.

Definir los elementos configurables particulares de cada instancia de una función. La implementación de las funciones permite incorporar opciones configurables, propias para cada instancia de cada dispositivo.

Un ejemplo ilustrativo es el caso de una función que monitoriza los valores de un sensor. Aunque el código de la función sea el mismo, el puerto a monitorizar donde se encuentra conectado el sensor puede variar. De esta forma, la función que monitoriza los valores podrá ser configurada en el momento del despliegue, bien manualmente por el administrador del sistema, o bien de forma automática con la información proveída desde el dispositivo IoT. Otro ejemplo ilustrativo es el hecho de modificar el tiempo entre envío de los valores monitorizados, ya que estas necesidades pueden variar según el dispositivo o el entorno. La Fig. 2 muestra precisamente una función donde estos dos parámetros han

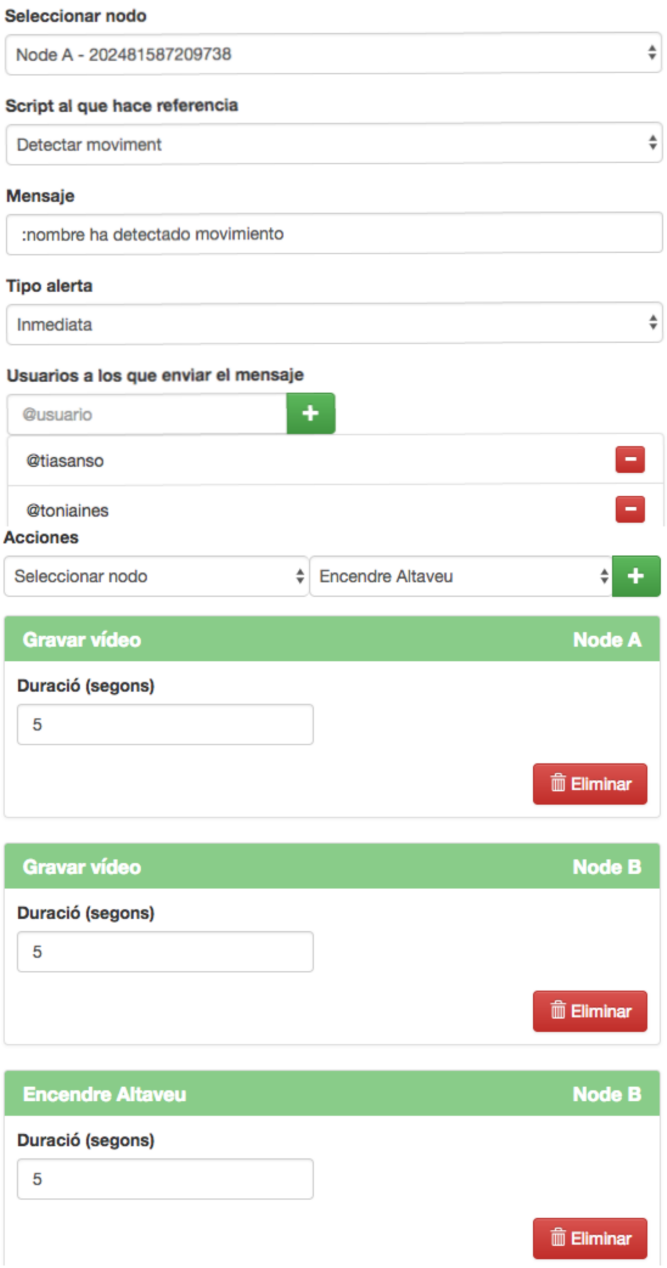

Fig. 3. Pantalla de configuración de las reglas de interoperabilidad de funciones.

sido incorporados para que puedan ser configurados durante el despliegue.

