# On a Fog Computing Platform Built on ARM Architectures by Docker Container Technology

Andreas Eiermann, Mathias Renner, Marcel Großmann,
and Udo R. Krieger(✉)

Fakultät WIAI, Otto-Friedrich-Universität,
An der Weberei 5, 96047 Bamberg, Germany
udo.krieger@ieee.org

**Abstract.** Fog computing constitutes currently a challenging effort to establish the concepts and services of cloud computing at the edge of converging wireless networks and wired high-speed backbones. We discuss the concepts of our fog computing platform HCL-BaFog. It is built on top of Hypriot Cluster Lab (HCL) which has been developed by the Hypriot Pirate Crew in recent years based on single board computers with an ARM architecture. It uses LINUX container technology as underlying open source platform that has been established by means of the rapidly evolving framework Docker. We present the design principles of our fog computing platform and discuss its different software components. To create clusters of fog cells subject to high-availability requirements and to provide failsafe data processing, we further summarize some performance results on the integration of the orchestration tools Docker Swarm Mode and Kubernetes on HCL and draw some conclusions regarding their suitability for fog computing.

**Keywords:** Fog computing · Hypriot Cluster Lab · ARM architecture · Container virtualization · Docker Swarm Mode · Kubernetes · High-availability

## 1 Introduction

In recent years cloud computing has established the versatile technical basis to deploy very rapidly new services and customer applications as effective, easily manageable, low cost entities by means of distributed data centers and software virtualization. The underlying service architectures such as platform as a service (PaaS), storage as a service (StaaS), or infrastructure as a service (IaaS) use virtual machines as foundation for effective service provisioning and modern communication infrastructures on top of powerful physical servers and software-defined networks.

Software virtualization provides the means to satisfy the scalability, performability, and quality-of-service as well as quality-of-experience requirements of modern applications, capturing fundamental redundancy, availability, reliability, and security issues.

Following traditional approaches, modern cloud services are hosted in large data centers that fulfill all customer needs. They allow, for instance, small enterprises to establish their own distributed service delivery by the powerful cloud infrastructure and adaptive service models.

However, the new requirements of a rapidly evolving Internet-of-Things (IoT) including smart cities, smart homes, new transport and e-health services is challenging this traditional approach of service deployment (cf. [3,4,10]). Integrating millions of small sensors, actuators, and analysis devices provides great challenges to classical cloud computing. Today the Internet-of-Things requires a fundamental transition of many commonly used paradigms towards new lightweight protocols and processing concepts. At the edge of a high-speed network, swarms of sensors require first of all a scalable system design. At this low level of the processing hierarchy it can be based on tiny devices with small computational power and limited memory. Here heavy virtual machines and energy hungry data center technology are not suitable for sensor networks running on top of small, energy efficient processing elements. In addition, sensor networks and related data analytics that will be employed in smart private and public environments like smart homes, e-health settings, or smart cities will benefit from high-availability concepts of cloud computing and fault tolerance which is essential for critical infrastructures and their related services.

Considering traditional cloud computing for IoT purposes, an approach based on distributed data centers in the cloud would endorse much unnecessary data communication and heavily load the core network. Following Azam and Huh [1], fog computing is one possible solution to relocate virtualized services from the cloud to the network edge. In this regard the new concept of fog computing currently constitutes a challenging effort to establish the well determined concepts and services of cloud computing such as IaaS and PaaS at the edge of wireless networks based on 5G technology with their next generation optical backbones towards the cloud (see Fig. 1).

In this paper we present the design of our fog computing platform HCL-BaFog built on top of Hypriot Cluster Lab (HCL[1]) which has been developed in recent years. It is realized by suitable concepts to satisfy the computational challenges of IoT and to support services and applications, which are needed to successfully manage large data streams and communication of a large number of connected devices. In this respect HCL is able to provide a virtualized computing basis at the edge since it is running on ARM architectures, ships energy efficient features by design and can behave similar to a small data center. HCL utilizes interconnected single board computers (SBC's) with ARM architecture and HypriotOS as operating system. The HCL software platform already offers basic cloud functionality based on LINUX virtualization while running efficiently on 32-bit or 64-bit multi-core ARM processors like Raspberry Pi or Pine A64. Thus, it provides a key factor to minimize cost. Due to the fact that those processors offer less computational power, services must be packed into lightweight containers.

