fogØ5: Unifying the computing, networking and storage fabrics end-to-end

Conference Paper · July 2018

DOI: 10.1109/CIOT.2018.8627124

2 authors:

Angelo Corsaro
ADLINK Technology
64 PUBLICATIONS **694** CITATIONS
SEE PROFILE

Gabriele Baldoni
ADLINK Technology
7 PUBLICATIONS **17** CITATIONS
SEE PROFILE

fog∅5: Unifying the computing, networking and storage fabrics end-to-end

Angelo Corsaro, PhD
Advanced Technology Laboratory
ADLINK Technologies, Inc.
28 rue Jean Rostand, 91400 Orsay, France

Gabriele Baldoni
Advanced Technology Laboratory
ADLINK Technologies, Inc.
28 rue Jean Rostand, 91400 Orsay, France

***Abstract*—Fog computing aims at providing horizontal, system-level, abstractions to distribute computing, storage, control and networking functions closer to the user along a cloud-to-thing continuum. Whilst fog computing is increasingly recognized as the key paradigm at the foundation of Consumer and Industrial Internet of Things (IoT), most of the initiatives on fog computing focus on extending cloud infrastructure. As a consequence, these infrastructure fall short in addressing heterogeneity and resource constraints characteristics of fog computing environments.**

In this paper, we (1) explain the requirements of fog computing infrastructure and how they extend well beyond those traditionally addressed by Cloud Computing infrastructures; (2) introduce fog∅5, a fog Infrastructure that unifies computing, networking and storage fabrics end-to-end, while addressing the challenges imposed by resource heterogeneity, (3) explain the novel architectural approach adopted by fog∅5 to have a server-less data-centric architecture that is scalable, secure, and highly resilient to failures, (4) demonstrate the use of fog∅5 in some real-world use cases and (5) conclude and reports on future works.

***Index Terms*—IIoT; Fog Computing; Edge Computing; MEC; Internet of Things; cyber-physical systems;virtualisation;IoT infrastructure;fog05;fog computing platform;function virtualization; platform-as-a-service layer; Multi-Access Edge Computing;**

I. Introduction

Early Internet of Things (IoT) applications adopted cloud-centric architectures where information collected from *things* is processed in a cloud infrastructure and decisions are pushed back from the cloud to things.

While this architectural paradigm is suitable for a subset of Consumer IoT (CIoT), it quickly shows its limitation in the context of Industrial IoT (IIoT). More specifically, the following assumptions, at the foundation of cloud-centric architectures, are generally violated in IIoT applications:

- **Connectivity.** Cloud-centric architectures assume that *things* are sufficiently often connected. While this is mostly true for IoT applications, it is far from being the common case in IIoT applications. As an example, autonomous agricultural vehicles, or robots in a smart farm are often deployed in locations with very poor connectivity.
- **Latency.** Cloud-centric architectures assume applications can tolerate the latency associated with pushing data from *things* to the cloud, processing information on the cloud and eventually sending back some control information. This latency, is orders of magnitude higher of the reaction times required by several IIoT applications, such as, autonomous vehicles, smart factories and smart grids.
- **Throughput.** Cloud-centric architectures assume that the throughput required to push data from **things** to the cloud may be massive when looking at the aggregate traffic, but it is generally composed by limited individual flows. In IIoT the situation is quite different as, often, there are data flows with high individual throughput and in several applications the aggregate volume is incredibly high. This makes it unfeasible or not very effective to stream this massive volumes of data to a data center.
- **Cost of Connectivity.** CIoT commonly assume that the cost of connectivity is negligible. This stems from the fact that consumer pays for connectivity – either via their mobile data plan or their home internet connection. In IIoT the situation is completely different, it is the owner of the system that pays for connectivity. This cost is non-negligible in applications with large number of data streams, such as Smart Grids as well as for applications deployed in remote areas such as oil exploitation rigs that can only rely on expensive satellite communication where 1MByte of data can cost as much as $8!
- **Security.** Cloud-centric architectures operate under the assumption that end-users are comfortable in giving away their data. While this may be true for CIoT applications, the situation is completely

different in IIoT. Information is a strategic asset which the vast majority of companies, operating in an industrial, do not want to leave their premises.