Definir las reglas de interoperabilidad. El administrador del sistema podrá definir unas reglas de la forma condición-acción asociadas a un dispositivo, para que se pueda llevar a cabo una interoperabilidad entre los dispositivos del sistema. Los detalles de esta interoperabilidad se incluyen en el apartado III-D de esta sección. La Fig. 3 muestra la ventana de gestión de las reglas.

Almacenamiento de los datos monitorizados. Como una función adicional, nuestra capa de gestión y configuración depende de unos servicios habilitados para que las funciones que se ejecutan en los dispositivos puedan enviar los datos que monitorizan continuamente para que sean almacenados

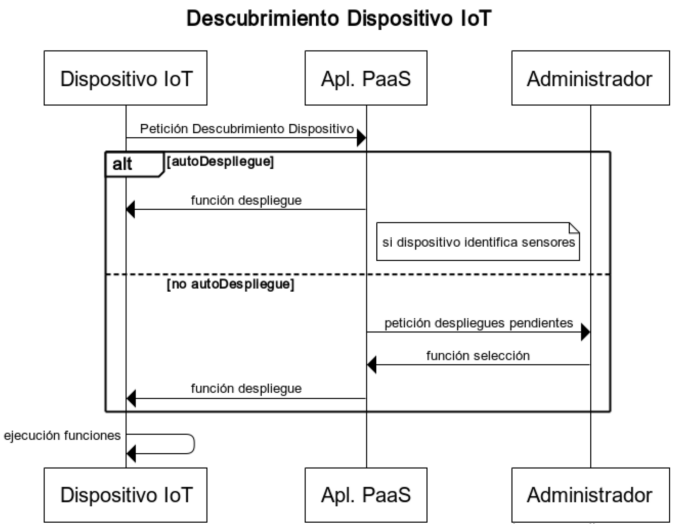

Fig. 4. Diagrama de secuencia correspondiente al descubrimiento de dispositivos IoT y despliegue de funciones.

en una base de datos centralizada. De esta forma, se posibilita el posterior análisis de los datos, incluso utilizando técnicas de análisis masivo de datos. El uso de esta funcionalidad simplemente supone llamar periódicamente a un servicio con un conjunto de datos que se han monitorizado durante un periodo de tiempo. Estos pasarán a ser almacenados en la capa centralizada.

Las tecnologías utilizadas para el desarrollo de la capa PaaS de gestión y configuración fueron MongoDB para la persistencia de datos y NodeJS [14] para el desarrollo basado en un patrón MVC (*Model-View-Controller*) [15], y en el que se utilizaron ExpressJS[2] y Passport[3] como sistema de autentificación.

Definir las reglas de despliegue automatizado. Cada vez que un nuevo dispositivo IoT sea agregado al sistema, la plataforma lo deberá de identificar y asociar con una serie de funciones a desplegar. Esto lo hará el administrador del sistema de forma manual, pero también se permitirá que se asocien reglas para que, mediante las características del nuevo dispositivo, se decida automáticamente las funciones a desplegar. En la versión actual de la plataforma, esta última característica no está totalmente implementada.

B. Descubrimiento de Dispositivos y Despliegue de las Aplicaciones

Esta parte del sistema es la que establece la forma en que el núcleo central de la plataforma (capa de gestión y configuración) descubre los nuevos dispositivos que se conectan y lleva a cabo el despliegue de las funciones que forman parte del conjunto de la aplicación IoT.

La Fig. 4 muestra el diagrama de secuencia que representa el conjunto de acciones que se llevan a cabo cada vez que se añade un nuevo dispositivo IoT en el sistema. El primer paso que se produce es que el dispositivo debe de notificar a la

[2]http://expressjs.com/es/
[3]http://www.passportjs.org/docs/

aplicación de gestión central que se ha añadido a la plataforma, informándole de su dirección IP (*Internet Protocol*) y de sus características, por ejemplo, sensores que tiene conectado. De esta forma, se podrá realizar el despliegue de las funciones que correspondan, e iniciar la ejecución de las mismas. Por tanto, nada más que arranca el dispositivo, este envía una notificación vía un servicio RESTful al servidor central. Por eso, el dispositivo ha de conocer la dirección del servicio (dirección IP del servidor y servicio encargado de añadir el dispositivo).

