Edge computing in IoT context: horizontal and vertical Linux container migration

Corentin Dupont, Raffaele Giaffreda, Luca Capra
FBK Create-Net, via alla Cascata 56/D, 38123 Trento, Italy.
<cdupont,rgiaffreda,lcapra>@fbk.eu

***Abstract*—In recent years, the development of the Internet of Things (IoT) has largely been supported by the parallel development of the Cloud computing capacities. Cloud computing offered elastic and on-demand computing power able to host and support crucial IoT functions. However, the proliferation of connected objects poses the problem of the confidentiality of the data produced. This proliferation also threatens the performances of the supporting networks. To tackle those problems, the research community has recently turned to Fog Computing. In this paper, we present Cloud4IoT, a platform able to perform horizontal (roaming) and vertical (offloading) migration of IoT functions. We implement our demonstration using a Kubernetes cluster organised in three tiers: Cloud, Edge and IoT gateways. Our first use cases shows how IoT function roaming can be used in the context of health care data exploitation. The second use case exploits the IoT offloading capacity of our platform to optimize the remote diagnostic of mechanical engines.**

***Index Terms*—PaaS, Orchestrator, Internet of Things, OpenStack, Fog Computing, Edge Cloud, Docker, Kubernetes, Linux container**

I. Introduction

According to a recent survey by Gartner[1], the total number of connected objects in 2020 will be 20.8 billion. This proliferation of objects and associated data streams has put significant stress on the network and edge infrastructures. Furthermore, the locality and security of the data produced are becoming increasingly challenging. The uptake of data-intensive IoT applications is also apparent in several sectors such as industrial manufacturing, energy provisioning, transportation, healthcare and mining. Similar challenges appear in remote and distributed contexts, such as rural, offshore and logistics. Such contexts have intermittent internet connections, and low bit rates.

In a traditional IoT setting, the data is sent from the IoT device to the Cloud, where it is processed and stored. However, in recent years the development of Fog computing [?], [?] brought new kind of solutions to these problems. Fog Computing allows to bridge between the IoT and Cloud domains [?]. Furthermore, data sent from personal IoT devices to the Cloud need to be regarded as private. In this case, the network edges need to be personalized with applications taking care of the privacy of the data being sent to the Cloud.

We introduce the concept of *IoT functions* as the virtualisation of specific IoT capabilities, in order to deploy and migrate them easily. Some examples of IoT functions include data stream analysis, predictions and recommendations based on sensor data and data anonymisation.

In this paper we present Cloud4IoT, a platform able to containerize IoT functions and optimize their placement. The three main features of Cloud4IoT are:

- deployment of containerized IoT functions directly on the IoT gateway, according to a specification manifest,
- horizontal migration (roaming): migration of IoT functions from one gateway to another,
- vertical migration (offloading): migration of IoT functions from the Cloud to a gateway (or opposite).

Hardware resources, like computing power, memory and storage are virtualized on the whole computing infrastructure (the Cloud, the edge and the gateway). Cloud4IoT leverages the containerization technology in order to attain automatic deployment, dynamic configuration and orchestration of IoT functions. Cloud4IoT is able to support a cluster of IoT gateways. The platform uses the state of the art Open Source frameworks Docker[2] and Kubernetes[3].

We discuss two use cases:

- IoT roaming in health care context
- IoT offloading for remote engine diagnostic

In the first use case, we demonstrate how to dynamically specialize gateways to a specific domain such as eHealth. This is done by dynamically deploying eHealth data processing containers directly to the gateway were the data is collected. The second use case concerns a remote diagnostic application for industrial/mechanical engines. In this context, we will show that running the application close to the data source on the gateway or back in the Cloud has several trade-offs.

The rest of this paper is organized as follows: an overview of Cloud4IoT is presented in Section II, we detail the implementation in Section III, and the use cases are presented in Section IV. We present the related works in Section V and finally the future works and conclusion are in Section VI.

II. Overview

Cloud4IoT offers two main features: *IoT function migration* and *IoT function scaling*. It supports the automatic migration of IoT functions from one IoT Gateways to another. The second feature is the migration of IoT function containers from the gateway to the edge layer and finaly to the Cloud.