Fog computing has emerged as an architectural approach to deal with the limitations exposed by cloud-centric architectures in the context of IIoT applications.

A. Fog Computing

Early fog computing initiatives and demonstrations [9], focused on enabling cloud-less architectures by leveraging edge infrastructure. The main aim was exploiting relatively capable edge infrastructure to bring computing closer to were data was being produced and control needed to be actuated. Things were left out of the picture. In other terms we had moved from a cloud-centric to an edge-centric architecture but not necessarily toward an end-to-end solution. As a result, *mist-computing* [5], [10] was evoked as the virtualisation of computing, storage and communication resources on things. Thus, for some time, instead of unifying the perspective, the community was segregating around the three tiers found in a typical CIoT and IIoT, in other terms the cloud, the edge and the things.

The segregation was eventually resolved by the OpenFog Consortium Architecture Working Group in the vision paper [7], where the authors of this paper, along with the key companies driving fog computing agreed on the definition reported below.

Fog computing. *A system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum* [7]

In other terms, fog computing aims at addressing the technological fragmentation existing today across the cloud the edge and things, by providing a unifying abstraction. This should sound completely logic as in general CIoT and IIoT applications will span across the three tiers and it would be ideal to have a unified set of abstractions to manage them.

B. Multi-Access Edge Computing

While the IoT community was debating about the need of fog computing, the telecommunication community started working on the concept of Multi-Access Edge Computing (MEC). Whilst MEC and fog computing emerged from different communities the main problem they try to solve is essentially the same, exception made for few differences w.r.t. the application domain and the induced constraints.

In other terms, MEC aims at providing unified management across the cloud down to the network edge. Fog computing, on the other end, expands down to things. The other big difference, is that as fog computing infrastructures have to deal with industrial real-time applications. As such, their ability to manage and virtualize resources in a real-time environments is essential. Beside this differences, MEC and fog computing share very similar requirements, this has been acknowledged by the collaborations between ETSI and the OpenFog Consortium, announced in September 2017, for driving the convergence.

In this paper we introduce **fog**$\varnothing$5 a fog computing infrastructure that unifies compute, storage and networking across cloud, edge and things. The reminder of the paper is organized as follows. Section II summarizes the state of the art and identifies the gaps and the weakness intrinsic in the approach taken thus far, *i.e.*, that of reusing cloud-based infrastructures for fog computing; Section III introduces **fog**$\varnothing$5, its goals and its core abstractions; Section IV explains in details the architecture of **fog**$\varnothing$5 and how it enables security and decentralization; Section V describes the uses cases for which **fog**$\varnothing$5 has been adopted and those we are working on; finally in Section VI we draw the conclusions and summarize future work.

II. State of the Art

The research and development on fog computing has focused on adapting infrastructures conceived for cloud computing, such as the OpenStack, to fog computing. For instance, in [4] an OpenStack based infrastructure is proposed to serve as a fog layer for smart cities. In [8], the authors identify some of the limitations induced by OpenStack's centralized architecture and propose the use of a distributed key value store to eliminate the dependency on a centralized database. While these contributions are on the right track toward enabling fog computing, they are biased toward virtualizing the edge infrastructure as opposed to providing an end-to-end solution that can virtualize, manage and operate from the cloud to the thing while addressing the heterogeneity specific to fog computing.

In the reminder of this section we present a summary of the key architectural aspects of OpenStack. Then, we motivate the need for a different solution by showcasing how OpenStack fails in addressing the key requirements of fog-computing.

A. OpenStack Overview

OpenStack adopts a modular service-oriented architecture, where each component provides a service to other components. Only a minimal set of services is required in order to run a minimal OpenStack deployment, these are:

Requirement	OpenStack	Fog Computing Platform
Dynamic discovery of new hosts/devices/VIM	no	yes
Discovery of specific HW and I/O the devices	no	yes
Decentralized Architecture	no	yes
Low Latency	no	yes
Low Resource Utilisation	no	yes
Generalized Constraint-based Deployment	no	yes
Devices assignment to entities	PCI Only	yes
Support for resource constrained devices	no	yes
Unification of computing, networking and storage end-to-end	no	yes
Large scale deployment	complex	simple
Extensibility	partial plugin architecture	full plugin architecture
Support for Real-Time OS	no	yes

