Edge Computing Resource Management System: a Critical Building Block! Initiating the debate via OpenStack

Ronan-Alexandre Cherrueau, Adrien Lebre, Dimitri Pertin, Fetahi Wuhib, João Monteiro Soares

Edge Computing Resource Management System: a Critical Building Block! Initiating the debate via OpenStack

Ronan-Alexandre Cherrueau, Adrien Lebre, Dimitri Pertin
STACK Research Group - IMT-Atlantique, Inria, LS2N, France

Fetahi Wuhib and João Monteiro Soares
Ericsson Research, Canada & Sweden

Abstract

While it is clear that edge infrastructures are required for emerging use-cases related to IoT, VR or NFV, there is currently no resource management system able to deliver all features for the edge that made cloud computing successful (*e.g.*, an OpenStack for the edge). Since building a system from scratch is seen by many as impractical, this paper provides reflections regarding how existing solutions can be leveraged. To that end, we provide a list of the features required to operate and use edge computing resources, and investigate how an existing IaaS manager (*i.e.*, OpenStack) satisfies these requirements. Finally, we identify from this study two approaches to design an edge infrastructure manager that fulfils our requirements, and discuss their *pros* and *cons*. This paper aims at initiating the discussion in our community.

1 Introduction

While several academic studies have highlighted the advantages of the edge computing paradigm in various domains [8, 15, 16, 19, 20], progress on how to operate and use infrastructures that serve it are marginal. Solutions like Akamai Cloudlets [1] or AWS Lambda [2] are close to the initial fog proposal that allows to run domain-specific applications on NFV-enabled infrastructures (at the edge) and centralized clouds [8]. Since these solutions do not allow to run stateful workloads in isolated environments (e.g. containers, virtual machines (VM)), they do not fully cover the requirements from developers and operators (DevOps) who expect to find most features that made current cloud solutions successful also at the edge. Therefore, such solutions are not considered in this (paper's) initial debate.

The ETSI Mobile Edge Computing Industry Specification Group defined in 2016 an architecture to orchestrate distinct independent cloud systems, *a.k.a.* Virtual Infrastructure Managers (VIM) [14]. The idea consists in federating VIMs of the different Data Centers (DCs) that compose the edge infrastructure. By reusing VIMs, ETSI targets edge computing resource managers that behave in the same fashion as traditional ones, while mitigating development requirements. Although there is no implementation available, the idea of federating VIMs seems promising as several projects have been built on similar concepts. ONAP [3], an industry-driven solution, enables the orchestration and automation of virtual network functions across distinct VIMs. From the academic side, FogBow [9] aims to support federations of Infrastructure-as-a-Service (IaaS) providers. More recently, NIST initiated a collaborative effort with IEEE to advance Federated cloud through the development of a conceptual architecture and a vocabulary[1].

Although all these projects provide valuable contributions, they all have been designed by only considering the DevOps' perspective. They provide abstractions to manage the life cycle of geo-distributed applications, but do not address administrative requirements. However, edge computing infrastructures differ from federated cloud systems in various aspects [18], for instance: edge sites are potentially unmanned and therefore must be administered remotely; management systems should be designed to cope with intermittent network access to sites; distinct operators might be interested in interconnecting their infrastructures (like network peering).

To capitalize on the advent of edge computing, our community should take part in current discussions and actions in order to deliver a well-suited resource management system. A system that will (i) let an operator aggregate, supervise and expose the massively distributed resources of the infrastructure, and (ii) let DevOps implement new kinds of services on top of an infrastructure that may be deployed and managed on-demand.

In this paper, we present reflections to initiate discussions through our community.

- First, we introduce a classification of the features expected by both administrators and DevOps. This classification is valuable to identify missing mecha-

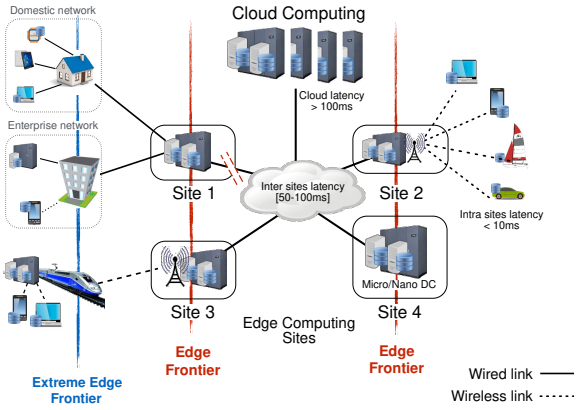

Figure 1: Edge Computing Infrastructure [11]. The red dashed lines depict a split-brain situation that isolates *Site 1* from other sites.

nisms in resource management systems.