[1]http://www.gartner.com/newsroom/id/3165317

[2]https://www.docker.com/

[3]https://kubernetes.io/

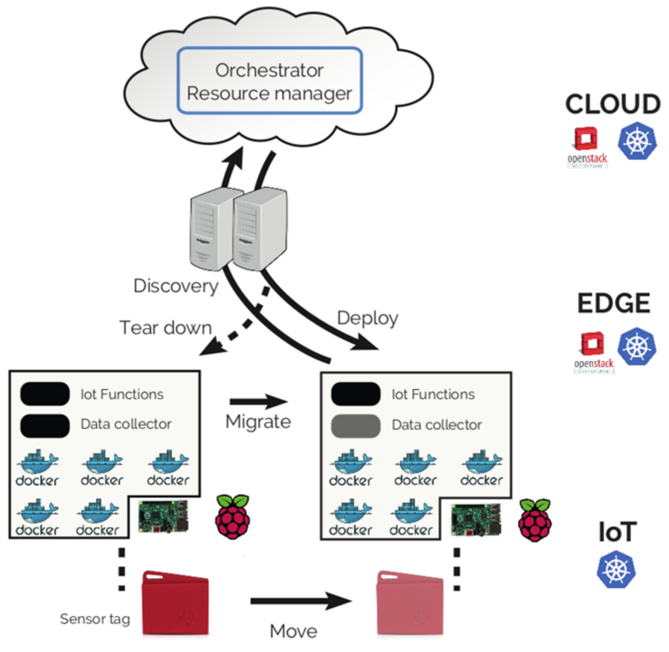

Fig. 1. IoT roaming.

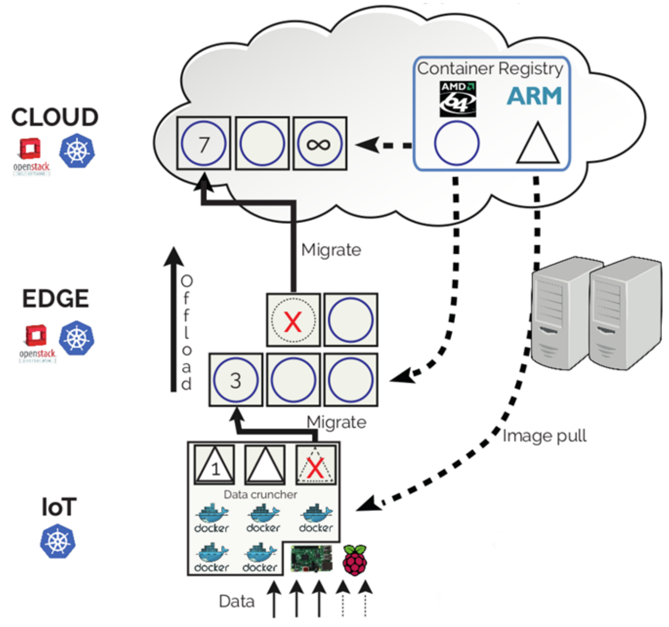

Fig. 2. IoT offloading.

Such containers can be initially deployed near the data source, scaled at the need within the edge layer, and eventually re-deployed onto the central cloud.

A. IoT function migration: device discovery and automatic service migration

The Figure 1 shows the roaming feature in action. The roaming is enabled by migration of Docker containers from one gateway to another. When the device is moved, the discovery from the second gateway will detect it, and send a signal to the Cloud orchestrator. The Cloud orchestrator will then change the *Node affinity*[4] of the data processing container, and trigger a re-deployment. This will force the container on the first gateway to be destroyed, while another is created on the second gateway.

The migration strategy simply consists in destroying and re-creating them on the target node. This is possible because they are stateless. Secondly, their creation time is very short.

B. IoT offloading

The Figure 2 shows the offloading of IoT functions. The Gateways being resource constrained devices, they are constantly monitored by the Cloud orchestrator. The Cloud orchestrator is then in charge of optimizing the placement of the containerized IoT functions. Placing IoT functions that performs routine data consolidation on the Gateway results in less data being transmitted to the Cloud, and thus in less network traffic. However, only a certain number of theses functions can fit on the Gateway, since the CPU and RAM are limited. For example, if too much users connect to the same gateway, it will become necessary to instantiate further containers on the edge or in the Cloud, rather than on the gateway itself.