TABLE I
FOG COMPUTING REQUIREMENTS

- **Horizon**. This component provides a web dashboard used to interact with OpenStack.
- **Nova**. This component provides the virtualisation layer and controls the instantiation of VMs and Linux Containers (Nova-LXC). Nova is is composed by a number of sub-components usually installed in different machines, the most important of these being:
  - *nova-compute*. Enables the communication with the underlying hypervisor.
  - *nova-api*. Provides a RESTful API for accessing the nova service.
  - *Advanced Message Queueing Protocol (AMQP) MessageQueue*. Provides the RPC mechanism used inside nova.

  These services rely on a database for storing state information about VMs and a RabbitMQ server for providing the AMQP message queue.
- **Glance**. This service is responsible for storing images. It is in charge of the creation of VMs' virtual drives and for managing base images used to provision the VMs. It uses a database to store information about virtual disks and images. It also provide a RESTful API for interaction.
- **Neutron**. This component provides the provisioning of networks, for virtual (overlay networks) and physical networks. Neutron exposes a RESTful API for interacting with other components and *external world*. For internal communication it uses an RPC mechanism based on AMQP. Finally, it relies on a database to store information about the network state.
- **KeyStone**. It provides the identity service used by components for authentication. It also exposes a RESTful API to interact with other services as well as external components.

In summary, all of these components use (1) a RESTful API for external or inter-component communication, (2) an AMQP based RPC for inter-component communication, and (3) a data-base for storing component related state.

The minimal OpenStack deployment requires two servers hosting the following services:

- *Controller Node*
  - SQL Server Database
  - RabbitMQ Server
  - HTTP Server
  - Keystone
  - Glance
  - Nova
  - Neutron
  - Horizon
- *Compute Node*
  - Nova with nova-compute
  - Neutron

B. Fog Computing Requirements

From the list above, it emerges how OpenStack has a clear client-server architecture with dependencies toward centralized services such as the AMQP broker and the database. From the list of services required for the minimal deployment, it also follows that a *minimal* installation requires relatively resourceful hardware.

This is a consequence of the fact that OpenStack was designed for resourceful, symmetric and relatively static environments such as data centers. A typical fog deployment is rather the opposite as (1) nodes may not be extremely resourceful, (2) the system is highly asymmetric w.r.t. computational and connectivity capabilities, and (3) nodes have a high degree of churn which beside from failures and intermittent connectivity stems from mobility – consider for instance autonomous driving vehicles in a factory.

Another key characteristic of fog environments is the need for supporting low reaction times, as well as the provisioning, in addition to VM and containers, of different kinds of deployable units, such as unikernels, micro-services, and binary executables, some of which may

need to run on real-time operating systems and leverage real-time networks such as Time Sensitive Networking (TSN) (IEEE 802.1) [11].

OpenStack, operates at a time scale that is incompatible with these kinds of systems. As an example the instantiation of a unikernel is dominated by OpenStack as opposed to the unikernel boot time. This clearly offsets the advantage of using unikernels for fast boot.

Table I summarizes some of the key requirements of a fog infrastructure and evaluates their support in OpenStack. From this table it emerges that OpenStack is ill-suited as the starting point for fog infrastructures.

III. FOG∅5 Overview

As described in the previous sections, the fog infrastructures proposed thus far in literature [3], [6], [9], [12], while being a step in the right direction, fall short in addressing the end-to-end unification of resources. The main limitations of previous solutions can be summarized as (1) deployment limited mostly to VM and/or Containers, (2) lack of support for real-time, (3) lack of mudularity with consequences on extensibility, and (4) limited security.

fog∅5 defines a set of abstractions to unify the compute, storage and communication fabric end-to-end and thus allow applications to be managed, monitored and orchestrated across the cloud to thing continuum.

Before delving into the architectural details of **fog**∅5 let's see the key abstraction that it uses in order to provision, manage and orchestrate, generic applications end-to-end.

