Internet of Things 22 (2023) 100777

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

Cloud–edge microservices architecture and service orchestration: An integral solution for a real-world deployment experience

Luis Roda-Sanchez [a,b,*], Celia Garrido-Hidalgo [b], Fernando Royo [a], José Luis Maté-Gómez [c], Teresa Olivares [b], Antonio Fernández-Caballero [b]

[a] *NEC Iberica, Madrid, Spain*
[b] *Instituto de Investigación en Informática de Albacete, Departament of Sistemas Informáticos, Universidad de Castilla-La Mancha, Albacete, Spain*
[c] *NEC Europe, London, United Kingdom*

A R T I C L E I N F O

Keywords:
Service orchestration
Edge
Smart cities
Real-world solution
Industrial Internet of Things (IIoT)

A B S T R A C T

After some years in which information management has been centralized in cloud platforms, the need to implement architectures that perform more efficient and intelligent management of such information has become essential. In addition, the growth of Smart City solutions, which involve the deployment of large-scale network infrastructures, has revealed the benefits of processing data as close as possible to the source, a key principle of Edge Computing. However, the benefits of service orchestration in real-world applications have not been demonstrated to date given the novelty and difficulties related to large-scale deployments. To overcome this barrier, this work presents the design, deployment, and validation of a cloud–edge microservice-based architecture for service orchestration on top of a real-world network consisting of 175 nodes. These nodes were deployed as part of a regional network of touristic terminals to raise tourism's potential in the region of Badajoz, Spain. The integration of the openBalena open-source cloud platform into the architecture enables remote and centralized management. The experiments carried out revealed an orchestration latency time to deploy new applications from scratch throughout the entire region between 22 s and 292 s depending on the size of the application image. In addition, CPU load and memory usage have been monitored and the lack of suitability of virtualized environments for interactive applications has been overcome, and a step-by-step guide from a successful deployment is provided. As a result, this integral solution contributes valuable insights into how to bring the benefits of orchestration to reality.

1. Introduction

Application requirements are constantly evolving so that their supporting infrastructure should rely on appropriate technologies and developments to meet their demands. Thus, real-world deployed systems should adapt dynamically to changes or needs and enable straightforward, remote and centralized management. However, the lack of proper architecture, management tools and automated mechanisms in production deployments do not facilitate the needed continuous improvement of the system. Obviously, a hardware replacement is rarely a viable option due to limited access and low modularity. Nevertheless, software should assume the challenge of evolution according to new requirements and advancements, avoiding system obsolescence in a short period of time.

* Correspondence to: NEC Iberica, Product Development Office, 7 Anabel Segura St., Alcobendas, Madrid, 28108, Spain.
E-mail address: Luis.Roda@emea.nec.com (L. Roda-Sanchez).

https://doi.org/10.1016/j.iot.2023.100777
Received 10 August 2022; Received in revised form 29 March 2023; Accepted 31 March 2023
Available online 3 April 2023

For this reason, monolithic architectures and applications are being replaced in recent years due to problems related to rigidity in implementing changes, difficulty of applying upgrades to certain components and complexity of coordination between developers to reach the proper operation. Moreover, when a system is composed of a large number of devices, the traffic between end nodes and cloud can become large enough to cause congestion or overload of servers. In regard to management, such deployments are often located where access is limited, so that more efficient management systems should be implemented in order to achieve centralized and remote administration.

The rise of microservices has enabled the development and deployment of multi-tenant architectures, scalable and technology-agnostic systems which provide resilience to the applications in order to deal with and recover from problems. Regarding the development phase, these architectures facilitate a correct integration of the different services, even though they have been developed separately. However, since they are isolated, communication between components and debugging become complex.

This work addresses the design and real-world large-scale deployment of an end-to-end solution for service orchestration and centralized management based on a microservices architecture. The aim of the described project was to deliver a seamless microservices architecture from cloud to edge, providing the network of devices with centralized management able to orchestrate different services to the edge nodes rapidly and efficiently. The purpose of the application was to bring forth a tool to encourage connected tourism, forming the basis to lead the development of not only a smart city but an entire smart region. To the best of our knowledge, to date, the number of existing works addressing a large-scale and flexible deployment and orchestration of edge nodes in a real-world application setup is limited, which highlights the novelty of this work.

The main motivation of this work is to present an architecture that meets all the current requirements for the deployment of real-world large-scale applications. To this aim, as a summary, the main contributions of the presented work are the following:

- A novel, highly-scalable and flexible microservice-based architecture including the deployment and validation in production in the field of smart cities.
- Service orchestration within seconds and centralized management of the entire real-world network deployed integrating both openBalena and BalenaOS.
- Step-by-step guide and lessons learned from the deployment of the real-world network in production solving significant challenges related to the lack of suitability of virtualized environments for interactive applications.
- Establishment of the foundations for the dynamic orchestration of services and applications based on available resources, shared metadata, and network congestion.

The structure of the paper is as follows. Section 2 introduces a review of other works related to this project. The materials and methods are presented in Section 3. The use case deployed is described in detail in Section 4. The performance experiments and results obtained are discussed in Section 5. Lastly, Section 6 outlines the conclusions extracted during the development of the project, currently in production, and the next steps to further improve the solution.

2. Background

The deployed system is composed of several discrete elements, each deserving to be explored in particular. The Edge is a key element, as the project is underpinned by the idea of implementing an infrastructure capable of providing different services at edge nodes as required. Therefore, some significant papers will be briefly discussed in this section to obtain an overview and provide a sufficiently rich background to understand the contribution of the project.

2.1. Service orchestration and edge

The aim of this section is to provide an overview of studies related to orchestration, in particular, to services orchestration. This field of research is clearly experiencing an enormous growth. Many references to the orchestration concept are connected to 5G networks [1]. Recently, a multi-connectivity 5G network solution has been proposed to avoid network failures in components such as virtualized network functions (VNFs), local servers and communication links [2]. Another study has proposed a system aimed at orchestrating network resources efficiently through the use of 5G network slicing [3]. Furthermore, the authors performed simulations in order to show the adaptation of their proposal to massive IoT scenarios.

However, service orchestration is a wider concept that can be used in different kind of networks. As is shown in recent papers [4,5], a flexible orchestration is a challenge nowadays, even it is a requirement in certain industrial environments. Hence, many researchers are striving to find a solution to deploy services dynamically and in a suitable time frame. An interesting approach is called *FogFlow* [6]. By means of this framework, easy programming and dynamic orchestration of services for Edge devices is enabled. Furthermore, *FogFlow* provides high interoperability due to its standard-based approach. The standardization is a key factor to provide hardware agnostic solutions as well. In this sense, a hyper-connected IoT framework based on a federated architecture through FIWARE IoT platform has been proposed as a solution to improve scalability [7]. Additionally, there are studies related to service orchestration oriented to vehicular networks, as in paper where a simulation based on real data was carried out to evaluate the proposed service orchestration [8].

Other research involving real smart city deployments is particularly interesting, as it is a fundamental factor in the project carried out. Thus, the search has focused on innovative works in this field involving deployments in real environments. Due to the novelty of the underlying technology, the number of papers found is limited. A proposal has presented several Smart Cities use cases and focus on problems related to large-scale IoT deployments [9]. Moreover, a comparison of platforms for IoT was carried out to highlight strengths and weaknesses of different alternatives and the corresponding challenges in the development of management platforms for Industrial IoT deployments. In [10], the authors also review the current challenges of smart cities, where they highlight the variability of real environments, the importance of communications, data interoperability and security, areas with challenges to solve in the near future. Some major benefits that lightweight virtualisation can bring to edge computing environments, specially in smart cities applications, are outlined in [11], as well as authors in [12] remark on the importance of an additional layer between sensor nodes and cloud. A vertical service management solution over the 5GT platform and its testing as through a proof-of-concept testbed has been proposed [13].

In line with smart cities and real projects deployed, pilot studies in Santander city, Spain, and the gold coast in Australia have been described [14]. An IoT framework covering different applications is presented in [15]. Also, a remarkable smart city solution has been provided in [16,17]. It is based on the development of atomic services in order to share components between projects. A real implementation was carried out in 27 cities in Europe and South Korea.

2.2. *Microservices architecture: balenaos and openbalena*

This section focuses on studies related to the microservices architecture used. In particular, the cloud platform known as openBalena and the Linux-based operating system (OS) used in the end devices, called balenaOS, are introduced. The amount of work in the literature related to them is very limited due to the novelty of this platform and OS. However, microservices-based architectures are a research field with high interest nowadays. The architectures proposals based on virtualized components in the smart city domain are focused on different applications such as air quality monitoring [18], urban risk management [19], and power supply system [20].