Una vez que el dispositivo está incluido en el sistema, pueden presentarse dos escenarios. Que el dispositivo conozca y sea capaz de identificar los sensores o actuadores que tiene conectado y que por tanto se pueda llevar a cabo un despliegue automatizado de las aplicaciones que han de ejecutarse en él. Por el contrario, podría ocurrir que el dispositivo no pueda identificar, o no tenga, ningún sensor/actuador. O que no se hayan establecido las reglas automatizadas para desplegar aplicaciones en el caso de los sensores/actuadores que dispone. En este último caso, la aplicación de gestión incluye al dispositivo en una lista de pendientes para que el administrador del sistema decida las aplicaciones que se han de desplegar sobre él.

El despliegue de las aplicaciones se realiza abriendo una conexión SSH entre la aplicación de gestión (que ejerce de cliente) y el dispositivo IoT (que ejerce de servidor), para llevar a cabo la transmisión de las funciones (scripts) que han de ejecutarse en el dispositivo y su posterior ejecución.

Gracias a estas decisiones de diseño, se solucionan una serie de problemas relacionados con la heterogeneidad de los dispositivos IoT. Estos problemas de heterogeneidad hacen referencia fundamentalmente a dos cuestiones: (a) la comunicación y transferencia de datos y servicios con los dispositivos ya que cada uno dispone de unos protocolos de comunicación distintos; (b) los requisitos software del dispositivo para que las aplicaciones puedan ser ejecutadas en él. Las únicas condiciones que han de satisfacer son, para el primer caso, que haya disponible alguna implementación de servidor SSH para la arquitectura particular del dispositivo. Este protocolo de comunicación es de gran popularidad y la mayoría de dispositivos del mercado disponen de implementaciones del mismo, tanto para la parte cliente como para la parte servidor [16]. Y para el segundo caso, el propio usuario será capaz de definir las necesidades software del dispositivo que ha de ejecutar las aplicaciones, o bien, implementar diferentes versiones de la misma aplicación usando distintos intérpretes.

C. Capa de Dispositivos IoT para el Despliegue de Aplicaciones FaaS

Esta capa es la que está formada por los dispositivos IoT sobre los que se desplegarán y ejecutarán las funciones de la aplicación IoT. Estos dispositivos incorporan típicamente sensores y actuadores. Uno de los grandes problemas de los sistemas IoT es la gran heterogeneidad en hardware y software de estos dispositivos. Para solucionar este problema se ha optado por llevar a cabo el despliegue y definición de las funciones utilizando estándares de comunicación que están disponibles en un gran número de plataformas, como es por ejemplo el caso de SSH. El resto de procesos de comunicación se basan en peticiones RESTful sobre protocolo HTTP (*Hypertext Transfer Protocol*), compatible con cualquier arquitectura.

Para el caso de la ejecución de las funciones, es el programador de la aplicación que deberá de escoger un lenguaje que disponga de intérpretes en todos los dispositivos que formen el ecosistema IoT para el que esté desarrollando su aplicación.

Así que, de forma resumida, los dispositivos IoT deberán cumplir dos condiciones: disponer de un servidor SSH y disponer de un intérprete para el código en el que están escritas las funciones de la aplicación.

Finalmente, estos dispositivos deberán de configurarse para que, en el momento de iniciarse por primera vez, realicen una petición RESTful al servicio de nuestra plataforma que se encarga del descubrimiento de los nuevos dispositivos. Para ello es suficiente incluir un pequeño script que se ejecute al iniciar el dispositivo.

