A Demo of Application Lifecycle Management for IoT Collaborative Neighborhood in the Fog

Practical Experiments and Lessons Learned around Docker

Loïc Letondeur
Orange Labs, France
loic.letondeur@orange.com

François-Gaël Ottogalli
Orange Labs, France
francoisgael.ottogalli@orange.com

Thierry Coupaye
Orange Labs, France
thierry.coupaye@orange.com

Abstract— **Regarding latency, privacy, resiliency and network scarcity management, only distributed approaches such as proposed by Fog Computing architecture can efficiently address the fantastic growth of the Internet of Things (IoT). IoT applications could be deployed and run hierarchically at different levels in an infrastructure ranging from centralized datacenters to the connected things themselves. Consequently, software entities composing IoT applications could be executed in many different configurations. The heterogeneity of the equipment and devices of the target infrastructure opens opportunities in the placement of the software entities, taking into account their requirements in terms of hardware, cyber-physical interactions and software dependencies. Once the most appropriate place has been found, software entities have to be deployed and run. Container-based virtualization has been considered to overpass the complexity of packaging, deploying and running software entities in a heterogeneous distributed infrastructure at the vicinity of the connected devices. This paper reports a practical experiment presented as a live demo that showcases a "Smart Bell in a Collaborative Neighborhood" IoT application in the Fog. Application Lifecycle Management (ALM) has been put in place based on Docker technologies to deploy and run micro-services in the context of Smart Homes operated by Orange.**

Keywords—Fog Computing; IoT; Application Lifecycle management; Containers; Docker; Micro-services; Orchestration; Deployment; Heterogeneity

I. Introduction

After a decade of continuous growth of Cloud Computing, fueled by mega datacenters which concentrate computing and storage resources for on-demand enterprise and web applications, we can now witness the emergence of more distributed paradigms, such as Fog Computing. Fog Computing [1, 2] is typically motivated by Internet of Things (IoT) applications (Smart Home, Smart City, Smart Agriculture, Smart Car, Smart Grid, etc.) for which it appears more adequate to distribute computing, storage, interaction and control at the edge of the network and beyond, closer to things of the physical world – where sensors and actuators are – rather than only in remote mega datacenters. This is especially appealing for applications that require, for example, low latencies, data privacy enforcement, or the control over the amount of data commuting by the core network.

Fog Computing brings new challenges regarding Application Lifecycle Management (ALM) [8, 12, 13]. ALM covers all the stages from development to deployment and reconfiguration of an application. Fundamentally, one core issue of ALM in the Fog is how to place a set of software entities defining a distributed application onto a Fog infrastructure made of Fog nodes providing resources for the execution (i.e. computing, storage, communication and interaction). Major challenges of ALM in Fog environment (compared to Cloud environment) are the following: *scale* (we might expect the number of Fog nodes to be one or several orders of magnitude higher compared to Cloud); *heterogeneity* (Fog nodes exhibit a much higher variability than Cloud nodes in terms of computing, storage, network connectivity, energy consumption and electricity supply); *volatility* (Fog nodes are not always connected or available due to, for example, energy or connectivity shortage, or because they are moving from one place to another); *locality* (Fog nodes are sensible to geographical localization as they tend to be placed at the vicinity of the sensors and actuators hosted by the IoT devices). At the end of the day, the way ALM could deal with concerns such as *elasticity* regarding Cloud Computing, can just not be reused as they are in Fog Computing. New models, algorithms, optimization methods and technologies must be developed [7, 9, 10].

In order to get practical feedback on these issues, we have built and made experiments on an industrial testbed for Fog Computing and IoT that is deployed in several sites within Orange Labs (i.e. infrastructures multiple sorts of Fog nodes along with different set of sensors and actuators). This paper reports some experiments that were presented as a live demo during the Orange Research Exhibition in Paris, France in December 2016. The demo showcases a "Smart Bell" IoT application in a "Collaborative Neighborhood" context as an extension of a Smart Home, that could be representative of home services proposed and sold by Orange in a near future. Section II introduces the "Smart Bell" demo itself and its *user story*. Section III describes the physical infrastructure of the demo, i.e. the various Fog nodes and things deployed inside the different houses. Section IV describes the software functional architecture of the IoT "Smart Bell" distributed application. Section V details the setup of the demo, i.e. how the application was actually deployed onto the Fog infrastructure. Section VI provides feedback and lessons learned from the realization of the demo. Section VII concludes by proposing some requirements on ALM for Fog that emerged from our experiments. Note that since Docker, a virtualization orchestration solution based on Linux containers, was somehow the cornerstone of our ALM experiments, sections V to VII deal quite a lot with the usage of Docker that was made and practical feedback on this usage.