The comparison between monolithic and microservices architecture is being discussed by the scientific community [21], as well as the proper management of data inside these Cloud–Edge microservices architectures [22]. Even models to decide whether changing to a microservices architecture or continuing using a monolithic one [23] and methods based on knowledge graphs to the extraction of microservices candidates from monolithic applications [24] are present in the literature. The methods available to manage the orchestration of containers are also under study by using different models to the placement of containers to reduce the deployment cost [25]. In addition, microservices architectures are useful for technology agnostic solutions based on Business Process Model and Notation (BPMN) approaches [26].

The use of a complete architecture based on containers provides several benefits. There is an improvement in security [27], since the platform enables the isolation of devices to block inter-device communication as well as the OpenVPN server running in openBalena that grants secured connections. Indeed, different reasons supporting the use of balenaOS have been exposed, highlighting that it ensures a cross-compilation of the solution in order not to depend on a particular hardware [28].

The capability of remote control of devices using balenaCloud has also been mentioned recently in a paper [29], where the use of resources in balenaCloud and Amazon Web Services (AWS) has been compared. It should be mentioned that balenaCloud is the commercial version of the cloud platform. Given the nature of this project we have opted for an open-source version, openBalena, that has been deployed on premise. A framework to deploy deep learning services on edge devices has studied the latency and resources use under different nodes [30].

Some challenges related to the technical and business parts such as reliability, agile software deployments, economy of scale, generic and targeted software updates, initial provisioning and context-specific configuration have been addressed [31]. Moreover, balenaCloud has been cited as a young platform to deploy full-featured container images on Edge devices remotely. Finally, an architecture to evaluate balenaOS power consumption and performance measurements, as well as a comparison with other OSs, has been performed [32].

3. Materials and methods

The main objective of this section is to present the components and the process followed to deploy the edge nodes running a novel OS named balenaOS.[1] and the cloud management platform denominated openBalena[2]. The developments implemented in the fleet of edge nodes and the integration of the different elements in the built infrastructure will also be described. This section is organized into three different parts. Hardware (HW) and software (SW) components are specified at the beginning in order to show the basic elements used to develop and deploy the project. Afterwards, the principal characteristics of the chosen OS are outlined. Lastly, the different elements which compose the cloud management platform are described.

The main HW and SW components are shown in Table 1 and Table 2, respectively. The edge nodes include a small-form-factor computer, namely an Intel NUC5i3MYHE with 4 GB of RAM and 128 GB of storage. Two possible network connections, Ethernet and Wi-Fi, are provided. A liquid-crystal display (LCD) and a capacitive touch panel have been added to allow user interaction. Regarding the software of the edge nodes, the OS selected was balenaOS. The reasons behind the choice of this OS and some significant features will be detailed in the following section. The application involving the chromium browser and Xibo player will be explained in the case study.

[1] https://www.balena.io/os/
[2] https://www.balena.io/open/

Table 1
Hardware components.

	Edge nodes	
	Element	Specifications
HW	Intel NUCi3MYHE	Intel Celeron J1900
	Connectivity	Ethernet & Wi-Fi
	Display	LCD 18.5” (1366 × 768)
		LCD 32” (1920 × 1080)
SW	BalenaOS	2.48.0 rev3
	Supervisor	v10.8.0
	Browser	Chromium 87.0.4280.66
	Xibo-player	1.8-R6 stable

Table 2
Software components.

	Cloud server			
	VM	CPU	RAM	Storage
HW	Rancher	2 cores (2 GHz)	4 GB	8 GB
	LB 1	8 cores (1.8 GHz)	31.4 GB	57.7 GB
	LB 2	8 cores (2.1 GHz)	31.4 GB	57.7 GB
	LB 3	8 cores (1.8 GHz)	31.4 GB	57.7 GB
SW	Rancher	v1.6.30		
	Hosts	Ubuntu 18.04.5 LTS		
	Docker	19.03.11		
	openBalena	v2.0.3		
	Xibo CMS	2.3.7		

With regard to the cloud server, it is composed of four different virtual machines (VMs). The workload is distributed among three load balancers (LB), where each of them is configured with 8 cores, 31.4 GB of memory and 57.7 GB of storage. The remaining machine is responsible for hosting and managing the containers required by the openBalena platform and application-specific services such as Xibo.[3]

3.1. Edge container-based OS: balenaos

The selected OS is a determining element in this project, as the requirements demand a series of special characteristics to achieve the expected management system. In order to understand balenaOS properly, it is necessary to explain briefly what Docker is, since balenaOS is purely docker-based. Docker is a software tool capable of virtualizing environments in self-contained packages called containers by exploiting inherent features of the Linux kernel. Containers are isolated components which can communicate by creating interconnection networks, although each of them has enough elements to operate independently. Docker also adds functionality such as image building, pull and push, management, version control, etc. The term Docker actually refers to the Docker Engine[4] which consists of three main components: a server with a daemon process called dockerd, APIs to interact with this server, and a command line interface (CLI) client.

The developers of the balena tools built a lightweight fork of Docker Engine called balenaEngine. It provides some advantages in relation to smaller binaries and efficient bandwidth usage (binary deltas), conservative management of resources and smaller engine (less than 25 MB, while the standard engine is above 100 MB), and resiliency to power and network failures, thus being suitable for low-power devices. The basic elements of balenaOS are presented in Fig. 1. Systemd is the initialization daemon in charge of launching the rest of the services. balenaEngine provides Docker container support and management of application images. The main services used in this work are *NetworkManager*, which is used to manage the network connections via Ethernet and Wi-Fi; *Dnsmasq* provides lightweight tools and servers as network infrastructure: DNS, DHCP, router advertisement and network boot; *chrony* manages time synchronization in balenaOS via NTP; the VPN service is provided by *OpenVPN*, enabling remote SSH access; and *OpenSSH* is used as server and client to allow remote access through SSH protocol. The supervisor container is used to manage the running user applications, allows communications with openBalena API and checks updates in the openBalena application to ensure that the user containers are running and up to date on the edge device.

A container-based OS, and balenaOS in particular, offers several significant advantages:

- Modularity: isolated containers provide modularity, allowing the development of some features independently, even while the device is running without the need to stop other services.

[3] https://xibo.org.uk/
[4] https://docs.docker.com/engine/

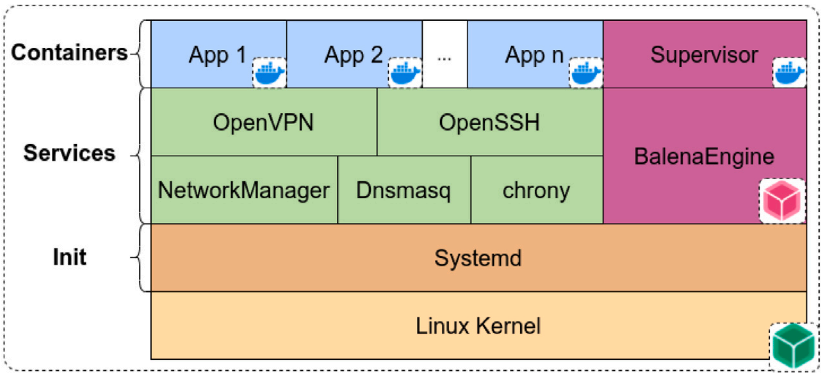

Fig. 1. BalenaOS stack.

- Flexibility: balenaOS is prepared to run on different architectures including ARMv7, ARMv6, ARMv5, AArch64, x86_32 and x86_64. This versatility combined with containerization enables deployment on multiple devices with minimal changes.
- Scalability: balenaOS combined with the cloud platform openBalena provides the capability of managing large volumes of devices in a centralized and simplified form, enabling control over the devices fleet and remote upgrades and troubleshooting.
- Reliability: the use of this type of OS avoids problems related to dependencies, and ensures stable operation under different environments and compatibility among components. It is based on images that remain unchanged, which ensures that devices run the same application, and it provides version control functionality.
- Security: a fully containerized OS isolates the deployed services and applications from the supervisor container and the rest of the OS components. In addition, the straightforward procedure to update services and applications provides flexibility and timely updates necessary to preserve the security of the system.