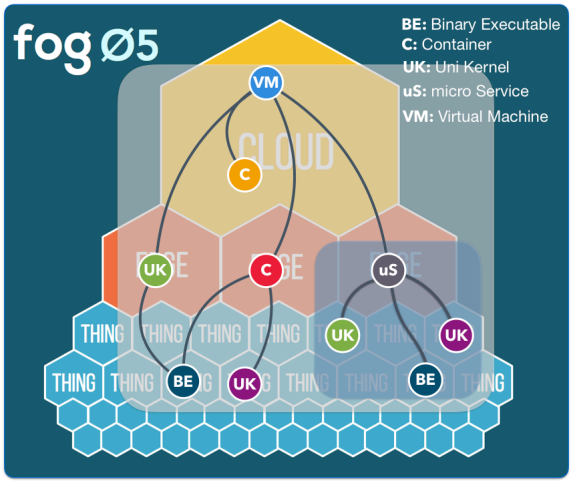

Fig. 1. **fog**∅5 unified abstraction

A. *fog*∅5 *Entity*

The abstraction used by **fog**∅5 to provision, manage and orchestrate applications, or network functions, is the *entity*. An **fog**∅5*entity* is either an *atomic entity*, such as a Virtual Machine, a container, a Unikernel, a binary executable, or a Directed Acyclic Graph (DAG) of entities (see Figure 1). Where the set of atomic entities supported by **fog**∅5 can be extended through plugins.

Entities and *Atomic Entities* have a FSM that defines the legal state transitions. These FSM have been defined to be sufficiently generic to encompass essentially any kind of *atomic entity* ranging from a VM to a binary executable. Entities may define a deployment affinity w.r.t. to each other as well as with respect to compute, storage, I/O and accelerators such GPUs and FPGAs.

The state machines that define the legal state transitions for *entities* and *atomic entities* are depicted in Figure 2a and and Figure 2b respectively.

As shown in Figure 2b, the statuses of an *atomic entity* are distinguished from those of an *instance* of an *atmic entity*. This is similar to the difference between the Virtual Network Function (VNF) Descriptor (VNFD) and theVNF Record (VNFR), were the first represents the *template* and the latter *instance* for that *template*. **fog**∅5 also provides an implementation for the ETSI VNF Managment and Orchestration (MANO) framework. Additional MANO can be supported by implementing a plugin.

In Figure 2a it can also be seen how *entities* have a FSM describing their legal state transitions. Each *entity* is described as a DAG of *atomic entities* or *entities*. The DAG defines a partial order indicating the start-up dependencies. Additionally, an entity affinity graph can be defined to provide hints concerning the entities that need to be close to each other.

B. *fog*∅5 *Resources*

fog∅5 uses resources, expressed as URI, to represent everything in the system. For instance, a node is a resource, as is a resource a FPGA or a specific GPIO available on that node, a fieldbus interface, and so on. Resources are organized as a tree whose root indicates the administrative domain. The syntax used for resources is reported below:

```
<a|d>fos://<system_id>/<node_id>/<type_of_resource>
          /<id_of_resource>/<type_of_subresource>
          /<id_of_subresource>
          /...['?'query]['#'fragment]
```

In the URI it is possible to have wildcards, query and fragments. Sub-resources can be nested, some types of resources and sub-resources are reserved and used by the **fog**∅5 agent

Two types of wildcards are allowed:

- * to indicate an arbitrary sub-path of length 1.
- ** to indicate a sub-path of arbitrary length.

URI fragments are used to represent delta-updates to the value of a resource. Queries are used to retrieve resources matching a given predicate.