II. Context & purpose of the demo

Connected things/objects, such as *Netatmo* [20] weather stations, get people used to share data from their devices. Not only the data, but also the devices themselves could be shared to build high value personalized IoT applications as mashups of objects and services [14, 15]. For example, in the context of a neighborhood, people could be willing to collaborate to create a "Smart Bell" application. The application would tell the landowner, wherever s.he could be, when someone is coming to ring the doorbell at her.is usual home, especially when s.he is away in her.is secondary house (see Fig. 1, *Home #1*) for the week end. Sensors in the hallway will trigger an event when someone is approaching the door of the main house (see Fig. 1, *Home #3*). A connected light will then be turned on in white nearby the landowner, in her.is secondary house, to announce that someone is coming. A face recognition service is engaged, using the video stream of the camera in the hallway, to try to identify the person approaching the door. If s.he is recognized, the light is turned to green, if not the light is turned to red. At the time when the doorbell is pushed, if the person has been properly recognized, a welcome message is streamed to the speaker in the entrance and the connected lock could be open if s.he is awaited. The light is then turned to blue for a couple of seconds to say that someone is entering the main house. If the person is not recognized, the application will send a command to close the lock, to enforce the closure of the lock, and a message could be sent (possibly with the picture of the unrecognized visitor) at the neighborhood level to tell the neighbors to be vigilant on what is ongoing. Outdoor video cameras could collaborate to follow the unidentified visitor up to the point when s.he will be recognized by someone, or s.he will leave the neighborhood. In both cases, a new message would be sent to announce that the situation is safe again.

To create such an IoT "Smart Bell" application, a set of sensors and actuators have to be "mashuped" with services. The infrastructure in the vicinity of the objects is leverage to run the software entities engaged in the mashups.

III. Physical Infrastructure for the demo

The infrastructure of the demo is made of a set of connected devices. Some of them can be used as a "Fog node", i.e. they can dynamically host and run on demand software entities. Fig. 1 depicts the infrastructure testbed which is composed of 5 *Raspberry Pi 3*, 2 *Arduino*, 3 *Livebox 4* (Orange home internet gateway, dual-core 32bits ARM CPU) and 1 *Ruggedpod* [24] as a neighborhood micro-datacenter (the RuggedPod is basically a "data center in a box": it is a water-proof passively cooled hardened server with four 8 cores *Xeon* CPUs that can be placed on a roof top or any collective place in a neighborhood).

These Fog nodes reproduce typical home environments of Orange customers. For the demo, three homes are involved. Each of them is connected to Internet with an Orange Livebox 4. All the homes have one or more Raspberry Pi 3 connected to the home gateway through WiFi. Each home has its own specificities regarding connected things. In *Home#1*, *Philipps Hue* connected lights are present in addition to one *Awox StreamLight* [25], a WiFi connected light able to play sounds. In *Home#2*, a WiFi motorized camera is connected to the Livebox 4. A thermometer (used in another part of the application/demo not described in this paper) is also linked to a Raspberry Pi 3 through Bluetooth thanks to an Arduino. In *Home#3*, entrance door has a connected lock with APIs exposed via Internet. Bluetooth connected things are also present in *Home#3* such as a connected doorbell button, a thermometer and a connected speaker. To finish with *Home#3*, a WiFi security camera is connected to the home internet gateway (i.e. the Livebox 4). At "the level above", that is the level of the neighborhood, a collective and more powerful fog node (the RuggedPod micro-datacenter) is in use. Such a micro-datacenter could be placed in a collective area of the neighborhood: typically near to actual mobile phone antennas or in actual telecom operator Points of Presence (PoP) for fixed internet network access.

This hierarchical Fog infrastructure has been used to demo deployments as described in next section.