Nonetheless, there are some drawbacks when dealing with containers. One of the most significant is the complexity when deploying different functionalities. Interaction and data exchange between containers, network and port management, and the implementation of graphical environments often add difficulties. In the following section, these problems are addressed further.

In order to install the OS, a base image has to be downloaded from the official website of balena. Balena provides two possible base OS images: *dev* and *pro*. The *dev* version is used to develop new functionalities, since it offers local deployments (with no need to push images to openBalena) and SSH access without keys. The balenaOS *pro* image is optimized for production environments, since it reduces the image size and requires secure access via SSH. In this project, a single base *pro* image for Intel NUC has been employed. The command *balena os download nuc –version menu –output balena-intel-nuc.img* can be used to download the base intel-nuc image required.

The base image configuration is attached to the application previously registered in openBalena. A *config.json* file can be generated with the proper set of configuration variables. The configuration used in this project is offered in Listing 1, where some data are omitted to show only the relevant information (*domain*, *certificates*, and *sshKeys* cannot be provided due to the confidential nature of data).

```
{"applicationName": "tourism_terminals",
"applicationId": 1,
"deviceType": "intel-nuc",
"userId": 1,
"username": "root",
"appUpdatePollInterval": 600000,
"listenPort": 48484,
"vpnPort": 8443,
"apiEndpoint": "https://api.domain.com",
"vpnEndpoint": "vpn.domain.com",
"registryEndpoint": "registry.domain.com",
    ...
"wifiSsid": "mySSID",
"wifiKey": "myKey",
"balenaRootCA": "[ca.crt]",
"apiKey": "L2BXTwi9LrDtROl8EcoMQ7TTnBz8ukV5",
"os": {
    "sshKeys": [
        "ssh-rsa [AAAAJ4Pdr...M8GB] ..."
    ]
}}
```

Listing 1: Example of configuration file: config.json

As shown, the application name (also known as fleet name) is specified in the configuration file used to build the base image (balenaOS) of the edge device, which means that the node is flashed pointing to a particular application deployed in openBalena. In our scenario, the application named tourism_terminals should be created in advance with the command *balena app create tourism_terminals –type intel-nuc*. After this, an application-custom *config.json* can be generated by using the following command *balena config generate –app tourism_terminals –device-type intel-nuc –version intel-nuc-2.48.0+rev3-v10.8.0 –network wifi –wifiSsid [ssid] –wifiKey [key] –appUpdatePollInterval 10 –output config.json*. As additional parameters, the Wi-Fi configuration, the update interval and the specific version of the OS and supervisor container are specified.

A major requirement for the proper management and future maintenance of the system is a secure remote access. Since the nodes are placed in distant locations, a mechanism becomes essential to apply adjustments, to obtain further information apart from what is collected via Balena API or to solve issues during long-term operation. The procedure of gaining SSH access to the nodes starts with the installation of the *proxytunnel* tool. The balena CLI API key should be generated with the command *balena api-key generate proxytunnel* and saved for future use. The SSH requires a configuration to use *proxytunnel*. The first file is *~/.ssh/config* with information related to the proxy host:port (3128), destination host:port (22222), and the authentication file as shown in Listing 2.

```
Host *.balena
ProxyCommand proxytunnel -p vpn.balena.domain.com:3128 -d %h:22222 -F ~/.ssh/balena-ssh
ServerAliveInterval 30
```

Listing 2: Proxytunnel configuration file

The second file *~/.ssh/balena-ssh* is used for HTTPS proxy authentication, containing two lines with data regarding *proxy_user = root* and *proxy_passwd = [generated by balena api-key generate proxytunnel]*. The permissions should be set properly to grant the owner read/write access to the file. Finally, when production base images are used, the creation of authentication key pairs for SSH is a requirement. Both private and public keys are stored in the *~/.ssh* directory of balena-cli container, and the *id_rsa.pub* key generated has to be added to the *config.json* file to allow SSH access to the terminals, as can be seen at the final lines of Listing 1. The command used for access is *ssh root@[full_UUID].balena*.

3.2. Open-source container-based application orchestrator: openbalena

The balena ecosystem implies the use of a management platform as cloud server. In this case, the open-source option called openBalena was selected. The administration is handled by means of the balena CLI, since there is no dashboard in this open-source version. It provides an interface to manage connected devices through the use of specific commands, which are sent to the openBalena API and subsequently to the supervisor container in the edge device, if necessary.

The devices connected to openBalena are divided into fleets in order to enable grouped management. The main limitation of this management system lies in the fact that the fleet can only include devices sharing the same microprocessor architecture. In addition, to increase reliance on edge devices, although balenaOS images point to a specific application in openBalena, the certificate is necessary in order to achieve the connection between the device and the cloud platform. Once the *config.json* file is properly configured, the command *balena preload /volume/balena-intel-nuc.img –app tourism_terminals –splash-image myImage.png –add-certificate ca.crt* is used to preinstall the current application in the base image in openBalena. In addition, *–splash-image myImage.png* is used to change the default background image and *–add-certificate ca.crt* to include the openBalena certificate to the modified balenaOS base image. Subsequently, when the installation is finished, openBalena provisions the nodes automatically, providing access through the balena-CLI.

With regard to openBalena's elements, the basic services, in this case docker containers, required for the deployment are the following:

- HAProxy: open-source load balancer that provides traffic routing among the different containers.
- API: provides commands to interact with openBalena, manage edge nodes, deploy application updates, etc.
- Registry: manages the images deployed in the platform and enables pulling images to install them in the terminals.
- VPN: grant a secure network for control commands and SSH access to the nodes.
- DB: an open-source database, in particular PostgreSQL, which stores information related to deployed fleets, devices, environment variables, configurations, etc.
- Redis: open-source database used mainly for cache and message broker.
- S3: MinIO open-source storage server where the deployed app images are stored.
- Certs-provider: collects and manages the certificates of the different components in order to provide secure communication between elements.

Apart from the basic containers, a balena CLI could be included to collect information, deploy application updates in the containers and manage devices through balena commands.

The management of the environment variables provides interesting features. There are different levels of environment variables depending on the desired target. This option offers flexibility, since variable values can be different depending on the service, the device or the concrete fleet, and it can be configured at any time during execution. The use of environment variables also provides an improvement in terms of security, as sensitive data are not included in the base code. Priorities are also defined, where the order of precedence from high to low priority is as follows: device service specific variable, device variable for all services, fleet service specific variable, and fleet variable for all services. Therefore, if a device specific variable is set with the same name as a fleet specific variable, as the first one has higher priority, the device specific value will be used.

4. Use case

This section is intended to provide details about the development of specific functions to implement the desired end-use application. The purpose of the project was to deploy a network of edge nodes or terminals scattered across 175 cities and towns in the Badajoz region, Spain, in order to create a regional network to promote and digitize tourism. The terminals included a display to provide touch user interaction in order to browse through the map of the region, find the most significant monuments, nature and to plan the itinerary, being able to carry the map on their own smartphone.

Regarding the hardware components, as mentioned previously, each edge node is composed of a small form factor computer, a USB camera, a display and a touch panel, which can be either 18" or 32" in size, two USB charging ports and a reinforced cover to prevent damage. The devices are configured to be interchangeably connected via Ethernet and Wi-Fi.

However, the significant elements of this project are related to software. As mentioned above, the OS installed, so-called balenaOS, provides interesting features to deploy applications and control the nodes remotely. Together with the open-source cloud platform used, openBalena, a complete microservice architecture has been deployed in a real-world project. Thus, the entire solution, from the cloud to the edge nodes, is built on top of docker containers.

Microservices architectures provide several benefits, but also involve development difficulties owing to the proper OS and the containerized environment. In the following sections the developments carried out to deploy the regional tourism network and issues overcome will be explained.

4.1. Design requirements

The novelty of the fully containerized architecture posed a challenge when implementing both the server side and the edge devices. Fig. 2 illustrates a diagram of the different specific components deployed and integrated in this project, as well as the connections between them and the ports used for communication.

The edge node commands sent by the terminals (represented in light red colour) to the cloud platform are divided into two groups: VPN and non-VPN commands. The F5 firewall (in purple colour) performs tasks of proxying and redirecting to the internal server. VPN commands are tunnelled through an alternative, more secure route, so different ports are defined to separate this traffic. VPN commands use 8443 as destination port and commands related to API, registry, S3 and Xibo are sent to port 443. The latter requests reach the openBalena's HAProxy via SNI directly. Instead, VPN requests are redirected to the CONNECT proxy in order to establish a tunnel that enables communication.