[1] See URL: https://blog.hypriot.com/downloads/.

In the HCL approach they are built by means of the framework Docker to avoid the overhead of virtual machines.

In essence, HCL provides an energy efficient micro-cloud computing platform. To support the fast development of fog computing, the developed code basis of HCL is publicly available on Github[2].

In the following we first present the design principles of our HCL-based platform HCL-BaFog and discuss its utilization for the purposes of fog computing. We illustrate how it handles redundancy and replication of service tasks between several hosts at different locations via secured wide-area networking. To enable encrypted communication between distributed hosts at layer 2 and 3, we use an interconnection approach based on overlay meshes like virtual private LANs. Our platform can establish fault tolerant, reliable and secure connections among several independent Docker Engines and provide means to achieve the QoS and performability objectives of mirco-cloud services.

To guarantee high availability and to provide failsafe data processing, we further discuss how software-defined networking and the integration of orchestration tools like Docker Swarm Mode and Kubernetes can be realized. Then we illustrate some first performance results on their application and draw a few conclusions about their suitability regarding fog computing.

The paper is organized as follows. In Sect. 2 we describe the technical basis of Hypriot Cluster Lab and discuss the establishment of secure overlay meshes among Docker hosts. In Sect. 3 we elaborate on the resource management and orchestration processes that are used in a fog cell cluster of our test bed HCL-BaFog. Then we illustrate some performance results on the container orchestration by means of Docker Swarm Mode and discuss resilience issues in related fog cell clusters. Finally, we draw some conclusions on the suitability of containerization by Docker on ARM-based architectures in fog computing scenarios.

## 2 Architecture of a Fog Computing Platform

The development of our fog computing platform HCL-BaFog is based on the existing Hypriot Cluster Lab (HCL). Its basic implementation of a resource management framework and runtime libraries for 64-bit ARM platforms is a Debian-based operating system called HypriotOS, that has been first deployed in February 2014. The related installations of the container virtualization framework Docker including Docker Engine, Compose and Machine as well as Docker Swarm Mode by the Hypriot team in late 2015 have been the starting point of the development described here.

### 2.1 The Basic Technology of Hypriot Cluster Lab

The technical basis of Hypriot Cluster Lab is provided by single board computer (SBC) systems based on a 32-bit or 64-bit multi-core ARM processor architecture. The software design is built by means of a lightweight LINUX container

[2] See URL: https://github.com/hypriot.

virtualization technology. Both components can provide the required hardware and software functionality of a powerful, low cost fog computing framework that includes the services, middleware and protocol stacks to interconnect sensor or actuator devices and appliances of a modern Internet-of-Things (IoT) settings at the first aggregation level of the underlying distributed infrastructure (see Fig. 1, cf. [3]).