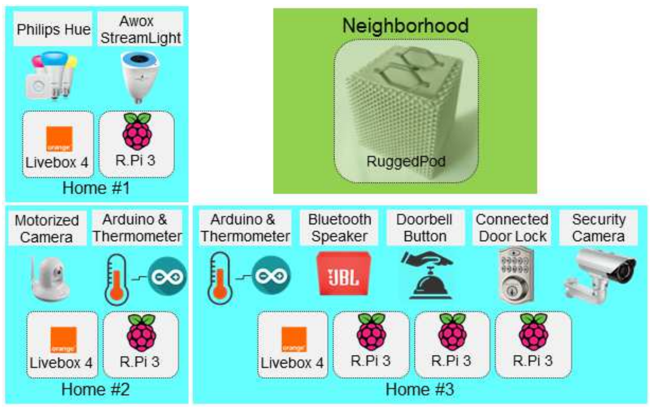

Fig. 1 Infrastructure of the demo

IV. APLLICATIVE FUNCTIONAL ARCHITECTURE

The "Smart Bell" application is based on a set of software entities which are hosted and run onto a set of Fog nodes. Fig. 2 depicts the functional architecture of the demo. These software entities make use of different technologies and have both configuration and functional dependencies among them. The software entities of the "Smart Bell" application are:

- A Message Oriented Middleware that permits asynchronous communications and eventing among the software entities. It is made of an MQTT broker based on ActiveMQ with dependencies to Java Runtime Environment (JRE).

- A Complex Event Processor (CEP) that operates as an IoT event hub to aggregate, filter and trigger IoT events according to business rules. Our CEP is an extension of *Esper* [18] which requires JRE version 8 and depends on an MQTT broker.

- A Mashup Engine called *Cocktail*, based on SNAP [16], which permits the execution of a graph of actions on actuators and services accessible via APIs, based on contexts made of events from sensors and services. Using Python, Cocktail has dependencies to an MQTT broker, a media center and IP-capable connected things. Its configuration requires external versioned scripts.

- An IoT Capillary Router called *Sensonet* [21] that enlists heterogeneous connected things, collects data and pushes them to the CEP. Sensonet is divided into two different C-written software entities: Sensonet core and Sensonet connectors. Sensonet connectors are dependent of Sensonet core and must be run on a Fog node which satisfies their hardware requirement (e.g. Bluetooth Low Energy, Arduino physical connection). Sensonet core is dependent to an MQTT broker.

- A Face Recognition enabler based on OpenCV. This software entity requires a huge amount of computation resources only available from the neighborhood micro-datacenter.

To cope with production grade requirements, each software

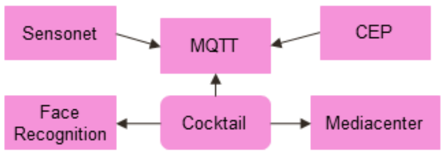

Fig. 2 Functional architecture of the Smart bell application

entity is packaged into a Docker image and each instance is an individual Linux container. Fig. 3 depicts the deployment of the architecture. Placements have been determined according to locations and resources capabilities. Concerning locations, Cocktail runs mashups which are built for a dedicated home: for privacy considerations, they are run inside Fog nodes in their respective homes. On the contrary, face recognition software entity is deployed on the micro-datacenter of the neighborhood as it is the only place where computation requirements could be satisfied.

This experiment highlights technical issues regarding ALM in Fog Computing: (1) *Software & hardware heterogeneity* (2) *Software dependencies* and (3) *Software (Re)Configuration.*

(1) Software & hardware heterogeneity. From hardware point-of-view, devices mix CPUs of different architectures and manufacturers. Additionally, each node has its own connectivity specificities with both IP and non-IP capabilities for instance. On software side, deployed software entities are very heterogeneous as they mix many implementation languages

(2) Software dependencies. Each software entity can have dependencies to software artifacts and/or libraries that must be satisfied at runtime. Some software instances have even hardware dependencies as in the case of the Media center which depends to Bluetooth to connect the speaker in *Home#3.*

(3) Software (Re)Configuration (elasticity). Software architecture has to evolve due to changes in software (e.g. varying applicative load, new features) and hardware (failures, equipment that have moved, are disconnected or are not available for any reason). Consequently, many parameters are only known at runtime and hot reconfigurations have to be supported.