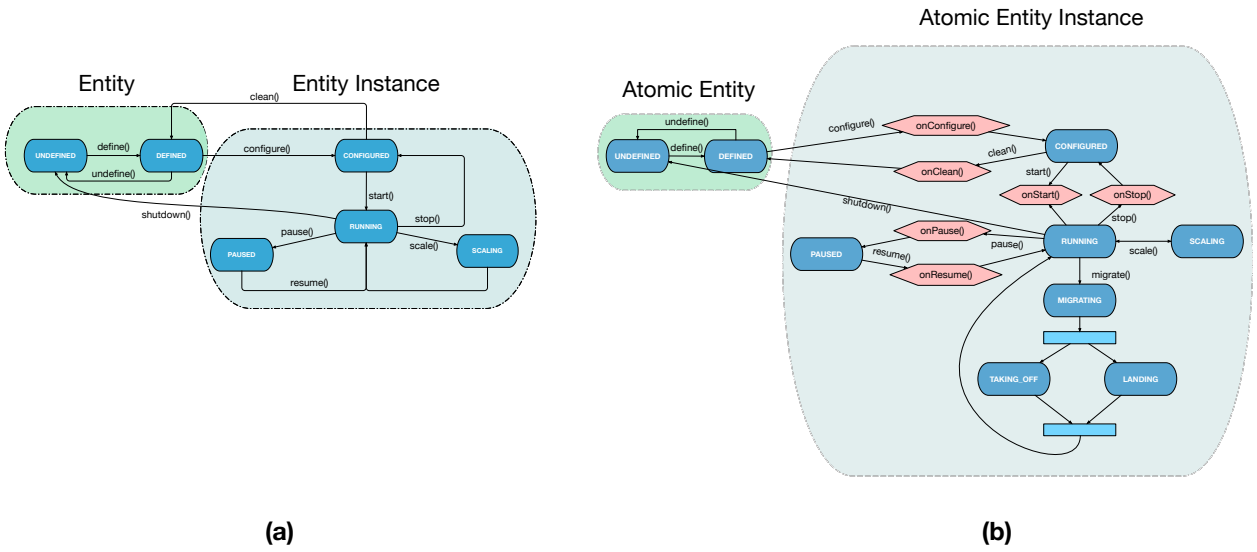

Fig. 2. Finite State Machine (FSM) for entities (a) and atomic entities (b).

Each resource in the system has conceptually two values, the actual value and the desired value. These values are maintained on two separate distributed key value stores. The actual state can only be changed by the owner of the resource, while the desired value is used to request changes to the actual value. This technique allow us to have concurrency only on the desired value of a resource, but never its actual actual value. As a single writer exist for the actual value it can take care of serializing writes on the information that represents the actual state of the system.

C. *fog*∅5 Distributed Key/Value Store

As mentioned in the previous section, everything in **fog**∅5 is a resource and all resources are stored in a distributed key value store. This store provides an eventual consistency semantics and has built-in versioning.

The store, as explained in more detail in next the sections, has a *root*, a *home* and a *cache capacity*. The *root* and the *home* are URI indicating respectively the scope of resolution for the store and the resources to be maintained in main memory. In other terms, a store will keep in main memory any resource that has *home* as a prefix, and will store on a fixed capacity cache resources whose prefix is the *root* but not the *home*. Resources that are not found in the store are resolved using a distributed cache miss protocol. A distributed cache coherency protocol is used to ensure eventual consistency of cached resources.

This abstraction allow to bound the amount of memory used by a node while at the same time access a virtual memory/storage that is distributed and potentially as bit as the sum of all memory/storage in the system. This abstraction along to being used at the core of **fog**∅5 is provided to the user as a way to virtualizing memory and storage end-to-end.

As a result this store can be used as a single and scalable abstraction across high-end and extremely constrained nodes.

The distributed store can also be used for sharing information that may be useful for other application, as an example veNB may share Radio Network information though the distributed store, thus allowing other applications and services to take decisions based on real-time data.

The store while providing a high-level abstraction has been implemented to have extremely low latency. As such it facilitates interaction between different services and allows for extremely short reaction times.

IV. FOG∅5 Distributed Architecture

The high-level architecture of **fog**∅5 is depicted in Figure 3. From Figure 3 it can be seen how **fog**∅5 has been designed to run as either a process on a traditional OS or as a trusted service inside a Trusted Execution Environment (TEE). Where the TEE could be running on an hypervisor – as depicted in Figure 3 – or on bare metal, ideally on a trust zone such as those supported by ARM processors. This deployment ensure that all communications with the external network are done through a run-time that has a very small surface of attack. Internal communication is performed through a VPN.