[4]https://kubernetes.io/docs/user-guide/node-selection/

III. Implementation

We implemented our architecture using OpenStack as a VM manager, and Kubernetes as a container manager. The OpenStack environment is configured with 3 Controllers nodes and 2 Compute nodes on dedicated servers. Kubernetes is installed inside the 2 Compute nodes. The Kubernetes cluster includes the Cloud nodes and the Edge nodes, and as been extended to also include two IoT Gateways as nodes.

The IoT Gateway consists of a Raspberry PI version 3 and an extension shield able to host various wireless communication modules. We deployed on the gateways an ARM version of docker, a service able to run containerised applications. We also compiled specific versions of the IoT function containers for ARM and i386. The compiled images are stored on a container registry, with the tag corresponding to the target architecture. Using Kubernetes labels, the central orchestrator is then able to force Kubernetes to deploy the correct container.

The discovery container takes care of initiating the communication over the Bluetooth Low Energy[5] (BLE) hardware device available on the gateway. Initially the BLE receiver is turned on and the discovery started. The reliability and privacy level of the discovery is optimized via a configurable threshold on initialization. The limit is calculated from the RSSI (Received signal strength indication) value as received from the device chip. When a device is found an event notification is propagated to the orchestrator which take care of initiating the next steps.

[5]https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy

The data reader container uses the contextual information from the orchestrator to create a data stream from the GATT (Generic Attribute Profile) services exposed by the device. In order to get the data from the device, first a connection via the device MAC is created. Once connected, in order to optimize battery consumption, the data reader software enable the required subscriptions service to receive updated data. When the connection get disconnected device from the orchestration (or due the device getting out of range) the reader attempt to reset the state on the device and clean up its state disabling the subscription.

In sum, the gateway follows those steps when discovering a user close to it:

- the BLE discovers the sensor tag,
- the orchestrator is notified,
- the Orchestrator deploys the container,
- the container image is pulled from the Cloud (if not already present on the Gateway),
- the container is created/attached/connected,
- the container starts,
- the first data point is received in the application container.

Bluetooth discovery is kept active to immediately react to a device detection. This setup impact negatively on the energy consumption, which can be reduced optimizing how long the discovery is active [?].

Regarding the destruction of the containers, the steps are the following:

- the BLE detects the loss of connection,
- the ochestrator is notified,
- the ochestrator undeploys the container,
- the container is destroyed.

IV. Use Cases

The previous section highlighted two types of migration supported by our architecture. In this section, we illustrate how such flexibility is being used in two different application domains.

A. *IoT roaming for health care*

In this use case, a doctor needs to receive notifications from a smart bracelet worn by his patients. Once the patient leaves his/her home and reaches the hospital, the wearable device is automatically associated with a new gateway and monitoring service is configured accordingly.

The wearables produce data using the Open mHealth standard[6]. Our first implementation transmitted all the messages to the Cloud backend for the appropriate processing. However, this created unnecessary load on the infrastructure and a high latency. To avoid this problem we created a docker image able to parse Open mHealth data and consolidate them before sending to the Cloud. This container image was instantiated on the corresponding Gateway.

Using this mechanism, we were able to specialise an otherwise generic gateway to a specific domain (eHealth in this case). The roaming capacity of Cloud4IoT then allows to migrate the Open mHealth data processing capacity to a new gateway if necessary.

[6]http://www.openmhealth.org/

B. *IoT offloading for remote engine diagnostic*

Similar needs have been encountered in an Industrial IoT use-case where IoT offloading has been applied. As part of their maintenance plan, manufacturing companies deploy continuous and remote monitoring of mechanical engines. The data is sent continuously to diagnostic platforms. Such diagnostic platforms run algorithms able to process data samples extracted from dedicated sensors mounted on the products.

In this context, migrating the diagnostic component from the Cloud to a gateway, or offloading it back to the Cloud is extremely useful. It enables different monitoring modes according to the expected sampling frequency and overall situation. For example, product test and deployment phases can use a more intensive monitoring close to the data source, whereas normal behaviour of the product can lead to relaxing such requirements and free-up resources on the gateway. Since the Gateway has limited resources, it is possible to run different diagnostic algorithms at different point in time, keeping only the necessary container while cleaning the others.

V. Related works

A survey of the literature shows that the research community only recently began to explore Edge Computing in the context of Internet of Things.

In [?], Bonomi et al. explained how the characteristics of Fog computing are crucial for the Internet of Things. In particular, the low latency, the location awareness and the widespread geographical distribution of nodes are key characteristics of Fog computing. In [?], the authors are proposing a design for a Cloud system that features containers within smart objects. They use a message passing middleware for communication.