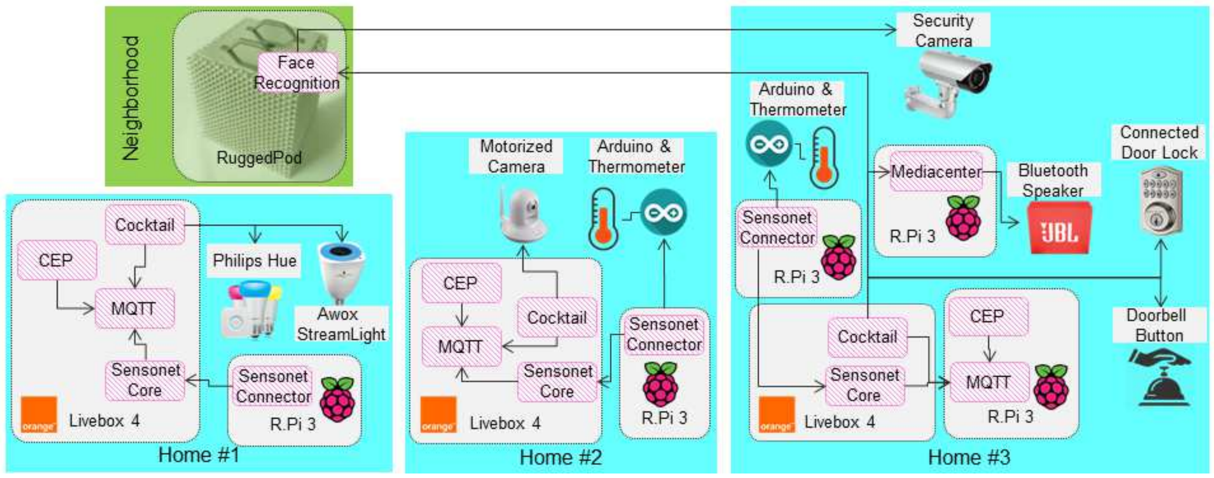

Fig. 3 Deployment of the Smart bell application onto the physical infrastructure

V. Demo setup : deployment of the application onto the infrastructure

Virtualization thanks to containers is currently commonly regarded as a "heterogeneity breaker" able to bring other advantages like isolation, packaging facilities or lightness compared to traditional virtual machines commonly used in Cloud Computing [8, 10, 11, 13]. On top of containers engines, orchestration tools can satisfy software entities dependencies and automate applications deployments and reconfigurations when infrastructure changes.

Docker, a virtualization container technology, has been selected because of the easiness to use it, its availability over multiple architectures (e.g. X86 and ARM) and its adoption in industry and academy [5, 6, 12]. Additional tools like Docker Compose or Docker Swarm seemed to meet our requirements for automation of ALM as we need to reliably reproduce this experiment several times in different environments.

Because the infrastructure is highly reconfigurable, each software pieces could be reconfigured on demand. Docker Engine, in conjunction with Docker Swarm and Docker Compose, is used to meet this purpose. Docker Engine is a container solution widely used in industry to get DevOps environment. Docker Swarm permits to manage a cluster made of multiple Docker Engines as if it was a single machine. Docker Compose allows for composition of software entities into one application. Compose offers a YAML formalism to describe Docker images to be used and runtime parameters to be solved during deployment. Compose is often used in industry on top of Swarm for DevOps to easily address deployments on clusters. For our demo, some extensions were added to Swarm & Compose to address heterogeneity and hot deployment issues:

1- Docker hosts can be tagged to add extra metadata. This feature is used to enable dependency checking. For example, if a software entity requires a Bluetooth connection, only nodes that have a "*Bluetooth*" tag can host and run this software entity.

2- All images can be provided for different hardware architectures. An smart mapping between CPU architecture (e.g. ARM vs. X86) of the target Fog node and the name of the Docker images is done, at runtime, to identify the Docker image to select regarding the software entity to instantiate on a targeted host.

VI. Feedback from experiment

The experiments that were made in building, deploying and running the demonstration in front of a live audience brought a lot of practical experience, especially on the use of Docker that was the technical cornerstone of the ALM for the demo.

The Docker environment, in conjunction with several extensions we developed (see section V), brings several good properties.