D. Interoperabilidad de las Aplicaciones y Dispositivos

Una de las grandes barreras de los sistemas IoT y de entornos inteligentes es la reducida interoperabilidad entre servicios [17]. Por ese motivo, nuestra solución incluye, adicionalmente a la propia interoperabilidad que el programador de la aplicación pueda implementar en las propias funciones, la posibilidad de que el administrador del sistema defina ciertas reglas básicas para de la forma condición-acción. De esta forma, el administrador del sistema podrá hacer que un dispositivo que incorpore un actuador lleve a cabo una acción cuando se cumpla alguna condición sobre los valores monitorizados sobre otros dispositivos. Como ejemplo simple, nos podemos imaginar una vivienda que incorpore un dispositivo con un sensor de temperatura y otro dispositivo con un relé para que active la calefacción bajo ciertos valores de temperatura. La Fig. 3 muestra la pantalla del sistema correspondiente con un conjunto de reglas muy sencillas. En ella, se inicia la grabación de un video de 5 segundos en un nodo A y un nodo B, cuando el nodo A detecta un movimiento en un sensor de movimiento.

Como característica adicional, la plataforma también permite añadir una acción de tipo especial que es la de enviar un mensaje a un usuario de Telegram a través de un robot programado que se desarrolló con ese objetivo.

IV. Resultados

A. Entorno de Pruebas

Existen muchos dominios de aplicación para un ecosistema IoT [18]. En nuestro caso de estudio, hemos escogido un ejemplo basado en una vivienda inteligente (*Smart Homes*). Hemos implementado un banco de pruebas para un sistema de vigilancia de cámaras, con sensores de movimiento que detectan algún tipo de actividad. Cuando se produce este hecho, las cámaras del sistema se encargan de grabar un video de la escena. Durante la grabación, la iluminación de la zona se activa mediante una bombilla que se encuentra conectada a un relé. El video es almacenado en el servidor central, y también es enviado por el robot programado de Telegram al

TABLA II
DISPOSITIVOS DEL BANCO DE PRUEBAS PARA *Smart-Home*

Id.	Función	Componentes
RB1	Dispositivo IoT	Sensor infrarrojo pasivo de movimiento
RB2	Dispositivo IoT	Sensor cámara
		Actuador Relé Bombilla
RB3	Servidor Cloud	—

teléfono del propietario de la vivienda. Para tener más claro el funcionamiento del sistema, existe un video donde se puede ver un ejemplo del mismo[4].

Todo el sistema ha sido implementado utilizando Raspberry Pi 2 model B. La Tabla II muestra la función y los componentes de las 3 placas utilizadas.

Los scripts utilizados para implementar la aplicación IoT del banco de pruebas se encuentran disponibles en línea[5]. Todos ellos han sido desarrollados con Python 2.7. Las placas Raspberry que actuaban de dispositivos IoT fueron instaladas con una imagen modificada del sistema operativo Raspbian, en su versión Jessie, propio de las arquitecturas Raspberry Pi. A la imagen base del sistema operativo, se le incluyeron los scripts Python de nuestro repositorio[6], y se configuró el archivo `/etc/rc.local` para que ejecutara el script `init.py` en el arranque. Dicho script es el que se encarga de realizar la petición RESTful encargada de iniciar el proceso de descubrimiento del dispositivo. Dicha imagen modificada también ha sido colgada en línea[7].

Para el caso de la placa Raspberry que ejercía las funciones de servidor centralizado, se instaló igualmente un sistema operativo Raspbian Jessie y los paquetes `nodejs` y `mongodb-server`, necesarios para ejecutar nuestra aplicación de gestión y configuración del sistema IoT. La aplicación se instaló desde los repositorios que ya se han comentado que contienen todo nuestro sistema de gestión.

La Fig. 5 muestra el detalle de algunos de los dispositivos que forman parte del sistema.