The following components, all of them docker containers, are managed by Rancher (v1.6.30). This open-source platform allows the orchestration of docker containers and facilitates the management of multiple containers across multiple hosts. In this project, three separate hosts were configured to host the different components of the system and ensure high availability services in a future version. Hence there are three LBs, each deployed on a different host, in order to ensure that all of them know where to route the packets. This first stage of LBs, which appear in dark blue colour, manages the traffic that comes from the terminals. The second stage of LB, called offloading (also in dark blue colour), decrypts or encrypts HTTPS traffic to enable internal HTTP communications in order to facilitate and accelerate the information exchange inside openBalena containers.

After the LB stages, the services deployed for the correct operation of openBalena are depicted in light blue colour. As can be noticed, port 80 is used for the communication between components internally, except for special commands or SSH connection from balena-CLI to VPN, which uses 3128 port to establish a secure connection. The so-called VPN commands are related to operations that involve acting on the nodes, as opposed to commands that only request information. These commands imply a risk and thus additional security measures must be taken. Some special commands are those concerning rebooting, shutting down the node and setting up the SSH connection.

The HAProxy acts as an LB, storing the self-signed certificates of the services included in openBalena stack. Furthermore, in the first LB stage, VPN commands from port 8443 are redirected to VPN service in port 3128 through HAProxy.

4.2. Edge nodes: Requirements and developments

The objective of the edge nodes, terminals in this particular application, is twofold. When the terminals are being used by a tourist or citizen, they have to allow a proper surfing on the Internet. Each device is pointing to a specific URL created for that place. If the terminal is inactive, i.e. no one is touching the display for a certain time, the device should show a series of multimedia content that is managed by the organization in order to select the information displayed depending on the season, place, interests, local festivals, regional information, etc.

Regarding the browser-based user application, chromium was selected as it is open source and compatible with the set of hardware and software elements used. This browser is the basis on which a wide variety of companies develop different features and offer their own products like Google, Opera, or Microsoft recently.

The application developed has to ensure safe browsing for all users and for the system itself, which is why it was considered appropriate to use kiosk mode, i.e. to reduce the options for user interaction with browser tools. In addition, it was necessary to add a virtual keyboard extension so that users could enter characters in the fields that required it, and to enable camera permissions to be used in the browser. The configuration of both tasks inside setup files to be ready in advance of running the container was a challenge.

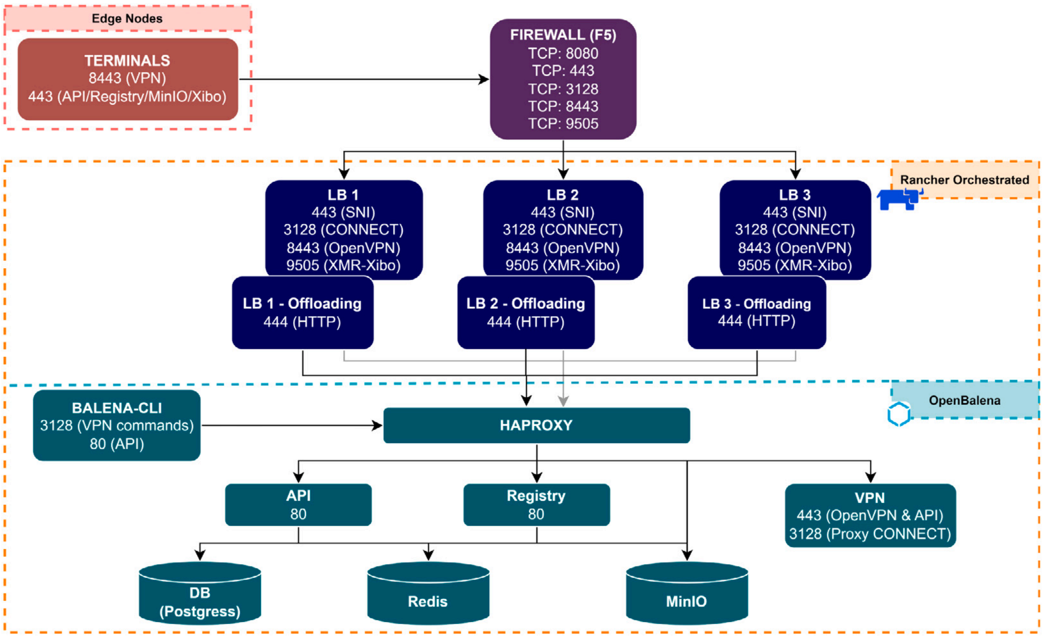

Fig. 2. Deployed microservices architecture.

Apart from the challenge arising from the application to be developed and owing to the OS used, there were some problems associated with the hardware control in a containerized environment. As the terminals would be placed in different layouts, it was necessary to establish mechanisms to control the rotation of the display and the touch panel. It was necessary to use environment variables embedded in the developed code to adjust the devices to each application, size and orientation. These environment variables are set up in openBalena, which, as explained previously, defines a hierarchy between them. This hierarchy is used to provide three levels of environment variable: app level, service level and device level. In this work, two of them were used. App or fleet variables (fleet variable for all services) are applied to all devices provisioned by the app selected, while device variables (device variable for all services) override app variables values in case of managing the same environment variable. As functionality is contained in a single service, no other levels were required. Fig. 3 shows more details on the configuration process, the start-up of the services and the operation, which will be covered throughout this section.

The solution adopted was to employ these environment variables in the code to increase flexibility in order to adapt the system to suit any configuration. In total, five environment variables are set in openBalena to provide the required flexibility, namely LAUNCHER_URL, TOUCH_ORIENTATION, WINDOW_ORIENTATION, WINDOW_SIZE, and TIMEOUT. The hierarchical level of priority for each type of environment variable is used to enrich the options that the system provides.

The first environment variable set is prescriptive to point each terminal, more specifically, the chromium browser that will open in kiosk mode, to the corresponding URL. The display size variable, which allows to adapt the content to both screen dimensions, 1920 × 1080 or 1366 × 768 depending on the terminal. The screen orientation variable was essential to configure portrait mode instead of landscape. The calibration matrix of the touch panel should be modified depending on the orientation as well, since the touch panel is independent from the display. The value of this variable refers to the path of the corresponding configuration file. Finally, when the application requires a switching between windows, as will be explained afterwards, a timeout was configured to set a threshold time to change the foreground window if no user interaction is present.

The project required the deployment of two different applications in openBalena, since a reduced group of devices are used as purely informative devices showing content managed by the organization and the rest are provisioned by another app with dual functionality, user interaction and content display. Therefore, the first version is a variant that includes only a part of the complete application, hence the full version will be discussed.

With regard to the complete application, it involves two main functionalities: allow the user to explore the municipality's resources via Chromium browser pointing to the corresponding URL and display content with Xibo player when not in use. The integration of both functionalities was a key point in the development of the final application. There are two possibilities, to build two separate containers or a single container that combines both solutions. The latter option was selected, as both applications make use of the same resources, such as display, touch panel and windows manager, and allow the application to be orchestrated and managed using a single component. In addition, the deployment of two separate containers, killing them and running on demand, implies a delay when the application has to switch, not providing acceptable usability, particularly in switching from Xibo to the browser, when a user touches the screen after a long period of inactivity.

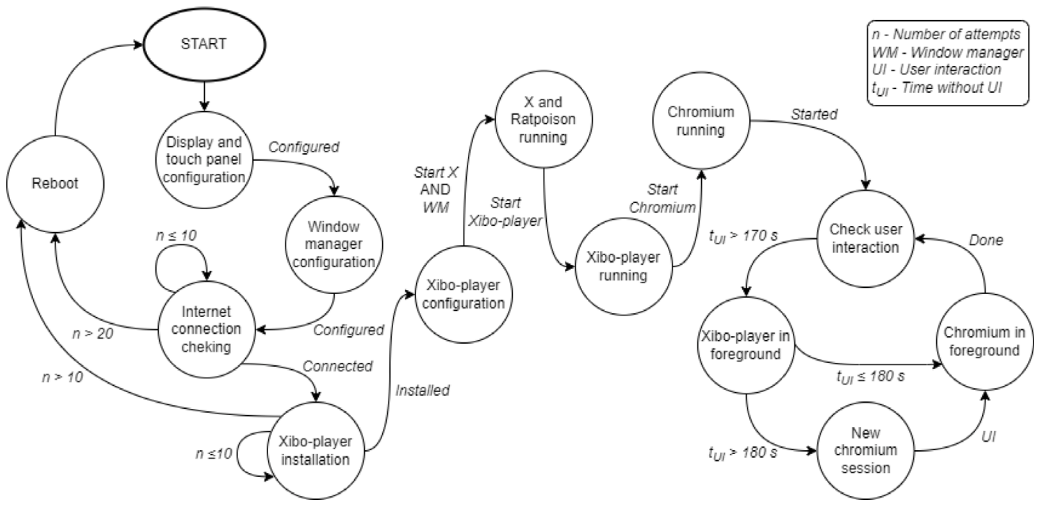

Fig. 3. Application container start-up and operation state machine.