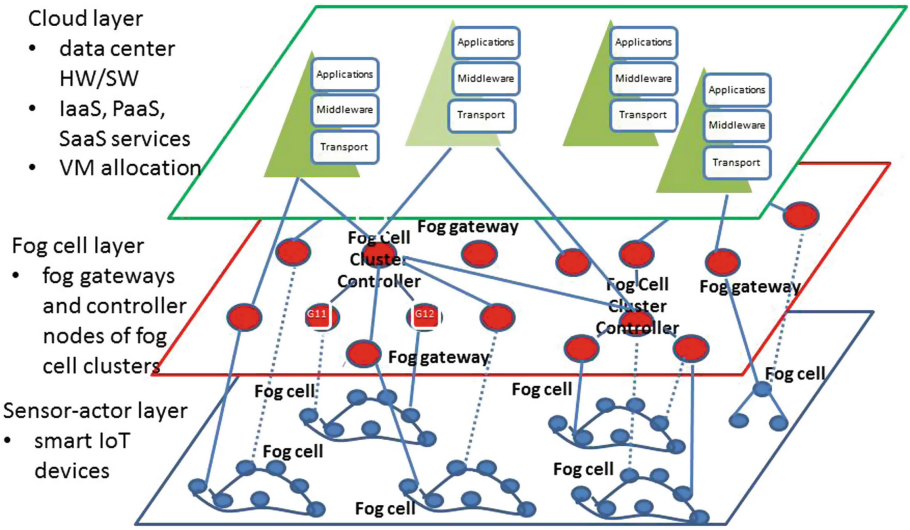


**Fig. 1.** Hierarchical layering of a fog computing system.

The sensor, actuator, and data sampling elements constitute the technical basis of data collection at layer 1 and distribute the information along redundant paths to the associated fog gateways. They are arranged in fog cells and controlled by more powerful cluster controller nodes that integrate software-defined networking, automatic deployment, as well as monitoring, orchestration, resource assignment, and data management functionality at a second level of the hierarchy. Putting the scalable SBC elements as first aggregation point or fog gateway at the edges of a fog cell cluster with its sensor and actuator hardware components, the virtualization offered by the container technology enables the deployment of a flexibly configurable software platform. It allows the designer to instantiate powerful processing derived from micro-cloud services in a IaaS or PaaS environment at the expense of very small latency and low cost. The architecture integrates the cell clusters by means of the secured internetworking functionality of the cluster controllers of fog cells and their interconnected gateways into the distant cloud infrastructure that offers IaaS, PaaS, SaaS and storage services at powerful data centers with their virtualized processing and storage components (cf. Fig. 1, see also [3,18]).

Hypriot Cluster Lab has adopted container virtualization since it allows the flexible realization of infrastructure and platform as a service architectures. HCL has realized the virtualization approach by means of the containerization framework Docker and its open standards running on top of all major infrastructure elements. Compared to the classical hypervisor approach or heavy LINUX container technology, a lightweight container virtualization is the appropriate design alternative for a resource constrained SBC platform like Raspberry Pi, Arduino or SolidRun'S more powerful ClearFog boards (see Fig. 2, cf. [5]). It allows to run processes efficiently at highest speed on those ARM boards without the overhead of a hypervisor using lightweight kernel namespaces derived from LINUX. The restriction caused by the binding of the container technology to the func-

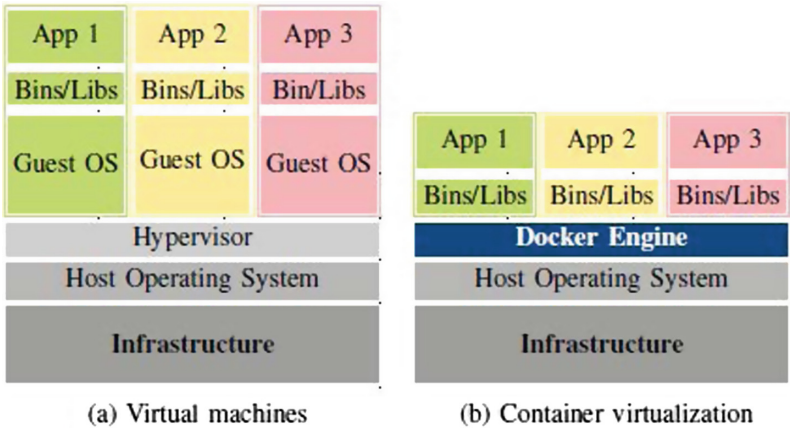


**Fig. 2.** Container virtualization (b) vs. a virtual machine environment (a).

tionality of the underlying host operating system HypriotOS is outperformed by the advantage of encapsulating all dependencies of an application into a single Docker image. Docker's capability regarding hierarchical layering and sharing of common images and the assignment to nodes, and the interconnection of containers using the clustering of Docker hosts by Docker Swarm Mode provides an effective way to enhance the required functionality of the edge computing nodes and to establish their interworking.

In our realized test bed we use Docker 1.12 released in June 2016. It has implemented all cluster management and orchestration features within the Docker Engine as main Docker binary. The Docker Engine consisting of a server daemon on a Docker host and a Docker client is the core of the ecosystem to create and manage containers as well as to build and manage local images. Then the Docker binary can be used in the 'Swarm Mode' which activates the cluster capabilities and allows to create the primary swarm master or leader node and all worker nodes on distributed hosts (see Fig. 3, cf. [13]).