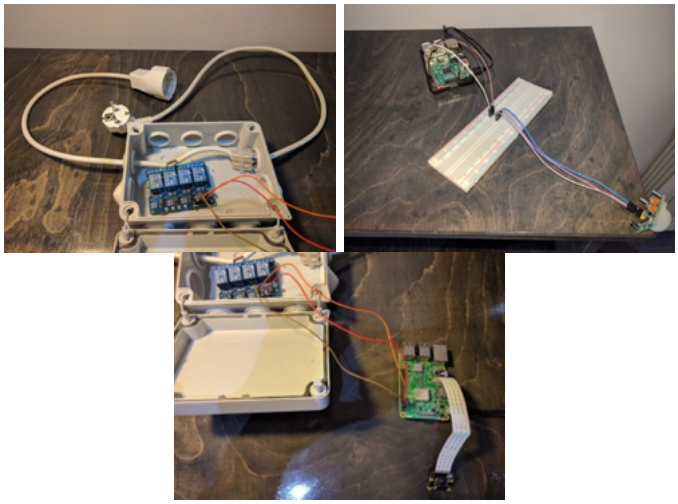

Fig. 5. Banco de pruebas utilizado para el caso de estudio de nuestra plataforma.

B. Resultados y Evaluación

Las dos principales ventajas de nuestra plataforma son la sencillez de despliegue de la aplicación IoT en los dispositivos, y la reducción de los recursos consumidos en dichos dispositivos en frente de otras alternativas, como por ejemplo sería el uso de contenedores (*containers* de Docker) para desplegar las aplicaciones. Otros estudios ya han presentado las mejoras de rendimiento de los contenedores en frente de las máquinas virtuales [19]. Nosotros mostramos que el uso de FaaS aun mejora más el rendimiento de las aplicaciones.

En este apartado presentamos un análisis de la segunda de las mayores ventajas de nuestra solución. Para ello hemos comparado el uso de recursos que se genera con el caso de ejemplo presentado cuando utilizamos nuestra plataforma, y cuando se usa una plataforma basada en Docker, como por ejemplo la que se plantea en [10]. Para ello, hemos medido el uso de recursos (CPU y memoria principal) en el sistema a medida que se incrementa el número de funciones (o contenedores en el caso con el que comparamos). Para ello hemos utilizado el script de la función que monitoriza el sensor de movimiento. La Fig. 6 y la Fig. 7 muestran la comparación entre ambas alternativas: nuestra plataforma etiquetada como *FaaS*; y la opción con la que comparamos, etiquetada como *Docker*, en la que se utilizó un contenedor con la imagen de Python 2.7 incluyendo el script comentado.

Se puede observar como en ambos casos, CPU y memoria principal, nuestra alternativa consume bastante menos recursos. Es interesante destacar que para el caso de Docker solo se muestran los valores hasta los 60 contenedores, ya que la placa Raspberry Pi 2 modelo B no consiguió desplegar un número de contenedores superior, y el sistema acababa cayendo. Tras el análisis de los números, probablemente este hecho se producía debido a que la memoria principal se agotaba.

Si analizamos la utilización de CPU, podemos ver que para el caso de FaaS nos encontramos con un crecimiento lineal, al contrario de Docker, en el que vemos un crecimiento exponencial. Esto se debe a que las ejecuciones de contenedores Docker requiere una sobrecarga adicional por cada contenedor, en contra del caso de FaaS, en el que la ejecución de más funciones no genera sobrecargas marginales. Estas sobrecargas adicionales se producen por el uso compartido de la CPU. Este uso compartido se ve más afectado por la ejecución de contenedores Docker, ya que es necesario la ejecución de un número mucho mayor de procesos, que suponen un mayor número de cambios de contexto en la ejecución. Esto supone una degradación mayor en la utilización de la CPU.