The service deployed to display dynamic content is based on Xibo. This is an open-source software, which enables the display of multimedia files managed by a content management system (CMS). The client is called Xibo player and it is linked with the Xibo CMS, which manages the connected devices, receives status reports and sends management commands and content to the clients running in the terminals. There was a main issue for the installation of Xibo player and it is the installation method. The package is installed by means of snap, which is a packaging tool used to build independent self-contained software, similar to docker containers. The main issue appeared when trying to deploy a container inside a container. Snap packages use snapd daemon to run their containers and need squashfs file system. Thus, it was mounted at FUSE level to enable running the snap of xibo player inside the deployed docker container.

The management and operation of both services is not trivial. Some issues had to be solved in order to achieve a proper and fluid operation of the solution adopted. As well as the integration of the two services in a single container, the start-up was carefully programmed and both services are running at the same time to guarantee a smooth, reliable and responsive operation. Fig. 3 also shows the main stages and actions adopted to provide an optimal service, both from a technical and user interaction perspective.

Once the application container is running, the configuration stage begins. In this phase, there is a set-up of the display and touch panel regarding orientation, size and calibration matrix. The next stage checks whether there is Internet connection, or has to establish it and install xibo player, in order to ensure the proper operation of the application. Due to the lack of basic components in balenaOS, a window manager had to be installed to allow window switching between services when needed. The selected window manager is called Ratpoison, which is a lightweight window manager that allows the control of foreground window through commands embedded in shell scripts in our docker image. The start-up of xibo player and chromium is carefully designed to guarantee a correct operation, especially in the case of xibo player, which requires a considerable launch time due to the amount of content to be downloaded for display.

As soon as the system is up and running, a script that was developed to check the user interaction through the touch panel comes into play. The periods of inactivity are measured. When a watchdog consumes 170 s a command is sent to switch to xibo player window. After 10 s from the switching, the previous browser session is cleared in the background to prepare the terminal for the next user. The opposite action occurs when a user touches the screen. In this case the idle period ends and chromium is instantly displayed.

In order to ensure a proper operation from the installation of the terminals, a preloaded image of the openBalena application is flashed in the edge nodes. This prevents the terminal from getting stuck and provides service from the initial deployment. Once the node is provisioned by openBalena, it checks for updates automatically.

4.3. Deployment

The deployment of the region-area network of devices consisted of 175 nodes, of which 165 include 18" displays and 10 terminals built with 32" displays. In Fig. 4 two examples of installed terminals can be seen, on the left, the 18" terminal installed in Puebla de Obando is presented, while on the right, a detail of the device located in La Roca de la Sierra is shown. In relation to the logistics of the installation, the same configured OS image was flashed in all devices in order to simplify the installation procedure. A shared base image allows to speed up the installation process, since no specific image needs to be configured for each node. This is possible since, when provisioning in openBalena, a UUID is automatically generated for each balenaOS image. Nevertheless, this leads to some issues when reinstalling the base image, as the nodes already provisioned by openBalena are not automatically removed from the database.

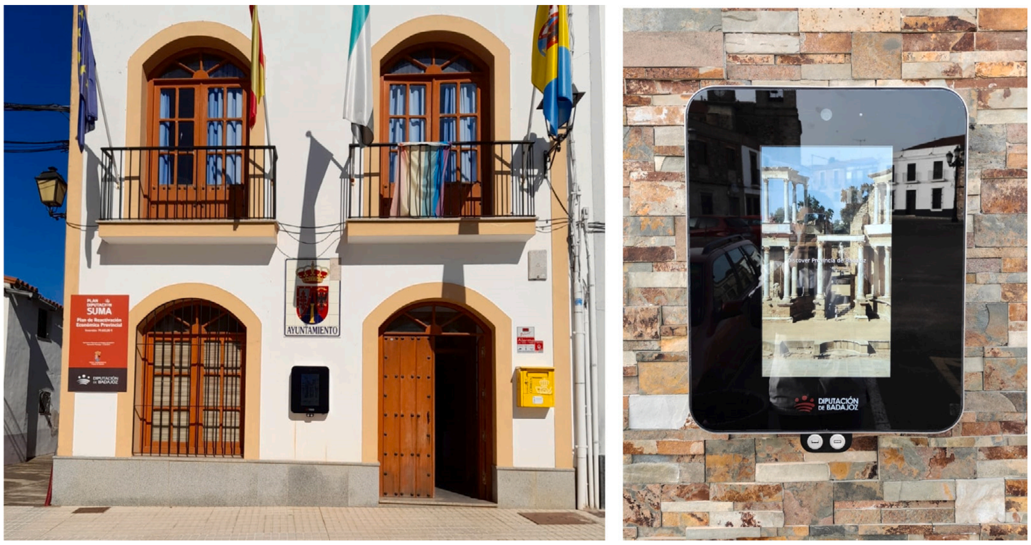

Fig. 4. Wall-mounted terminals in Puebla de Obando (left) and La Roca de la Sierra (right).

In addition, a further problem faced by a single-base image installation is the association between the generated UUIDs, which can be obtained through the balena CLI and the actual nodes. In large-scale deployments, managed remotely, and spread over a wide area, a reliable association mechanism is essential to avoid any node being left undetermined or requiring a visit to the site. The solution adopted to recognize each node is to use the MAC address of the interface in use. Once this address is known, the corresponding terminal address is obtained by means of basic Linux commands, establishing an SSH connection through balena CLI. In this way, the names of each one are configured according to a template containing the date of installation, a number given to each location, and the display inches, with the following structure: [YYMMDD]_[*place number*]_[*display inches*] (e.g. 211021_256_18). Thus, important information can be known through the name associated to each node, with no need to access remotely to obtain such details.

An example of a fragment of the output command *balena devices* can be observed in Table 3. From left to right, columns are divided into the identification and UUID provided automatically by openBalena; the device name, which is assigned by the network administrator; the device type, in this project all are Intel NUC; the application name, where you can see two applications that provision devices intended to provide information or interaction with the user for tourism purposes; the status provides information related to the device compared to the application, i.e. if the device is downloading a new image, idle because it is up to date, and so on; the next column shows whether the node is online or not, when the configuration and VPN connection is correct; finally, the supervisor and balenaOS version installed in each node.

The first row shows a regular 18" terminal, provisioned by the tourism application and properly connected, configured and active, since the output shows that the device is online. The second device is offline, which may be due to different reasons: the terminal is powered off, the VPN connection is not properly configured or the device was re-flashed and this base image no longer exists. This is one of the major issues to avoid, as the lack of connection prevents remote management of the device. The last node is provisioned by the application intended to provide information to the users. In addition, the device name contains details of the size, which in this case is 32". The last node is provisioned by the application aimed at displaying information to the users.

The installation using the same base image for the entire deployment implied some drawbacks with regard to Xibo CMS as well. In the Xibo server, two terminals are not allowed to have identical display's hardware keys, which are automatically extracted from the MAC address of the interface in use of each container. As they are containers built from the same image, their MAC addresses are equal. Therefore, the solution had to solve two issues arising from this restriction: the impediment of having the same hardware ID for several terminals and the incapacity to distinguish between them in Xibo CMS. Thus, in order to solve the display-node association issue in the Xibo server and ID duplication, a specific environment variable was set to modify the default hardware ID of each display as the UUID provided by openBalena. Hence, it is possible to link each Xibo server display with the location and node in the openBalena database. Afterwards, taking advantage of the manual interaction required to authorize the new client to consume the Xibo server, the name is modified in the Xibo CMS to match the one specified in openBalena.

A real picture of the network deployed in the region of Badajoz can be seen in Fig. 5. As can be noticed, the capacity to be remotely managed and the service orchestration implemented are key developments in order to accomplish the project successfully.