This means that only the software running on the TEE can communicate with the external network, which, as such mediates communication between the internal and external network. This architecture, while not compulsory – as mentioned above – was motivated by the

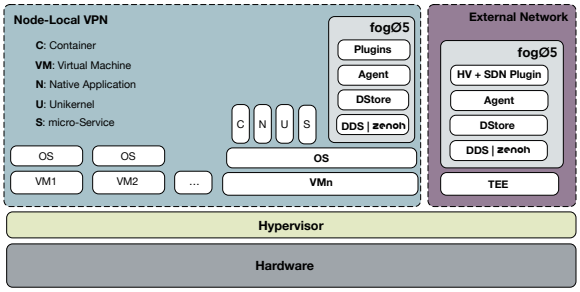

Fig. 3. **fog**∅5 high-level architecture.

fact that regular operating systems, such as Linux and Windows, have an extremely large surface of attack, which makes them extremely hard to secure. Finally, the **fog**∅5 instance running on the TEE has only the plugins for managing the network and the hypervisor, along with a trusted implementation of the distributed store.

A. *fog*∅5 *Agent*

The core component of **fog**∅5 is the **agent**. The agent represents a manageable resource on the system, which from now on we'll call a **fog**∅5 node. Each agent has two stores, one representing the actual state of the node and the other representing the desired state for the node. The actual state of a node can only be written by the agent running on that node, while the desired state can be written by anybody to cause state transitions, such as provisioning a VM, or a binary executable, and so on.

Agents dynamically discover each other by leveraging the dynamic discovery provided by the distributed store, which in turn leverage the dynamic discovery provided by the data sharing layer which may be implemented via Data Distribution Service (DDS) [1] or zenoh [2].

The set of functionalities supported by an agent is controlled by plug-ins. Different class of plug-ins exist for entities, network, OS, and so on. The agent "*simply*" orchestrates the plug-ins state transitions.

B. *fog*∅5 *Plugins*

As mentioned above, **fog**∅5 leverages plugins to manage just about anything. If we take as an example the plug-in for *atomic entities*, these are responsible for implementing the state transition depicted in Figure 2 for a specific kind of deployment, unit, such as a Robotic Operating System (ROS) nodelet, a unikernel, a binary executable, a container or a VM. As another example, operating system plugin, provides an implementation of the primitives required by **fog**∅5 to operate on the given OS.

C. *fog*∅5 *Distributed Store*

fog∅5's distributed store, is a distributed $< key, Value >$ store whose protocol has been implemented leveraging DDS or alternatively **zenoh**. Both DDS and zenoh, promote decentralized architectures and do not require a broker. Additionally, they have extremely low latency, high throughput and excellent scalability.

fog∅5's distributed store exposed the following simple but extremely poweful API:

- keys(path) list the keys under the given path.
- get(k) get the resource for the given key, resolving a miss if necessary.
- put(k,v) store the information v using k as key.
- dput(k,[v]) update the information for the key.
- pput(k,v) persistent put, these stores the ¡k,v¿ on a stable storage.
- getAll(k) same as get(k) but k contains wildcards, resolving a miss if necessary.
- remove(k) remove the information associated to the key
- observe(k, action) register an observer for the specified key, each time the value is update the action is called
- reg_metares(k, action) registers a meta-resource and the action that resolves it.
- unobserve(k) remove the registered observer.

The store also supports a versioning system to allow for (1) rollbacks in case of errors, (2) node with intermittent connectivity to be updated with latest version, and (3) dealing with cache miss and eventual consistency.

Finally, it should be remarked that registering an observer for a URI owned by another node (store) provide an elegant way of monitoring resources.

D. *fog*∅5 *and Fog Computing Requirements*

Now that we have provided an overview of **fog**∅5, it is time to review how it addresses the key fog-computing requirements listed in Table I:

- **Dynamic Discovery**. **fog**∅5 dynamic discovery is provided by the distributed store, which in in turns, leverages the dynamic discovery provided by **DDS** or by **zenoh**. This dynamic discovery is implemented by decentralized algorithms that do not introduce dependencies on servers and facilitate zero-conf deployments.
- **HW and I/O Discovery**. **fog**∅5 provides information on any kind of I/O and hardware accelerators available on the node. This information is managed by plugins and can be extended to provide access to standard API for dealing with embedded hardware and I/O such as the SGET eAPI.
- **Decentralized Architecture**. **fog**∅5 has a decentralized architecture as a consequence of the decentralized key/value store used at its foundation.
- **Low Latency**. As the distributed key/value store is implemented using either DDS or zenoh, the latency