El caso del comportamiento de la utilización de la memoria principal, es totalmente distinto al caso anterior. Tanto la curva de Docker como la de FaaS presentan un comportamiento totalmente lineal. Esto se debe a que el sistema operativo asigna un espacio de memoria fijo tanto a los contenedores Docker, como los procesos de ejecución del modelo FaaS. Por tanto, el uso total de la memoria es el producto entre el

[4]https://youtu.be/3bpwIy2xG9A
[5]https://github.com/awvaarua/monitorizacion
[6]https://github.com/awvaarua/nodo
[7]https://github.com/awvaarua/raspbian_link

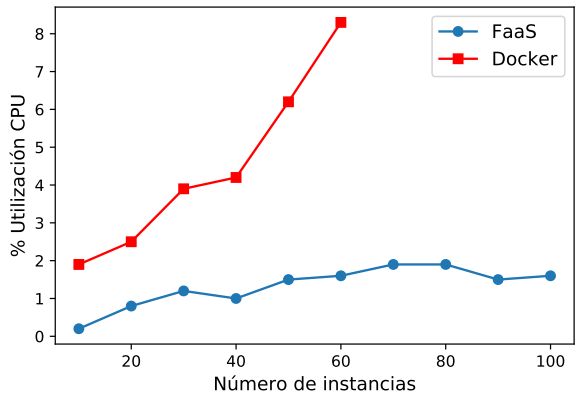

Fig. 6. Comparativa del uso de CPU entre nuestra plataforma y una solución basada en Docker.

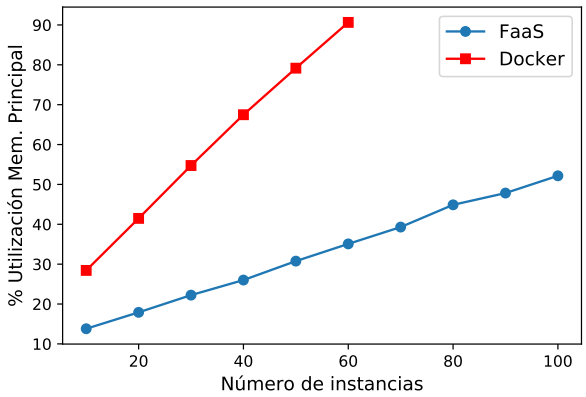

Fig. 7. Comparativa del uso de memoria principal entre nuestra plataforma y una solución basada en Docker.

TABLA III
COMPARACIÓN DESPLIEGUE DE APLICACIONES

Métrica	FaaS	Docker
Tamaño	1,0 kB	350 MB
Tiempo	13 s.	4 min. 23 s.

número de instancias y el espacio de memoria reservado para cada una de ellas.

También queremos remarcar que a pesar de que el uso de la CPU en el caso de FaaS disminuye el a partir de las 80 instancias, esto no es realmente así, y se trata únicamente de un pico marginal. Este comportamiento se observa también en el caso de 40 instancias, en el que se produce una bajada del uso de CPU, pero que, al analizar el uso para un mayor número de instancias, se ve que la tendencia creciente continua.

También medimos los consumos de recursos de los servicios necesarios para implantar cada una de las soluciones. En el caso de nuestra plataforma fue necesario activar el servicio de SSH, que suponía un consumo de 0,3% de CPU y 1,5% de memoria. Por el contrario, los servicios necesarios para Docker, consumían 1,4% de CPU y 5,3% de memoria. De nuevo se comprueba que nuestra solución supone una reducción en los recursos consumidos.

Es interesante remarcar también, otras métricas, como los tiempos y tamaños de transferencia necesarios para desplegar las funciones, o contenedores, en cada nodo (Tabla III). En nuestro caso fue suficiente con transmitir aproximadamente 1 kB del tamaño del script de la función. En el caso de la solución con Docker, durante el despliegue de la primera función, el servicio de Docker se descargó una imagen con Python que ocupaba aproximadamente 350 MB durante unos 4 minutos.

V. CONCLUSIONES

Este artículo presenta una plataforma que reduce la complejidad y el uso de recursos del despliegue de aplicaciones IoT usando un patrón FaaS. A parte de un despliegue autónomo de las funciones, nuestra plataforma también ofrece la posibilidad de definir reglas de interoperabilidad entre dispositivos, a parte de las que puedan establecer los programadores dentro del código de las propias funciones. Finalmente, también existe la posibilidad de almacenar datos obtenidos de los sensores del sistema para que sean procesados posteriormente.