- Second, based on the identified requirements, we discuss how an edge resource management system should be designed. In particular, we study *pros* and *cons* of *top-down* and *bottom-up* approaches. The former consists in interacting with each site only through exposed VIM APIs, such as in federated approaches. The latter aims at revising internal mechanisms of VIMs to enable native collaborations.

Since there are many possible edge infrastructure designs, we highlight that the infrastructure considered in this study is composed of several individually-managed and geo-distributed micro DCs (up to thousands) composed of up to one hundred servers (nearly two racks). Figure 1 depicts such an infrastructure. The expected latency and bandwidth between elements may fluctuate, in particular because networks can be wired or wireless. Moreover, disconnections between sites may occur, leading to network split-brain situations [12]. Finally, it is possible to consider additional DCs at the extreme edge, within private institutions or public transports. From the software viewpoint, since we investigate how existing cloud managers can be extended to the edge, we consider as our default VIM OpenStack [5]: the de-facto open-source solution for cloud computing infrastructures.[2]

The remainder of the paper is organized as follows. Section 2 provides a list of features expected by administrators and DevOps. Section 3 studies how OpenStack satisfies these features, highlighting the need for collaboration across the entire edge infrastructure. Section 4 discusses two approaches to design a resource management system for the edge. Section 5 concludes the paper.

2 Admin/DevOps' Requirements

Herein we classify features administrators and DevOps expect to find in the context of edge computing infrastructures. The classification is based on 5 *levels*, starting from the easiest aspects, *i.e.*, interacting with a single site (considered in level 1, L1), to more complex aspects like managing multiple sites (L2), up to taking into account that sites can be owned by different operators (L5). Table 1 summarizes the classification we detail below.

As previously mentioned, a large part of these features are common to the ones offered by current IaaS resource management systems. They are implemented by various services, each of which is in charge of the management of a particular aspect of the infrastructure [13].

L1. Operate/use any site This level considers the actions both administrators and DevOps expect to perform on a single, reachable site. Most operations are elementary from the edge viewpoint because they correspond to the ones already provided by a VIM such as OpenStack. In other words, each edge site can be considered as an independent cloud at this level. The unmanned aspect only impacts this level by requiring to perform all operations remotely if needed. Furthermore, the resource management system should provide means to ensure the integrity of the hardware resources taking part to the edge infrastructure. Strategies such as enabling/disabling physical interactions with the equipment should be considered.

L2. Operate/use several sites In L2, L1 features are considered but over several sites. This includes operations such as provisioning/managing multiple resources or gathering information from various sites simultaneously. Operations can be either intra-service (same service from different sites) or inter-service (different services from different sites). Examples of such operations include configuring users' access on a per-site basis; listing available VM images or pushing new ones on multiple sites. From the DevOps viewpoint, a user should be able to boot a VM on *Site 1* using an image defined in *Site 2* (inter-service operation). Similarly to L1, DevOps also expect metrics to be collected from several sites and collaboration mechanisms regarding the security (*e.g.*, secret key sharing, network encryption).

Because collaboration between sites can be either explicit (*i.e.*, the targeted sites are explicitly specified in the operation), or implicit (*i.e.*, the resource management system is in charge of selecting resources), we have defined two sub-levels: L2.1 and L2.2, as depicted in Table 1. The implicit manner suggests that policies (*e.g.*, performance objectives, energy consumption) and constraints (*e.g.*, affinity rules, hardware requirements) are provided by admins/DevOps so that the resource management system takes the right decisions regarding the defined desiderata and the state of the infrastructure (*e.g.*, auto-scaling, relocating workloads between sites, re-scheduling faulty resources across sites).