for distributing data on a LAN, is around 50 microseconds when using DDS and around 20 microseconds when using zenoh.
- **Low Resource Utilization**. The **fog**$\varnothing$5 agent takes around 3MB of memory on a 64-bit machine running Linux. This is for a fully functional agent with no additional external dependency. Notice also that, as the architecture is decentralized, there is nothing else beside the **fog**$\varnothing$5 agent that needs to be deployed.
- **Generalized Constraint-based Deployment**. **fog**$\varnothing$5 uses a plug-in for allocation. The default, plugin takes into account computational resources, networking, I/O accelerators as well as affinities with respect to other entities.
- **Device Assignments**. **fog**$\varnothing$5 can assign any kind of device or I/O, such as a GPIO to an atomic entity.
- **Support for Constrained Devices**. The term constrained devices may mean different things to different people. It should be clear from the number provided above that something like a Raspberry Pi is not a constrained target for us. We are able to target ARM-M and even less capable hardware, such as micro-controllers.
- **End-to-end Unification of Computing, Networking and Storage**. As **fog**$\varnothing$5 can run efficiently on high-end hardware as well as constrained things, it can manage the compute, storage and networking fabric end-to-end. Additionally, **fog**$\varnothing$5 provides with the ability to configure end-to-end networks that span across cloud, edge and things.
- **Large Scale Deployment**. Thanks to the dynamic discovery, large scale deployments are is simple and effective. The scalability of these deployments stems for the use of DDS and its dedicated infrastructure for large scale-deployments.
- **Extensibility**. **fog**$\varnothing$5 has a plugin architecture that allows for any aspect to be extended and customized.
- **Support for Real-Time OS**. At the present time, **fog**$\varnothing$5 can provision real-time OS and we are actively working on supporting TSN provisioning and management.

V. FOG05 REAL-WORLD USE CASES

fog$\varnothing$5 is currently being used in R&D laboratories around the world, such as ITRI, UC3M, III and NCTU, for applications ranging from 5G VNF in heterogeneous environments to deployment and orchestration of complex services inside a smart factory to enable on-demand manufacturing.

In these applications **fog**$\varnothing$5 is deployed on extremely different hardware platforms spanning from servers to Raspberry Pis and lego mindstorms!

We are also starting to work on service provisioning for V2X or V2V communication and collisions avoidance among swarm of vehicles. In general, **fog**$\varnothing$5 has proven quite effective in enabling resource virtualisation and harvesting across a wide range of devices.

In the reminder of this section we describe a use case that has been driven by our robotics business unit.

A. Connected Robots

In the near future, *general purpose* robots will be used in factories to accomplish a wide range of tasks through reconfiguration achieved by simply *deploying* a different control logic. Flexible provisioning of robot tasks, within a swarm of robots, requires dynamic discovery of available robots along with the discovery of their computational, I/O and manipulation capabilities.

Fog computing infrastructures matches well the requirements of next generation robotic applications with respect to their needs for dynamic provisioning, computational harvesting, minimal communication latency and jitter.

B. *fog*$\varnothing$5 in Robotics Applications

fog$\varnothing$5 is used today to provision, deploy and manage the components of a complex robotic control application, and dynamiucally decide the most suitable target among those spanning from the cloud to robot. For instance, **fog**$\varnothing$5 may decide to deploy analytics on the cloud or on edge nodes, while the control logic may deployed and migrated so to remain close the robot or perhaps directly on the robot.

As an example, one of our Business Units is using **fog**$\varnothing$5 for provisioning, deploying and managing the network as well as the applications required in swarm robotics applications. In one of these applications, a robot equipped with a stereoscopic camera, has to follow another robot. The logic controlling the *follower* robot is deployed on edge servers and live migrated as the robot moves to minimize network latency and jitter.