- Docker Engine and extended Swarm/Compose are a heterogeneity breaker: *hardware heterogeneity* is easily addressed through the provisioning of specific Docker images for each of the software entity. *Software heterogeneity* is handled at the Docker image level.

- Docker engine enables *isolation and resources management* at the container level.

- *Software dependencies management* can be achieved thanks to packaging capabilities provided by Docker images. Extensions made on Swarm and Compose permit to satisfy, for example, placements constraints by fulfilling the dependencies to hardware specificities (e.g. presence of Bluetooth connectivity).

- Compose allows for the *determination of runtime parameters*, and as a consequence provides a support for the hot (re)deployments of the software entities.

Altogether, the Docker suite meets the expectations of the demo regarding heterogeneity which is a major requirement in ALM for Fog environment. However, it appears that Docker technologies do not cover all the needs of our use case. The experiments brought up several other stringent requirements for ALM in the fog:

- *Granularity of the lifecycle management of the entities*. It appears clearly from our experiments that the granularity of lifecycle management in Docker, which is at the container level, could be too coarse. The lifecycle of the software entities embedded in a Docker image is hidden by the lifecycle management of the Linux container associated to it. In the demo, some extra processes must be added to Docker to manage at a finer grain (i.e. at the software entity level). As a first example, Cocktail needs some external artefacts (e.g. scripts) which actually power the mashups to orchestrate. These artefacts have to be packaged at runtime as they can be updated, added or removed at any points. In this case, a new image should be built each time a Cocktail artefact is modified. This approach was banned because it was too much time consuming and created too much network traffic. Hot copies of external artefacts into freshly created containers were preferred as depicted by part (a) of Fig. 4. Nevertheless, this approach clearly violates Docker image lifecycle by incorporating, at runtime, external resources. Another issue occurs if software components need to access hardware features as in the case of Sensonet with Arduino or Bluetooth connections (Fig. 4 (b)). To address this purpose, external wrappers are required which imply additional mechanisms to manage their lifecycles. A third example, that does not directly concern our experiment, is related to databases persistence in general as shown in Fig. 4 (c). Docker best practices recommend the creation of a container that runs a database engine on data hosted outside the container. The data have to be accessible

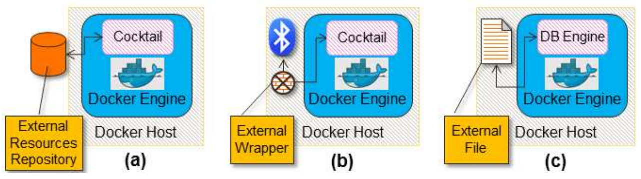

Fig. 4 Examples of Docker ALM restrictions

inside the container thanks to a mount point, which requires additional ALM process for application migrations for instance.

- *Link between the lifecycles of software and hardware entities (fog nodes and things).* In the demo, a connected door lock was used. It can be driven by a REST API. The software state, or hardware lifecycle, of the lock (i.e. open/closed and locked/unlocked) is supposed to be known by the Cocktail software entity. The Docker environment does not capture the lifecycle of the connected lock. We believe that it would be of great interest to have a common model that unifies the lifecycles of software entities (e.g. IoT Fog micro-services), hardware entities (e.g. Fog nodes) and physical world entities (e.g. things/sensors/actuators of the IoT).

- *Hierarchical distributed management.* It is not possible with Docker to distribute and hierarchically organize the management of a large set of Fog nodes by doing "cluster of clusters", for example. The Docker suite only permits flat architectures where one master drives all Docker Engines. It has impacts on the topology of the system as all Docker Engines must reside in the same network domain, which could lead to scalability and resiliency issues. Results of previous works on VAMP [4] and Vulcan [3] highlighted the benefits of this kind a distributed and hierarchical management for large distributed applications.

- *Reduced size of software packages (and network traffic).* Docker images stored in local Docker repositories could be big (e.g. over 600Mb), particularly if they are built without the respect of the best practices. Even if Docker images can be very tiny (i.e. tens of Mb), there is no warranty about Docker images lightness. This point is quite impacting in dynamic environments such as targeted by Fog Computing. It can massively impact the network bandwidth available for application [12] in the case of numerous reconfigurations. It is so important to minimize data transfers associated to reconfigurations.