5. Performance evaluation

The evaluation of the performance of the deployed network is not a trivial task, as it depends on numerous variables, many of which are inherent to the infrastructure already in place at the deployment location. Nevertheless, there are some interesting insights that are discussed in this section.

Table 3
Example of *balena devices* output from balena CLI.

ID	UUID	Device name	Device type	Application name	Status	Is online	Supervisor version	OS version
138	eecfa1b	210901_148_18	intel-nuc	tourism_terminals	Idle	True	10.8.0	balenaOS 2.48.0+rev3
171	d26ce78	210901_164_18	intel-nuc	tourism_terminals	Idle	False	10.8.0	balenaOS 2.48.0+rev3
234	078b042	210930_156_18	intel-nuc	tourism_terminals	Idle	True	10.8.0	balenaOS 2.48.0+rev3
242	c41b741	211214_242_32	intel-nuc	info_terminals	Idle	True	10.8.0	balenaOS 2.48.0+rev3

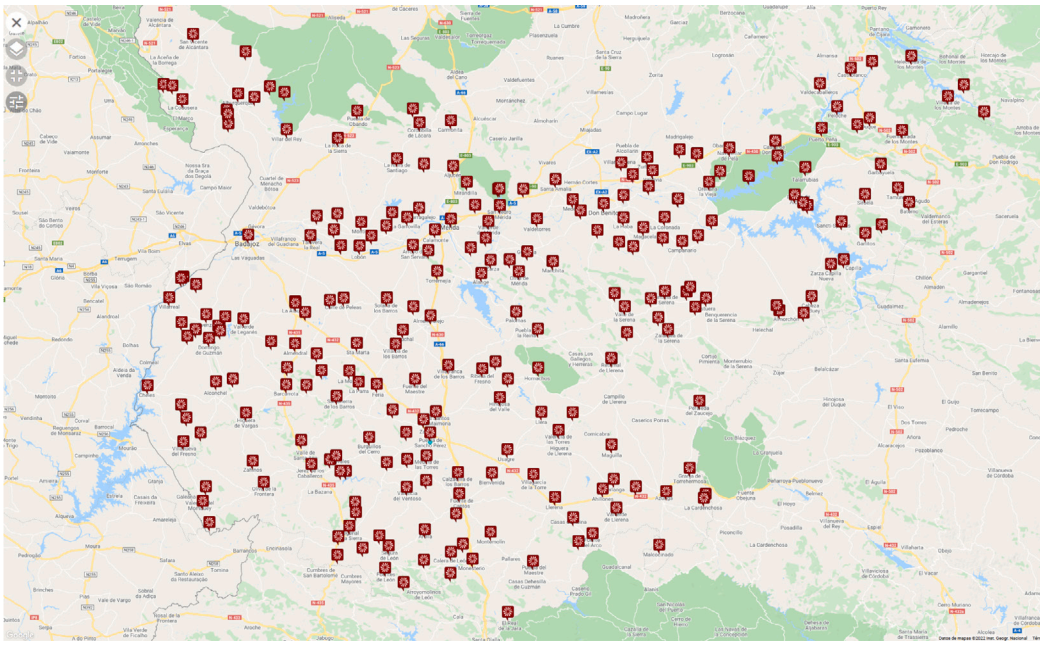

Fig. 5. Global map of nodes deployed in the region.

First, it was considered interesting to measure the resource usage of the network nodes once the application is deployed in production. To this aim, and with the intention of intervening as little as possible in the measurement, the collection of CPU load and memory usage data was configured employing basic Linux commands to avoid the use of complex monitoring applications that could affect the results. The raw data gathered were processed afterwards on a machine outside the network. The measurements were performed over 10 days on four nodes of the network and, in order to obtain a wider view, four nodes corresponding with places with different characteristics were selected. One of them is located in a large city, one in a medium-sized town, and the other two in small villages. Fig. 6 shows the CPU load and memory usage of these four nodes.

Due to the confidentiality of the data, the names of the specific places are not provided, instead, the actual names were replaced by generic ones. Each parameter was measured with a sampling frequency of 5 min. Regarding the charts, the maximum CPU load is set to 400% due to the hardware used. The small-form-factor computer in this edge node includes 4 cores, corresponding to the maximum load of every core. As explained previously, the OS is running two containers constantly. However, the CPU load and memory usage of the supervisor are around 0.02% and 2.15% respectively. These values are negligible with respect to the obtained for the application container deployed and were not depicted on the performance chart.

As can be observed, the CPU load is mostly steady in all cases, reporting around 50% of the maximum load, while the memory usage varied between 20% and 50%. These values are acceptable taking into account the relatively low-power device in use and the type of content they are displaying permanently. During this performance evaluation, the edge nodes were displaying high-resolution videos and photos of the corresponding places.

Apart from resource usage, the service orchestration latency was considered to be an interesting feature to explore in this novel architecture deployment. For this reason, an experiment was designed to measure the total time spent in the deployment of a new application in the edge nodes. The measurement starts when the new application or service is pushed to the cloud platform. Once the new application image is uploaded, the supervisor containers in the edge nodes will check that there is a new version available and they will download it. Finally, the nodes will mount the image and ensure that has been downloaded without errors in order to replace the running application with the new one.

In addition to the deployment procedure, docker uses a system of layers in the files used to configure and deploy containers to speed up the process and reduce data transfer. Thus, modifying the first layers of the image does not influence it to the same extent as modifying the last layers. In other words, if the new application image includes minor changes, the deployment time will

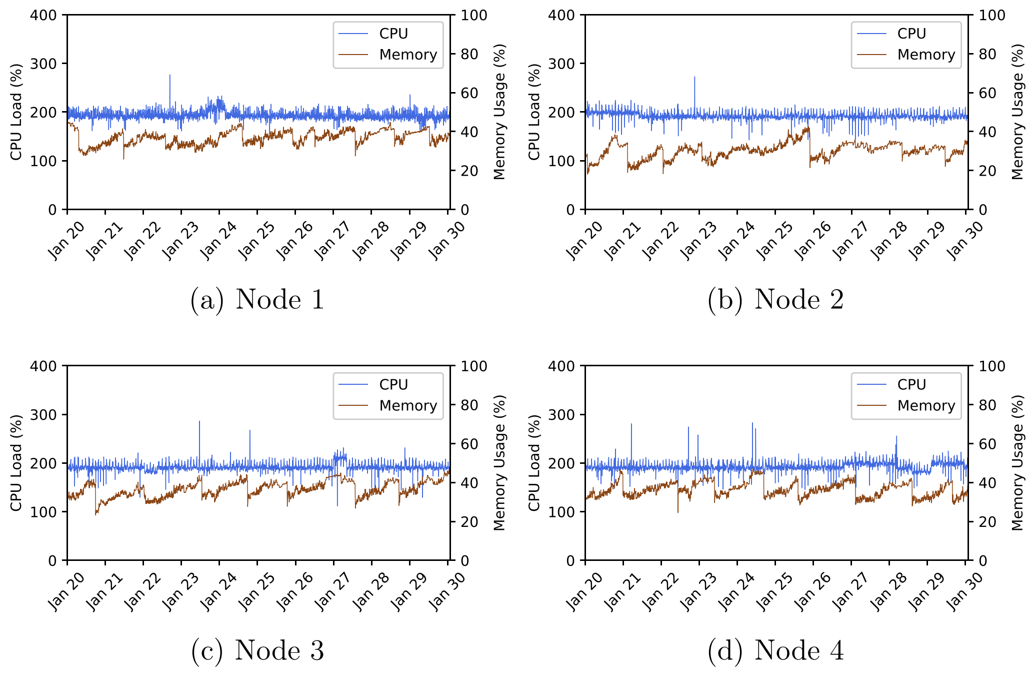

Fig. 6. CPU Load and Memory Usage of four nodes.

be reduced because the images share a large part of the information. This was considered in the experiments by editing the first, intermediate, and last layers in order to observe the difference in deployment time.