Levels	*Administrators*	*DevOps*	*Both*
L1: Operate/use any site	Manage any site: install, upgrade site's services; manage users, flavors, quotas	Provision compute, storage, network resources on-demand on any site	Collect metrics and ensure security, integrity and resiliency for any site
L2: Operate/use several sites	Manage multiple sites simultaneously	Cross-site collaborative resources	L1 but over a set of sites
- L2.1 Explicit manner:	Manage a specific set of sites	Provision on a specific set of sites	Aggregated metrics from multiple sites
- L2.2 Implicit manner:	Cross-site autonomous management	Cross-site autonomous provisioning	and collaborative security mechanisms
L3: Robustness *w.r.t.* split brains			L1 for an isolated site; L1 and L2 for
- L3.1 Application robustness:	——	Access reachable applications	isolated sets of sites
- L3.2 Management service robustness:	Manage reachable site(s)	Provision on reacheable site(s)	Support intermittent connectivity
L4: Multiple Cloud environments			L3 with different IaaS environments
- L4.1 Different IaaS versions:	Manage different IaaS versions	Provision on different IaaS versions	Discover sites' capabilities and
- L4.2 Different IaaS technologies:	Manage different IaaS technos	Provision on different IaaS technos	compatibility
L5: Multiple operators	——	Provision on one or many sites	L4 with multiple operators

Table 1: Classification of the requirements to operate and use edge computing infrastructures in 5 levels.

L3. Robustness *w.r.t.* network split-brains L3 includes L2 operations but with the possibility to face situations where the infrastructure is partitioned due to network disconnections. Figure 1 depicts such a case where *Site 1* is isolated from the other sites. In this scenario, administrators/DevOps that can reach *Site 1* (*i.e.*, located in the same geographical area) should be able to perform L1 operations on *Site 1* despite the split-brain. Such a requirement makes sense as L1 is limited to one site and does not impact other ones. For the remaining sites, L1 and L2 must be guaranteed.

Since split-brains might impact differently already-provisioned resources and management services, we refine L3 into two sub-features. L3.1 is related to features that allow already-deployed applications to continue to serve local requests without being impacted. For instance, an *apache* server or a storage backend should be able to satisfy requests coming from the same geographical area, even if management services cannot be reached. L3.2 features are related to the provisioning of new resources and other management operations as described in the previous paragraph. Note that guaranteeing L2 features will not be always possible at the L3.2 level because information cannot be gathered in case of a disconnection. For instance, guaranteeing quotas across the infrastructure might be a challenge without first relaxing the consistency requirement of the information. New approaches will have to be proposed for such operations.

Finally, the possibly intermittent network connectivity between edge resources requires the ability for sites to join and leave the infrastructure.

L4. Multiple cloud environments Delivering a resource management system at large scale in a unified manner poses a great challenge from the software viewpoint. Since different versions and types of infrastructure managers might co-exist at the same time across the whole infrastructure, L4 gathers the related requirements. More precisely, L4.1 considers L3 features when different versions of the same software stack co-exist across the infrastructure. L4.2 increases the complexity by considering the collaboration of mutiple systems, including possibly different concepts (*e.g.*, OpenStack for VMs and Kubernetes for Containers). Such a requirement implies the ability to get sites' capabilities in order to only allow meaningful collaborations.

L5. Multiple operators L5 corresponds to the holy grail in terms of expected features. In addition to L4, it includes the possibility to use sites owned by different operators. We do not specify any requirement for administrators at this level as an operator is unlikely to let other operators administer its own site. However, operators should be able to collaborate to offer their sites' resources to any DevOps like it has been done for a while for cellular networks. The requirements here are more related to the collection and sharing of relevant metrics enabling each operator to perform charging/billing.

Summary As previously mentioned, each level of our classification increases the complexity in terms of design and development constraints. Note however that since L4 and L5 both extend L3, they can be considered at the same level and can be swapped as a consequence.

Although we tried to be exhaustive, this list of features could probably be extended, *e.g.*, by considering different edge infrastructure scenarios (*e.g.*, including smaller and limited devices). However, we believe it is already valuable as it delivers significant insights on the design and implementation of an edge resource management system. In the next section, we study whether OpenStack can fulfill the L1, L2, and L3 levels. The discussion of L4 and L5 is left as future work.

3 OpenStack at the Edge

This section studies the use of OpenStack to control an edge infrastructure. Such an analysis is meaningful as the OpenStack code base has been evolving to deal with large scale and multi-site objectives for the two last years. Particularly, the section evaluates OpenStack, including its latest improvements, with respect to the requirements defined in the previous section.

At a high level, OpenStack has two types of nodes: data nodes delivering XaaS capabilities (compute/storage/network, *i.e.*, data plane); control nodes executing OpenStack services (*i.e.*, control plane). Whenever users

issue a request to OpenStack, the control plane processes the request which may potentially also affect the data plane in some manner. Key control plane services include `keystone`, `nova`, `glance`, and `neutron`, respectively responsible for authentication/authorization, VM life cycle management, VM image management and associated network management.