VII. Conclusion : requirements for Fog ALM

This paper reports a practical and realistic testbed and demo in a Fog Computing environment that address the lifecycle management of IoT applications. Dockers and associated services helped a lot in the dynamic deployment of the demo on a heterogeneous set of Fog nodes. However, the practical experiments around the "Smart Bell" IoT Fog application have shown that Docker suite alone is not sufficient to manage all the expected properties of ALM for Fog Computing. This finding and past experiments made around Ansible [19], Vamp, Vulcan, or TOSCA [17] led us to define the following properties (the "G.U.I.D.E. properties") that we think are desirable for ALM in Fog Computing context:

- ***G**ranularity.* A level-free granularity in the management of the lifecycle of individual software entities is required to enable a fine-grain (hierarchical) management of the lifecycle of the overall application. Typically, a granularity finer that the container is mandatory to cover the whole spectrum of ALM operations, especially those that occur at runtime as encountered in our demo.

- ***U**nification.* ALM in IoT must unify knowledge about software, hardware and cyber-physical interaction capabilities to preserve the overall coherence of IoT applications. The unification must ensure that the software requirements would be supported by the capabilities and the specificities of the hardware in use. A set of packaging, deployment, and lifecycle management models and tools, have to be unified, including standard device management protocols, such as TR-069 [26] for Internet gateway, OMA LwM2M [27] for mobile devices, or UPnP MD [28] for home devices.

- ***I**ntrospection.* At each moment, states of software entities, Fog nodes and connected things have to be accessible thanks to introspection mechanisms. The ALM needs to properly decouple the software entities from the underlying infrastructure but it also needs to maintain a consistent connection between these two views. Accordance to architectural propositions from OpenFog Consortium [29] would help to keep clarify those two views. Thanks to introspection capabilities, ALM can manage events like arrival or departures of software entities or hardware nodes, and achieves consistent placements and configurations. In this respect, previous works on *Models@Runtime* [22, 23] could find interesting developments in Fog Computing.

- ***D**istribution.* Regarding scalability and resiliency issues, Fog applications, but also ALM platforms themselves that manage Fog applications, must be distributed across the whole infrastructure. To fully leverage the whole hierarchy from connected things to datacenters, ALM could/should be hierarchically organized.

- ***E**nrichment.* Both hardware and cyber-physical interaction capabilities are augmented/restricted by software entities deployed inside Fog nodes. Knowledge on the infrastructure and knowledge on the software entities must enrich one another as they are mutually linked.

Future works will target the design and experimentation of an ALM solution for IoT Fog applications in line with the "*G.U.I.D.E*" properties. Ongoing works concern the definition of an application model, an infrastructure model, and algorithms to determine the placement of software entities on Fog nodes, together with an execution engine able to actually realize the actions of deployment. Next, elasticity and other autonomic features would be considered.

Acknowledgments

The authors wish to thank M. Belaunde, N. Ben Mabrouk, S. Bolle, E. Bouvet, G. Bruneau, F. Cochet, S. Derrien, R. Dupont, X. Etchevers, D. Excoffier, B. Herard, F. Roudet, C. Seureau, and all contributors to Orange IoT Fog testbed and demo.

REFERENCES

[1] F. Bonomi et al., "Fog Computing and Its Role in the Internet of Things", *Proc. 1st MCC Workshop Mobile Cloud Computing*, pp. 13-16, 2012.

[2] L.M. Vaquero L. Rodero-Merino "Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing", *Sigcomm Computer Comm. Rev.* vol. 44 no. 5 pp. 27-32 2014.

[3] L. Letondeur, X. Etchevers, T. Coupaye, F. Boyer and N. D. Palma, "Architectural Model and Planification Algorithm for the Self-Management of Elastic Cloud Applications," *2014 International Conference on Cloud and Autonomic Computing*, London, 2014, pp. 172-179.

[4] X. Etchevers, T. Coupaye, F. Boyer and N. de Palma, "Self-Configuration of Distributed Applications in the Cloud," *2011 IEEE 4th International Conference on Cloud Computing*, Washington, DC, 2011, pp. 668-675.

[5] O. Bibani *et al.*, "A demo of a PaaS for IoT applications provisioning in hybrid cloud/fog environment," *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, Rome, 2016, pp. 1-2.