Se ha llevado a cabo una evaluación de nuestra propuesta utilizando como caso de ejemplo un ambiente de *Smart Home*, donde se ha desplegado una aplicación de grabación de video controlado por un sensor de movimiento. Se ha visto que la sobrecarga generada por nuestra plataforma es muy reducida, y que los tiempos de despliegue y gestión también se reducen en comparación a otras alternativas.

Como trabajos futuros, en primer lugar, se plantea la posibilidad de extender esta plataforma a una arquitectura Fog, En ellas, las aplicaciones IoT pueden incorporar funciones que por su necesidad de recursos son ejecutadas en un servidor central (típicamente un servicio Cloud) en lugar de en los dispositivos IoT. Para mejorar los aspectos de latencia, las arquitecturas Fog proponen ejecutar dichas funciones en los nodos intermedios de la red de comunicación. Nuestra plataforma podría extenderse para que se pueda llevar a cabo un despliegue de funciones también en nodos de comunicación y no solo en los dispositivos IoT. Además, esto posibilitaría la integración de políticas de optimización del emplazamiento de las aplicaciones que ya hemos estudiado en trabajos anteriores [20], [21], [22].

Un segundo trabajo futuro sería agregar a la herramienta la posibilidad de definir funciones también con lenguajes compilados y no únicamente interpretados como ocurre actualmente. De esta forma, la plataforma, cada vez que desplegara una función en un dispositivo, debería de reconocer la arquitectura del mismo y compilar un ejecutable para esa arquitectura, para finalmente desplegar el ejecutable en el dispositivo.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Gobierno de España (Agencia Estatal de Investigación) y la Comisión Europea (Fondo Europeo de Desarrollo Regional) a través del proyecto TIN2017-88547-P (MINECO/AEI/FEDER, UE).

REFERENCIAS

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013, including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services Cloud Computing and Scientific Applications Big Data, Scalable Analytics, and Beyond. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000241

[2] D. C. Yacchirema Vargas and C. E. Palau Salvador, "Smart iot gateway for heterogeneous devices interoperability," *IEEE Latin America Transactions*, vol. 14, no. 8, pp. 3900–3906, Aug 2016.

[3] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *Journal of Grid Computing*, vol. 16, no. 1, pp. 113–135, Mar 2018. [Online]. Available: https://doi.org/10.1007/s10723-017-9419-x

[4] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a smart city internet of things platform with microservice architecture," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 25–30.

[5] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures – a technology review," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 379–386.

[6] B. I. Ismail, E. Mostajeran Goortani, M. B. Ab Karim, W. Ming Tat, S. Setapa, J. Y. Luke, and O. Hong Hoe, "Evaluation of docker as edge computing platform," in *2015 IEEE Conference on Open Systems (ICOS)*, Aug 2015, pp. 130–135.

[7] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, *Serverless Computing: Current Trends and Open Problems*. Singapore: Springer Singapore, 2017, pp. 1–20. [Online]. Available: https://doi.org/10.1007/978-981-10-5026-8_1

[8] D. da Silva Pereira, B. Agenor Santana, R. Silva Maia, and A. Souza, "A cloud robotics architecture clone based for a cellbots team," *IEEE Latin America Transactions*, vol. 15, no. 9, pp. 1587–1594, 2017.

[9] J. Pradilla, M. Esteve, and C. Palau, "Sosful: Sensor observation service (sos) for internet of things (iot)," *IEEE Latin America Transactions*, vol. 16, no. 4, pp. 1276–1283, April 2018.