Because OpenStack comes with several deployment alternatives and because the edge sites considered in our study can host data and control nodes, we elected to discuss two scenarios: a centralized management scenario and a multiple regions scenario described next.

3.1 Centralized (Remote) Management

In this scenario, OpenStack operates an edge infrastructure as a traditional single DC environment, the key difference being the wide-area networking found between the control and compute nodes [7]. The distinction between the different edge sites can be realized by leveraging the concept of host aggregates within OpenStack.

From the requirements' viewpoint, L1 and most L2 requirements can be fulfilled in a straightforward way (because the infrastructure can be spread over several network domains, some L2 operations including specific network actions cannot be satisfied). For L3, only L3.1 requirements can be satisfied while L3.2 cannot be met due to the centralized control plane. For instance, most OpenStack services collaborate through a message system and through the manipulation of logical objects that are persisted in shared databases (DBs). While this enables services to easily collaborate, it imposes a requirement of permanent connectivity between the services located at the compute nodes and the control services like the databases and message buses located in the DC. In other words, while this scenario provides a "single pane of glass" for administrators and DevOps, it has the drawback of being a "single point of failure" preventing DevOps to use edge resources in case of network split-brains.

3.2 Multiple Regions

In this scenario, each edge site corresponds to a *region* in the OpenStack terminology, which is a complete deployment of OpenStack with all control services and a "shared" Keystone. The main advantage of this deployment is related to the independency of each site in case of network disconnections. The downside relates to the fact that the current codebase does not provide any mechanism to allow the collaboration between several regions and thus L2 requirements cannot be met. [3]

3.3 Effective Collaboration is Needed

Despite the fallibility of the network, and frequent isolation risks of an edge site from the rest of the infrastructure, an edge infrastructure may be able to meet L3 requirements. This can be achieved by supposing a collaboration a la *peer-to-peer*, that is, an edge site always serves local resources and collaborates with other edge sites if need be. To develop such a resource management system, two fundamental design options exist: *top-down* or *bottom-up*. Both designs differently impact the way this required collaboration can be handled.

A top-down collaboration design implements the collaboration by federating VIMs' API. As a consequence the existing VIM codebases are used without introducing modifications/extensions. Examples of approaches following this design are: ONAP [3], Kingbird [4], FogBow [9] and p2p-OpenStack [17].

A bottom-up collaboration design lays on making VIMs mechanisms/services directly collaborative [11]. For example, having two OpenStack Nova services able to cooperate and communicate directly would be a realization of a bottom-up design. Such design implies either the modification/extension of existing VIMs or the creation of a completely new system.

4 Design Discussion

In this section, we discuss the pros and cons of the *top-down* and *bottom-up* designs targeted at the end of the previous section. More precisely, our discussion is driven by the following questions. (i) On the one hand, while the top-down design is the most common approach, can it fulfill all the expected requirements listed in our classification without requiring changes in the VIM codebase? (ii) On the other hand, while the bottom-up design seems to disrupt the design principles by requiring *a la peer-to-peer* strategies in VIM's internal mechanisms, should it be discarded?

Top-Down Design The top-down approach consists in designing a set of overlay components that interact with existing VIM APIs to avoid modifying VIM codebases. Avoiding codebase modifications is of particular importance in fast-growing software stacks. For example, a new version of OpenStack is released every six months with a lot of changes in the codebase, whereas changes rarely impact the APIs. Therefore, a top-down design makes VIM development and the overlay system components independent. If designed to do so, the system can easily allow L4, *i.e.*, the support of different versions and types of VIMs - as demonstrated by FogBow [9].

Unfortunately, a top-down design cannot satisfy all L2 requirements without extending or revising the existing VIM codebase. For example, OpenStack Tricircle [6], a top-down project to allow virtual networking across different sites, ended up "breaking" the core of OpenStack by introducing specific L2 mechanisms. Such intrusive

modifications negate the aforementioned independence. Moreover, L2 features in general require reimplementing many low-level VIM functionalities at the overlay level. For instance, the OpenStack "boot a VM" process looks as follows from a bird's-eye view: (1) Get the URL of the image by looking up in the database, (2) Schedule and boot the VM. Thus, booting a VM on *Site 1* using an image defined in *Site 2* would require implementing a dedicated workflow at the overlay level in order to interact with both sites and copy the image from the image manager (*i.e.*, Glance) of *Site 2* to the one of *Site 1* before booting the VM. This is valid for most L2 features such as a fine-grained authorization management with different rights in different edge sites (L2.1) or a cross-site scheduling functionality when sites are specified implicitly (L2.2). Such a mechanism will be similar to the placement workflow already available in `Nova`.