[6] O. Bibani *et al.*, "A Demo of IoT Healthcare Application Provisioning in Hybrid Cloud/Fog Environment," *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Luxembourg City, 2016, pp. 472-475.

[7] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu and M. Rovatsos, "Fog Orchestration for Internet of Things Services," in *IEEE Internet Computing*, vol. 21, no. 2, pp. 16-24, Mar.-Apr. 2017. M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[8] C. Pahl and B. Lee, "Containers and Clusters for Edge Cloud Architectures -- A Technology Review," *2015 3rd International Conference on Future Internet of Things and Cloud*, Rome, 2015, pp. 379-386.

[9] S. Yangui *et al.*, "A platform as-a-service for hybrid cloud/fog environments," *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, Rome, 2016, pp. 1-7.

[10] M. S. D. Brito, S. Hoque, R. Steinke and A. Willner, "Towards Programmable Fog Nodes in Smart Factories," *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, Augsburg, 2016, pp. 236-241.

[11] H. J. Hong, P. H. Tsai and C. H. Hsu, "Dynamic module deployment in a fog computing platform," *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Kanazawa, 2016, pp. 1-6.

[12] R. Baig, F. Freitag and L. Navarro, "Fostering Collaborative Edge Service Provision in Community Clouds with Docker," *2016 IEEE International Conference on Computer and Information Technology (CIT)*, Nadi, 2016, pp. 560-567.

[13] D. Pizzolli *et al.*, "Cloud4IoT: A Heterogeneous, Distributed and Autonomic Cloud Platform for the IoT," *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Luxembourg City, 2016, pp. 476-479.

[14] M. Blackstock and R. Lea, "IoT mashups with the WoTKit," *2012 3rd IEEE International Conference on the Internet of Things*, Wuxi, 2012, pp. 159-166.

[15] A. Kamilaris, V. Trifa and A. Pitsillides, "HomeWeb: An application framework for Web-based smart homes," 2011 18th International Conference on Telecommunications, Ayia Napa, 2011, pp. 134-139.

[16] M. Belaunde, F. Pinson and N. Pellen, "SNAP: An End User Service Composition Tool Based on Recommendations," *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, Ulm, 2014, pp. 418-421.

[17] Tobias Binz, Gerd Breiter, Frank Leyman, and Thomas Spatzier. 2012. Portable Cloud Services Using TOSCA. *IEEE Internet Computing* 16, 3 (May 2012), 80-85.

[18] M. B. A. P. Madumal, D. A. S. Atukorale and T. M. H. A. Usoof, "Adaptive event tree-based hybrid CEP computational model for Fog computing architecture," *2016 Sixteenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, Negombo, 2016, pp. 5-12.

[19] C. Ebert, G. Gallardo, J. Hernantes and N. Serrano, "DevOps," in *IEEE Software*, vol. 33, no. 3, pp. 94-100, May-June 2016.

[20] D. Wörner, T. von Bomhard, M. Röschlin and F. Wortmann, "Look twice: Uncover hidden information in room climate sensor data", *2014 International Conference on the Internet of Things (IOT)*, Cambridge, MA, 2014, pp. 25-30.

[21] B.Herard, M.Richomme, "Sensonet: a low-cost open-source objects network framework", W3C Workshop on the Web of Things, Berlin, June 2014.

[22] J. Floch et al., "Using Architecture Models for Runtime Adaptability," *IEEE Software*, Mar. 2006, pp. 62-70.

[23] Brice Morin, Olivier Barais, Jean-Marc Jezequel, Franck Fleurey, and Arnor Solberg. 2009. Models@Run.time to Support Dynamic Adaptation. *Computer* 42, 10 (October 2009), 44-51.

[24] RuggedPod website. http://ruggedpod.qyshare.com/

[25] Awox website. http://www.awox.com/

[26] TR-069 specifications.
https://www.broadband-forum.org/technical/download/TR-069.pdf

[27] Lightweight M2M description on Open Mobile Alliance website. http://openmobilealliance.org/iot/lightweight-m2m-lwm2m

[28] UPnP MD specifications.
https://openconnectivity.org/developer/specifications/upnp-resources/upnp/device-management1

[29] OpenFog Consortium Reference Architecture.
https://www.openfogconsortium.org/ra/