The virtualization techniques for Edge computing are studied in [?]. The authors confirm that containers have several key benefits for Edge computing over traditional virtual machines, in terms of size and flexibility. Furthermore, they show that cluster management tools such as Kubernetes or Mesos can manage and orchestrate applications that extends to the edge.

In [?], [?], docker containers are evaluated positively for usage in Edge computing contexts. For instance, the author of [?] have compared the performance of native applications versus containerized application, on a Raspberry PI. They show that the overhead of containers in term of performance is negligible.

This work is an extension of the paper [?]. With respect to this previous work, the aim of this paper is to show the IoT perpective of PaaS4IoT. We added the implementation, the IoT use cases, the related works review and the future works.

VI. Conclusion & Future works

The current challenges of IoT are clearly pushing the researchers toward Fog computing solutions. In this paper, we presented the architecture and implementation of a Fog Computing platform for IoT. We showed how a container could be migrated both horizontally (from one gateway to another) and vertically (from Cloud to the Edge and to the Gateway). Our demonstration included a central Cloud, an Edge and two raspberry PI Gateways. We discussed two use cases: the first showing IoT roaming in health care context and the second showing IoT offloading for remote engine diagnostic.

The flexibility and elasticity of Cloud services and the concept of Infrastructure as Code [**?**] bears a great potential for solving many of the problems currently hindering IoT from becoming a mainstream technology. Many different objects using many different technologies are used to harvest data in different contexts, contributing to the current fragmentation of IoT. Our future work will leverage on the Infrastructure as Code deployment techniques being available on edge devices such as Raspberry PIs. The goal being to increase flexibility with limited memory and computing footprint. We will therefore explore the usage of containers for dynamically loading Hardware Abstraction Layers, in order to handle diversity on a per-need basis. These Hardware Abstraction Layers enables temporary associations with diverse sets of connected objects at low level. Going further up the stack, we will use this technology to propose solutions for the problem of the many standards and associated protocols used in different application domains. Here the flexibility in our Infrastructure as Code will be used to dynamically (and temporarily) load the right libraries for understanding different protocols and interact with different IoT Platforms. This will enable future IoT gateways to become easily tailored and reconfigurable to meet the needs of the user.

Achieving results in these directions potentially exposes the infrastructure to security risks but it also weakens the association between users and devices, which will become temporary means to acquire data. From a security standpoint therefore we will aim to secure the mechanisms through which code can be injected and run in containers. Breaking the tie between devices and users will considerably reduce the risk associated with unauthorised access to owned data through misconfigured devices. This particular aspect and the related creation of personal data vaults is currently under investigation for e-Health and assisted living application scenarios.

Acknowledgment

This research received funding from the European Union's H2020 Research and Innovation Action under grant agreement n. 688088 (project AGILE) and grant agreement n. 687607 (project WAZIUP).

References

[1] Cisco, "Transformation to a next generation IoT service provider," CISCO, White Paper, 2016.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. of MCC*. New York, NY, USA: ACM, 2012, pp. 13–16.

[3] M. Yannuzzi, R. Milito, R. Serral-Graci, D. Montero, and M. Nemirovsky, "Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing," in *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Dec 2014, pp. 325–329.

[4] R. Woodings, D. D. Joos, T. Clifton, and C. D. Knutson, "Rapid heterogeneous ad hoc connection establishment: accelerating bluetooth inquiry using irda." in *WCNC*. IEEE, 2002, pp. 342–349.

[5] D. Mulfari, M. Fazio, A. Celesti, M. Villari, and A. Puliafito, *Design of an IoT Cloud System for Container Virtualization on Smart Objects*. Cham: Springer International Publishing, 2016, pp. 33–47. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-33313-7_3

[6] 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.

[7] R. Morabito, "A performance evaluation of container technologies on internet of things devices," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2016, pp. 999–1000.

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

[9] D. Pizzolli, G. Cossu, D. Santoro, L. Capra, C. Doukas, C. Dupont, F. De Pellegrini, F. Antonelli, and S. Cretti, "Cloud4iot: a heterogeneous, distributed and autonomic cloud platform for the iot," in *IEEE CloudCom conference proceedings*, 2016.

[10] K. Morris, *Infrastructure as Code*. O'Reilly Media, Inc., 2015.