Bottom-Up Design In a bottom-up design, the system is not limited by what is available through VIM APIs. Specifically, there are two possibilities: i) rearchitecting the services/components of existing cloud platforms, in our case OpenStack; ii) through a clean-slate approach.

By rearchitecting OpenStack to allow native collaboration, several 'local' features can be supported across the entire edge infrastructure for free. For instance, the aforementioned OpenStack "boot a VM on Site 1 with image on Site 2" process would be feasible without modifying the VIM codebase if *Site 1* can either directly contact the image manager of *Site 2*, or share the database backend with *Site 2*. However, since OpenStack has not been designed to be collaborative, most mechanisms must be revised to consider side-effects related to collaboration operations. For instance, a VM boot process initiated on *Site 1* can be completed on *Site 2*. The question is then to define where the states related to this VM should be stored, keeping in mind the split-brain challenge. It is noteworthy that identifying all these aspects requires the understanding of existing code, which is a daunting task in respect to the OpenStack ecosystem size.

Therefore, the complexity of rearchitecting existing services/components might make a clean-slate approach as the most likely approach for a bottom-up design.

Finally, L4 requirements cannot be intrinsically satisfied by the bottom-up approach while L5 implies strong limitations regarding how collaborations should be implemented (for instance sharing internal states of different edge sites between operators looks unlikely).

Summary There are *pros* and *cons* for both approaches, and none individually seems to meet all requirements. From the technical perspective, the bottom-up design seems to be the most appropriate to cope with L1, L2 and L3 requirements, while the top-down is the only one to satisfy L4 and L5. From the business perspective, a top-down approach is likely to fulfill the short and medium-term Time-To-Market (TTM) requirements while a bottom-up approach is required to fulfill all requirements. From the codebase perspective, top-down approach has limited impact on existing VIM codebase while the bottom-up, despite the impact on existing codebase, allows implementing functionalities efficiently, without code duplication. All in all, our community should investigate a long-term solution that builds on a bottom-up approach. This will require first diving into OpenStack and understanding its intricate internal mechanisms leveraging tools like EnOS [10] to mitigate the efforts, and second, to address scientific challenges such as the definition of a reference architecture for a resource management system for edge infrastructures, how to share internal system states in an edge context etc.

5 Conclusion

The emergence of Network Function Virtualization (NFV) technologies as well as Internet of Things (IoT) and Augmented/Virtual Reality (AR/VR) applications herald a new era of cloud computing where infrastructures need to leverage resources at the edge of the network in order to cope with latency requirements. Despite this growing need, there is no cloud management system that is designed with such an infrastructure in mind.

In this paper, we presented reflections to initiate discussions on this hot topic. We outlined a systematic grouping of the features that administrators/DevOps expect and the requirements an edge infrastructure puts on a cloud management system. We briefly studied the use of existing IaaS managers (*i.e.*, OpenStack) to control an edge infrastructure highlighting the need for an effective collaboration between the edge sites. We then discussed pros and cons of two possible strategies to follow when designing a solution: (1) a *top-down* approach which designs overlay components that interact with underlying IaaS managers without modifying their codebase; (2) a *bottom-up* approach that extensively improves existing software to fulfill the requirements. We concluded that in the short-term, a *top-down* approach is required to have a timely working solution available. In the long-term, both approaches should be considered simultaneously: *bottom-up* to realize a native and efficient system for the edge; *top-down* to enable collaboration between different cloud stacks (*e.g.*, OpenStack, Kubernetes).

Finally, it is noteworthy that our study considers one kind of edge infrastructure. Other variants can be envisioned where, for example, third-party hardware resources can dynamically join and leave the infrastructure. For such cases, this study needs to be extended to integrate additional features not discussed in this paper.

References

[1] Akamai Cloudlets. http://cloudlets.akamai.com. (Accessed: 2018-03-08).

[2] Amazon Lambda@Edge. https://aws.amazon.com/lambda/edge/. (Accessed: 2018-03-08).

[3] Open Network Automation Platform. https://www.onap.org. (Accessed: 2018-03-08).

[4] OpenStack Kingbird. https://wiki.openstack.org/wiki/Kingbird. (Accessed: 2018-03-19).

[5] OpenStack: Open source software for creating private and public Clouds. https://www.openstack.org. (Accessed: 2018-03-08).