[10] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over raspberrypi," in *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ser. ICDCN '17. New York, NY, USA: ACM, 2017, pp. 16:1–16:10. [Online]. Available: http://doi.acm.org/10.1145/3007748.3007777

[11] D. Roca, J. V. Quiroga, M. Valero, and M. Nemirovsky, "Fog function virtualization: A flexible solution for iot applications," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, May 2017, pp. 74–80.

[12] G. Pereira Rocha Filho, L. Yukio Mano, A. Demetrius Baria Valejo, L. Aparecido Villas, and J. Ueyama, "A low-cost smart home automation to enhance decision-making based on fog computing and computational intelligence," *IEEE Latin America Transactions*, vol. 16, no. 1, pp. 186–191, Jan 2018.

[13] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, "Status of serverless computing and function-as-a-service(faas) in industry and research," *CoRR*, vol. abs/1708.08028, 2017. [Online]. Available: http://arxiv.org/abs/1708.08028

[14] M. Satheesh, B. J. D'mello, and J. Krol, *Web Development with MongoDB and NodeJS*, 2nd ed. Packt Publishing, 2015.

[15] J. David Yanquen Correa and J. Antonio Ballesteros Ricaurte, "Web application deveplopment technologies using google web toolkit and google app engine-java," *IEEE Latin America Transactions*, vol. 12, no. 2, pp. 372–377, March 2014.

[16] D. J. Barrett, R. E. Silverman, and R. G. Byrnes, *SSH, the Secure Shell: The Definitive Guide*. O'Reilly Media, Inc., 2005.

[17] R. Ostos, V. Felix, L. Mena, A. Ochoa, and W. Mata, "Services composition modeling for migrating users in intelligent spaces," *IEEE Latin America Transactions*, vol. 16, no. 2, pp. 662–667, Feb 2018.

[18] S. Agrawal and M. L. Das, "Internet of things — a paradigm shift of future internet applications," in *2011 Nirma University International Conference on Engineering*, Dec 2011, pp. 1–7.

[19] R. R. Yadav, E. T. G. Sousa, and G. R. A. Callou, "Performance comparison between virtual machines and docker containers," *IEEE Latin America Transactions*, vol. 16, no. 8, pp. 2282–2288, Aug 2018.

[20] C. Guerrero, I. Lera, and C. Juiz, "A lightweight decentralized service placement policy for performance optimization in fog computing," *Journal of Ambient Intelligence and Humanized Computing*, Jun 2018. [Online]. Available: https://doi.org/10.1007/s12652-018-0914-0

[21] I. Lera, C. Guerrero, and C. Juiz, "Availability-aware service placement policy in fog computing based on graph partitions," *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[22] C. Guerrero, I. Lera, and C. Juiz, "Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures," *Future Generation Computer Systems*, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18325147

Sebastià Sansó received his Bachelor degree in computer engineering at the Balearic Islands University in 2017. He is currently a computer programmer in an international enterprise and he collaborates in some research activities with the University of Balearic Islands. His main interests are Internet of Things and web programming.

Carlos Guerrero received his Ph.D. degree in Computer Engineering at the Balearic Islands University in 2012. He is an assistant professor of Computer Architecture and Technology at the Computer Science Department of the University of the Balearic Islands. His research interests include web performance, performance optimization, resource management, web engineering, Cloud computing, Fog computing, and IoT. He has authored around 40 papers in different international conferences and journals.

Isaac Lera received his Ph.D. degree in Computer Engineering at the Balearic Islands University in 2012. He is an assistant professor of Computer Architecture and Technology at the Computer Science Department of the University of the Balearic Islands. His research lines are semantic web, open data, system performance, educational innovation and human mobility. He has authored in several journals and international conferences.

Carlos Juiz (A'02–M'04–SM'12) received his Ph.D. degree in Computer Engineering at the Balearic Islands University in 2001. He is an associate professor of Computer Architecture and Technology at the Computer Science Department of the University of the Balearic Islands. His research interests include performance engineering, Cloud computing and IT governance. He has authored around 150 papers in different international conferences and journals.