The latency time in the deployment of the application involves the upload of the image from an external machine to the cloud platform, the download from the nodes, the corresponding configuration, and the final deployment and start-up of the application container. In order to provide a controlled environment and avoid the different characteristics of such a dispersed, heterogeneous network with diverse configurations affecting the final result, a network composed of four nodes (Raspberry Pi 4, 1 GB RAM) was deployed to carry out the corresponding latency tests. These nodes were provisioned by balenaCloud and the same image that is running on the edge nodes of the production network was deployed and the deployment in each setup was carried out at least 10 times. The result of the tests can be seen in Fig. 7. The changes introduced in the first layers have a greater impact than in the intermediate and last layers, reaching differences up to 270 s in the deployment of the application. The maximum deployment time obtained was 292 s, while the minimum was 22 s. As can be observed in Fig. 7(a), the latency time for nodes 2 and 4 was higher than in the rest. The reason is that these two nodes were located in places with low-speed connections (below 100 Mbps) and high-traffic networks to observe differences under different conditions.

With the aim of knowing the end-users adoption and determining the effect on the use of the resources, the number of sessions started in the different locations was also monitored. Fig. 8 shows the number of browser sessions that took place between 2022 December 23 and 2023 January 15. During vacation time the number increased considerably in three locations, reaching up to 26 sessions in a single day.

Hence, from the results obtained, we can appreciate that the application deployed using the proposed microservices-based architecture maintains a CPU and memory load of less than 50% of the total capacity, even when using resource-limited devices (2 cores and 4 GB of memory). In addition, there were no significant changes in CPU and memory load while the devices were in use, since, in order to favour the user experience, the applications are constantly running. In addition, the applications are not highly dependent on the usage of the terminals, as the most resource-demanding application is the Xibo client, which requires continuous processing of multimedia content.

Regarding the deployment time, in the experiments carried out, the application was deployed within less than 100 s in all cases if only the logical components are changed and less than 292 s if the changes involve every layer of the container image, e.g. when the operating system running inside the container needs to be replaced by a different one. Obviously, the deployment time depends on several aspects such as the content of the application, packages included, network properties and status, and device resources.

6. Conclusions

Network and service management requirements are constantly evolving. Monolithic architectures often do not meet today's requirements for highly dynamic systems with enough agility and modularity to tackle changes within a short period of time. In

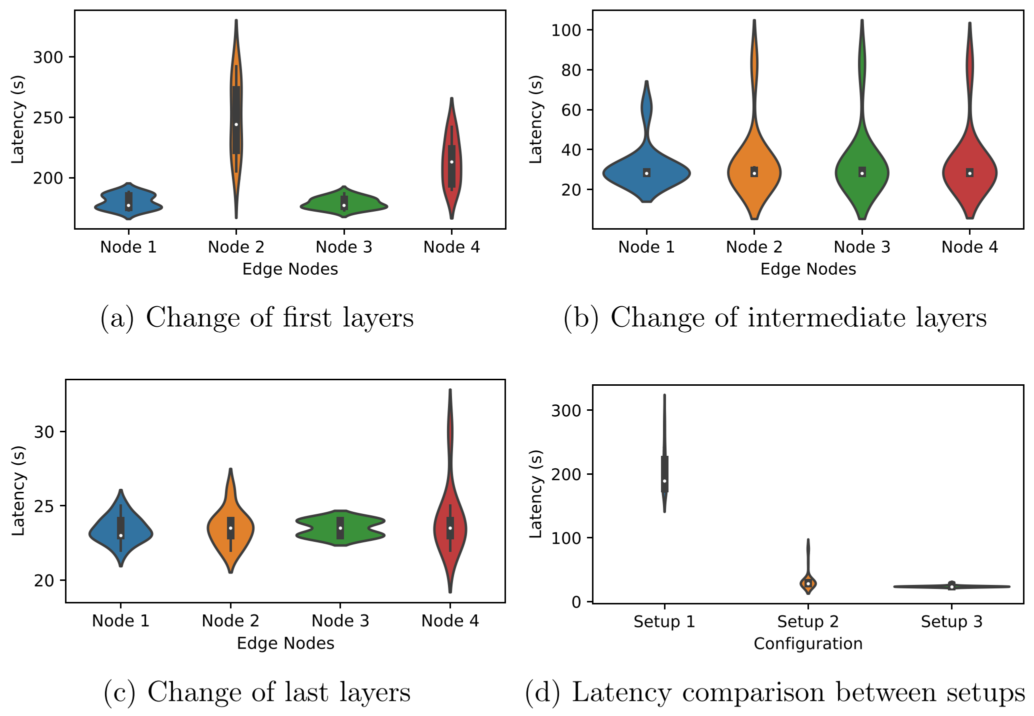

Fig. 7. Latency of Service Deployment Results.

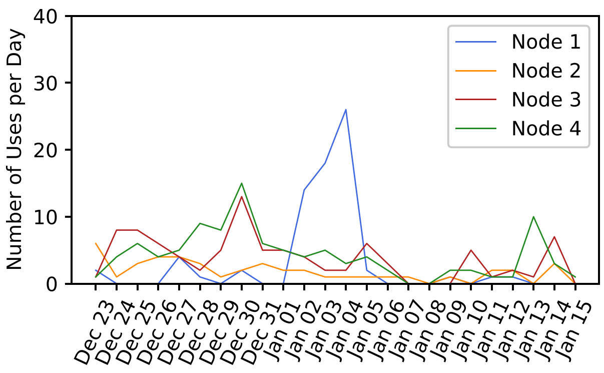

Fig. 8. Sessions opened on four different nodes during 24 days.

this work, a large-scale network has been designed, developed and deployed throughout a region, providing an end-to-end solution for service orchestration and centralized management by means of a microservices architecture.

The main objective of the project was to provide a scalable, resilient and agile system able to be managed remotely. The cloud platform, openBalena, and the deployed edge nodes, with balenaOS, provide service orchestration aimed at enhancing tourism in the region of Badajoz, Spain. The seamless microservices architecture enables multi-tenant operation, adjusting the content to the different places included in the network. The deployment of new services on the installed terminals is performed autonomously, since the nodes continuously compare the information with the cloud platform and the corresponding containers are downloaded and installed according to the configuration applied.

Lessons learned and practical insights for future contributions include: the microservices architecture deployed provides high scalability, since the addition of new edge nodes can be performed using the same base preconfigured image used in the rest of the nodes. However, the added complexity when dealing with interactive applications requires deep knowledge about the underlying operation of the OS and virtualization involved. Regarding the openBalena integration, although the Balena's community is not very large, it is very active, and the team is usually diligent in resolving queries.

As part of the future work, we want to increase the processing carried out by the edge nodes through dynamically assigning tasks to nodes depending on the available resources, location and workload of each node by means of FogFlow. Furthermore, the fusion of containers' orchestration and FogFlow can remove the constraint related to the burden of sharing the same hardware architecture, allowing the use of a wider range of devices within the same application depending on the computational needs, achieving a multi-architecture system.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Financial disclosure

Grant DIN2018-010177 funded by MCIN/AEI/10.13039/501100011033. Grant PID2021123627OB-C52 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way to make Europe". Grant 2022-GRIN-34056 funded by Universidad de Castilla-La Mancha, Spain. Grant 2019-PREDUCLM-10703 funded by Universidad de Castilla-La Mancha, Spain and by "ESF Investing in your future". We thank Diputación de Badajoz for the commitment and collaboration.