[6] OpenStack Tricircle. https://wiki.openstack.org/wiki/Tricircle. (Accessed: 2018-03-19).

[7] Toward Fog, Edge, and NFV Deployments: Evaluating OpenStack WANwide, Boston OpenStack Summit. https://www.openstack.org/videos/boston-2017/toward-fog-edge-and-nfv-deployments-evaluating-openstack-wanwide, April 2018. (Accessed: 2018-03-19).

[8] BONOMI, F., MILITO, R., ZHU, J., AND ADDEPALLI, S. Fog Computing and its role in the Internet of Things. In *Proceedings of the first edition of the MCC workshop on Mobile Cloud Computing* (2012), ACM, pp. 13–16.

[9] BRASILEIRO, F., SILVA, G., ARAÚJO, F., NÓBREGA, M., SILVA, I., AND ROCHA, G. Fogbow: A middleware for the federation of IaaS Clouds. In *The 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (2016), IEEE, pp. 531–534.

[10] CHERRUEAU, R.-A., PERTIN, D., SIMONET, A., LEBRE, A., AND SIMONIN, M. Toward a holistic framework for conducting scientific evaluations of OpenStack. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2017), IEEE Press, pp. 544–548.

[11] LEBRE, A., PASTOR, J., SIMONET, A., AND DESPREZ, F. Revising OpenStack to operate Fog/Edge Computing infrastructures. In *The IEEE International Conference on Cloud Engineering (IC2E)* (April 2017), pp. 138–148.

[12] MARKOPOULOU, A., IANNACCONE, G., BHATTACHARYYA, S., CHUAH, C. N., GANJALI, Y., AND DIOT, C. Characterization of failures in an operational ip backbone network. *IEEE/ACM Transactions on Networking 16*, 4 (Aug 2008), 749–762.

[13] MORENO-VOZMEDIANO, R., MONTERO, R., AND LLORENTE, I. IaaS Cloud architecture: From virtualized datacenters to federated Cloud infrastructures. *Computer 45*, 12 (Dec. 2012), 65–72.

[14] SABELLA, D., VAILLANT, A., KUURE, P., RAUSCHENBACH, U., AND GIUST, F. Mobile-Edge Computing architecture: The role of MEC in the Internet of Things. *IEEE Consumer Electronics Magazine 5*, 4 (Oct 2016), 84–91.

[15] SATYANARAYANAN, M. The emergence of Edge Computing. *Computer 50*, 1 (2017), 30–39.

[16] SHI, W., CAO, J., ZHANG, Q., LI, Y., AND XU, L. Edge Computing: Vision and challenges. *IEEE Internet of Things Journal 3*, 5 (2016), 637–646.

[17] SOARES, J. M., WUHIB, F., YADHAV, V., HAN, X., AND JOSEPH, R. Re-designing Cloud platforms for massive scale using a P2P architecture. In *The IEEE International Conference on Cloud Computing Technology and Science (CloudCom)* (Dec 2017), pp. 57–64.

[18] THE OPENSTACK FOUNDATION. Cloud Edge Computing: Beyond the Data Center (White Paper). https://www.openstack.org/assets/edge/OpenStack-EdgeWhitepaper-v3-online.pdf, Jan 2018. (Accessed: 2018-03-08).

[19] YI, S., HAO, Z., QIN, Z., AND LI, Q. Fog Computing: Platform and applications. In *The Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)* (2015), IEEE, pp. 73–78.

[20] ZHANG, B., MOR, N., KOLB, J., CHAN, D. S., LUTZ, K., ALLMAN, E., WAWRZYNEK, J., LEE, E. A., AND KUBIATOWICZ, J. The Cloud is not enough: Saving IoT from the Cloud. In *HotStorage* (2015).

Acknowledgement

A large part of the content presented in this document results from numerous discussions with the OpenStack community, notably through FEMDC SiG and Edge Working Sessions. Further information available at: https://wiki.openstack.org/wiki/Fog_Edge_Massively_Distributed_Clouds.

Notes

[1] https://collaborate.nist.gov/twiki-cloud-computing/bin/view/CloudComputing/FederatedCloudPWGFC (March 2018).

[2] Note that our study applies to any other resource manager built on the similar building blocks as OpenStack (*e.g.*, Kubernetes). Such building blocks are conceptually identified in [13].

[3] Due to space limitation, we did not discuss a third scenario leveraging the OpenStack *cells* concept. However, we underline that such an approach has the drawback of the two discussed scenarios.