In this specific a scenario the hardware platform is composed by:

- Two Edge Servers, one of which has a GPU accelerator
- One RaspberryPi3 with an extra 802.11ac NIC
- One Robot using Robotic Operating System v2 (ROS2), ADLINK Neuron board, a stereoscopic camera, a set of motors to more around
- One Robot using ROS2, ADLINK Neuron board, a set of motors to move around

All robots have some computational power, multiple NICs and I/O interfaces. Each device runs the

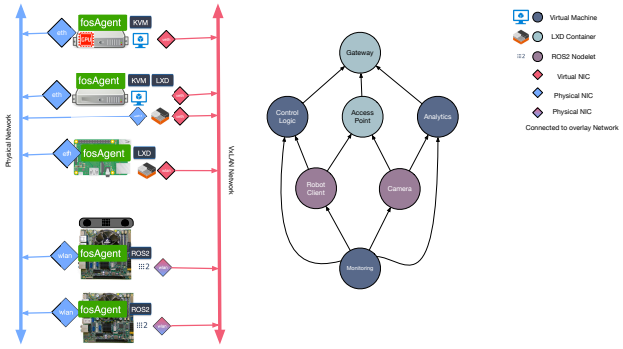

Fig. 4. Connected Robots Use Case

fog∅5 agent and has a NIC connected to the management.

The application is composed by:

- Complex Robot control application
- Analytics application
- Virtual Access Point VNF
- Gateway VNF
- Image recognition client
- Simple robot control logic
- Simple robot client
- Monitoring application

Each entity has different requirements in terms of I/O, compute, accelerators and networking requirements.

The application components are extremely heterogeneous. The analytics, monitoring, and robot controlling applications are packaged in VMs, the gateway VNF and the Access Point VNF are LXD containers, the other are ROS2 Nodelets

fog∅5 greatly facilitates the provisioning, deployment and management tasks as it dynamically discovers available nodes and deploys the components of this complex application in the right place – where the right place depends on resources and affinity. To maintain affinity between some entities it also executes live migration of some components.

Additionally, **fog**∅5 is also used to create an overlay network used by all components on the application.

VI. Conclusion and Future Works

This paper has provided an overview on the evolution of fog computing and summarized the key requirements that fog computing platforms need to address. These requirements highlight how, the main stream approach of adapting cloud-derived technologies such as OpenStack fall short in addressing the most important characteristics of fog computing environments.

The paper has thus introduced **fog**∅5 a novel fog computing infrastructure designed bottom up to address the key requirements of fog computing. **fog**∅5 features a decentralized and plug-in-based architecture that allow for zero-conf deployment, low-latency, and extensibility.

Our future work will focus on extending the applicability of **fog**∅5 to micro-controllers and providing consolidation and placement algorithms that perform dynamic resource optimization.

Finally, **fog**∅5 is available as Open Source at `http//fog05.io`.

We also working with the OpenFog consortium to make **fog**∅5 the Open Source reference implementation for the OpenFog Consortium.

References

[1] The data distribution service, 2017.

[2] zenoh: The zero network overhead protocol, 2017.

[3] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[4] D. Bruneo, S. Distefano, F. Longo, G. Merlino, A. Puliafito, V. D'Amico, M. Sapienza, and G. Torrisi. Stack4things as a fog computing platform for smart city applications. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 848–853, April 2016.

[5] Angelo Corsaro. Cloudy, foggy and misty internet of things. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ICPE '16, pages 261–261, New York, NY, USA, 2016. ACM.

[6] S. K. Datta, C. Bonnet, and J. Haerri. Fog computing architecture to enable consumer centric internet of things services. In *2015 International Symposium on Consumer Electronics (ISCE)*, pages 1–2, June 2015.

[7] OpenFog Consortium Architecture Working Group. Openfog architecture overview. Technical report, Feb 2016.

[8] Adrien Lebre, Jonathan Pastor, Anthony Simonet, and Frédéric Desprez. Revising OpenStack to Operate Fog/Edge Computing infrastructures. In *IEEE International Conference on Cloud Engineering*, Vancouver, France, April 2017.

[9] et al. M. Yannuzzi. A new era for cities with fog computing. *IEEE Internet Computing*, 21(2):54–67, Mar.-Apr. 2017.

[10] J. S. Preden, K. Tammeme, A. Jantsch, M. Leier, A. Riid, and E. Calis. The benefits of self-awareness and attention in fog and mist computing. *Computer*, 48(7):37–45, July 2015.

[11] Seifeddine Nsaibi und Ludwig Leurs. Time sensitive networking. *atp edition*, 58(10):40–47, 2016.

[12] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, pages 37–42, New York, NY, USA, 2015. ACM.