Data availability

Data will be made available on request

References

[1] L. Nkenyereye, J. Hwang, Q.V. Pham, J. Song, Virtual IoT service slice functions for multiaccess edge computing platform, IEEE Internet Things J. 8 (14) (2021) 11233–11248, http://dx.doi.org/10.1109/JIOT.2021.3051652.
[2] P.K. Thiruvasagam, A. Chakraborty, C.S.R. Murthy, Resilient and latency-aware orchestration of network slices using multi-connectivity in MEC-enabled 5G networks, IEEE Trans. Netw. Serv. Manag. 18 (3) (2021) 2502–2514, http://dx.doi.org/10.1109/TNSM.2021.3091053.
[3] V. Sciancalepore, F. Cirillo, X. Costa-Perez, Slice as a service (SlaaS) optimal IoT Slice resources orchestration, in: Proceedings - IEEE Global Communications Conference, GLOBECOM 2017, 2018-Janua, Institute of Electrical and Electronics Engineers Inc., 2017, pp. 1–7, http://dx.doi.org/10.1109/GLOCOM.2017.8254529.
[4] D. Hastbacka, J. Halme, L. Barna, H. Hoikka, H. Pettinen, M. Larranaga, M. Bjorkbom, H. Mesia, A. Jaatinen, M. Elo, Dynamic edge and cloud service integration for industrial IoT and production monitoring applications of industrial cyber-physical systems, IEEE Trans. Ind. Inform. 18 (1) (2022) 498–508, http://dx.doi.org/10.1109/TII.2021.3071509.
[5] W. Rafique, X. Zhao, S. Yu, I. Yaqoob, M. Imran, W. Dou, An application development framework for internet-of-things service orchestration, IEEE Internet Things J. 7 (5) (2020) 4543–4556, http://dx.doi.org/10.1109/JIOT.2020.2971013.
[6] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, A. Kitazawa, FogFlow: Easy programming of IoT services over cloud and edges for smart cities, IEEE Internet Things J. 5 (2) (2018) 696–707, http://dx.doi.org/10.1109/JIOT.2017.2747214.
[7] F. Cirillo, F.J. Wu, G. Solmaz, E. Kovacs, Embracing the future internet of things, Sensors 19 (2) (2019) 351, http://dx.doi.org/10.3390/S19020351.
[8] A. Dalgkitsis, P.V. Mekikis, A. Antonopoulos, C. Verikoukis, Data driven service orchestration for vehicular networks, IEEE Trans. Intell. Transp. Syst. 22 (7) (2021) 4100–4109, http://dx.doi.org/10.1109/TITS.2020.3011264.
[9] L. Zanzi, F. Cirillo, V. Sciancalepore, F. Giust, X. Costa-Perez, S. Mangiante, G. Klas, Evolving multi-access edge computing to support enhanced IoT deployments, IEEE Commun. Stand. Mag. 3 (2) (2019) 26–34, http://dx.doi.org/10.1109/MCOMSTD.2019.1800009.
[10] I.H. Sarker, Smart City Data Science: Towards data-driven smart cities with open research issues, Int. Things (Netherlands) 19 (2022) 100528, http://dx.doi.org/10.1016/j.iot.2022.100528.
[11] R. Morabito, V. Cozzolino, A.Y. Ding, N. Beijar, J. Ott, Consolidate IoT edge computing with lightweight virtualization, IEEE Netw. 32 (1) (2018) 102–111, http://dx.doi.org/10.1109/MNET.2018.1700175.
[12] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, O. Rana, The internet of things, fog and cloud continuum: Integration and challenges, Int. Things (Netherlands) 3–4 (2018) 134–155, http://dx.doi.org/10.1016/j.iot.2018.09.005, arXiv:1809.09972.
[13] X. Li, C. Chiasserini, J. Mangues-Bafalluy, J. Baranda, G. Landi, B. Martini, X. Costa-Perez, C. Puligheddu, L. Valcarenghi, Automated service provisioning and hierarchical SLA management in 5G systems, IEEE Trans. Netw. Serv. Manag. (2021) http://dx.doi.org/10.1109/TNSM.2021.3102890.
[14] G. Solmaz, F.J. Wu, F. Cirillo, E. Kovacs, J.R. Santana, L. Sanchez, P. Sotres, L. Munoz, Toward understanding crowd mobility in smart cities through the internet of things, IEEE Commun. Mag. 57 (4) (2019) 40–46, http://dx.doi.org/10.1109/MCOM.2019.1800611.
[15] D. Bruneo, S. Distefano, M. Giacobbe, A. Longo Minnolo, F. Longo, G. Merlino, D. Mulfari, A. Panarello, G. Patanè, A. Puliafito, C. Puliafito, N. Tapas, An IoT service ecosystem for Smart Cities: The #SmartME project, Int. Things (Netherlands) 5 (2019) 12–33, http://dx.doi.org/10.1016/j.iot.2018.11.004.
[16] F. Cirillo, D. Gómez, L. Diez, I. Elicegui Maestro, T.B.J. Gilbert, R. Akhavan, Smart city IoT services creation through large-scale collaboration, IEEE Internet Things J. 7 (6) (2020) 5267–5275, http://dx.doi.org/10.1109/JIOT.2020.2978770.
[17] F. Cirillo, D. Straeten, D. Gomez, J. Gato, L. Diez, I.E. Maestro, R. Akhavan, Atomic Services: Sustainable ecosystem of smart city services through pan-European collaboration, in: Proceedings - Global IoT Summit, GIoTS 2019, Institute of Electrical and Electronics Engineers Inc., 2019, http://dx.doi.org/10.1109/GIOTS.2019.8766431.
[18] G. Ortiz, J. Boubeta-Puig, J. Criado, D. Corral-Plaza, A. Garcia-de Prado, I. Medina-Bulo, L. Iribarne, A microservice architecture for real-time IoT data processing: A reusable web of things approach for smart ports, Comput. Stand. Interfaces 81 (2022) 103604.
[19] M. Krämer, S. Frese, A. Kuijper, Implementing secure applications in smart city clouds using microservices, Future Gener. Comput. Syst. 99 (2019) 308–320.
[20] B. Cen, C. Hu, Z. Cai, Z. Wu, Y. Zhang, J. Liu, Z. Su, A configuration method of computing resources for microservice-based edge computing apparatus in smart distribution transformer area, Int. J. Electr. Power Energy Syst. 138 (2022) 107935.
[21] G. Blinowski, A. Ojdowska, A. Przybyłek, Monolithic vs. microservice architecture: A performance and scalability evaluation, IEEE Access 10 (2022) 20357–20374.
[22] A. Das, S. Chakraborty, S. Chakraborty, Where do all my smart home data go? Context-aware data generation and forwarding for edge-based microservices over shared IoT infrastructure, Future Gener. Comput. Syst. 134 (2022) 204–218.
[23] A. Christoforou, A.S. Andreou, M. Garriga, L. Baresi, Adopting microservice architecture: A decision support model based on genetically evolved multi-layer FCM, Appl. Soft Comput. 114 (2022) 108066.
[24] Z. Li, C. Shang, J. Wu, Y. Li, Microservice extraction based on knowledge graph from monolithic applications, Inf. Softw. Technol. 150 (2022) 106992.

[25] C. Wu, Q. Peng, Y. Xia, Y. Jin, Z. Hu, Towards cost-effective and robust AI microservice deployment in edge computing environments, Future Gener. Comput. Syst. 141 (2023) 129–142.

[26] P. Valderas, V. Torres, E. Serral, Modelling and executing IoT-enhanced business processes through BPMN and microservices, J. Syst. Softw. 184 (2022) 111139.

[27] M.M. Salim, S.K. Singh, J.H. Park, Securing Smart Cities using LSTM algorithm and lightweight containers against botnet attacks, Appl. Soft Comput. 113 (2021) 107859, http://dx.doi.org/10.1016/J.ASOC.2021.107859.

[28] A. Wenz, T.A. Johansen, Real-time moving horizon estimation of air data parameters and wind velocities for fixed-wing UAVs, in: 2020 International Conference on Unmanned Aircraft Systems, ICUAS 2020, Institute of Electrical and Electronics Engineers Inc., 2020, pp. 998–1006, http://dx.doi.org/10.1109/ICUAS48674.2020.9214009.

[29] R. Botez, V. Strautiu, I.A. Ivanciu, V. Dobrota, Containerized application for IoT devices: Comparison between balenacloud and amazon web services approaches, in: Proceedings - 14th International Symposium on Electronics and Telecommunications, ISETC 2020, Institute of Electrical and Electronics Engineers Inc., 2020, http://dx.doi.org/10.1109/ISETC50328.2020.9301070.

[30] K.H. Le Minh, K.H. Le, Q. Le-Trung, DLASE: A light-weight framework supporting deep learning for edge devices, in: Proceedings - 4th International Conference on Recent Advances in Signal Processing, Telecommunications and Computing, SigTelCom 2020, Institute of Electrical and Electronics Engineers Inc., 2020, pp. 103–108, http://dx.doi.org/10.1109/SIGTELCOM49868.2020.9199058.

[31] R. Dautov, H. Song, Towards agile management of containerised software at the edge, in: Proceedings - IEEE Conference on Industrial Cyberphysical Systems, ICPS 2020, Institute of Electrical and Electronics Engineers Inc., 2020, pp. 263–268, http://dx.doi.org/10.1109/ICPS48405.2020.9274764.

[32] D. Lennick, A. Azim, R. Liscano, A microservice-based architecture for performance and energy benchmarking of docker-host linux distributions on internet-of-things devices, in: Proceedings - IEEE International Conference on Industrial Technology, Vol. 2021-March, Institute of Electrical and Electronics Engineers Inc., pp. 705–711, http://dx.doi.org/10.1109/ICIT46573.2021.9453517.