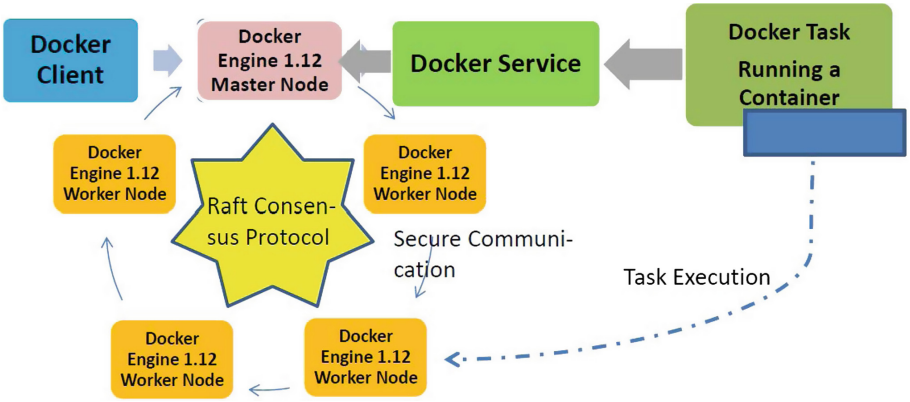


**Fig. 3.** Architecture of Docker Swarm Mode.

### 2.2 Building Secure Meshes of Fog Computing Nodes

The connection pattern of the initial Hypriot Cluster Lab platform was restricted to a local cluster paradigm derived from virtual private networking in a single domain. However, regarding fog computing scenarios it is necessary to interconnect the fog gateways as aggregation points of a fog cell above the sensor-actuator layer into a hierarchical structure, e.g. a tree or mesh of those cells (cf. Figs. 1, 4, see also [18]).

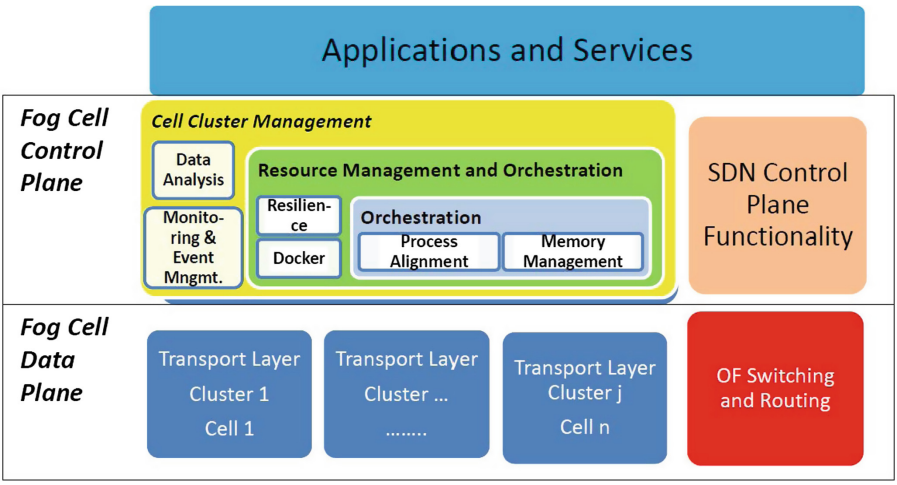


**Fig. 4.** Architecture of the fog computing platform.

In line with the structure of 4G and 5G networks we define the gateway as aggregation point represented by an ARM-based virtualized system as analogy to

an eNodeB that is governing the control, data, and management planes of a fog cell with its attached interconnected sensor and actuator elements. Analogous to routing areas and SAE gateways these cells are interconnected among each other by gateway nodes of fog cell clusters, e.g. more powerful SBC's including software-defined network elements like programmable switches and SDN controllers. As topologies we may consider trees or mesh networks interconnected by a stationary or mobile SDN based on WiFi, NarrowBand-IoT LTE, or 5G technology (cf. Fig. 1).

To realize the interconnection topologies among fog cells in a secure way, it is necessary to establish a virtual private network as overlays among ARM based virtualized master nodes in a fog cell. Therefore, we have provided an adaptive auto-configuration process to establish a secure mesh among these fog cell control nodes and their attached Docker hosts. It extends the basic VLAN communication paradigm of HCL to a fully operational VPN system.

For this purpose the HCL configuration concept has been enhanced and new functionality including secure tunneling has been adopted. It is based on the open source LINUX framework *Tinc* (see Fig. 5) and may be substituted by evolving improved Docker functionality in the near future.

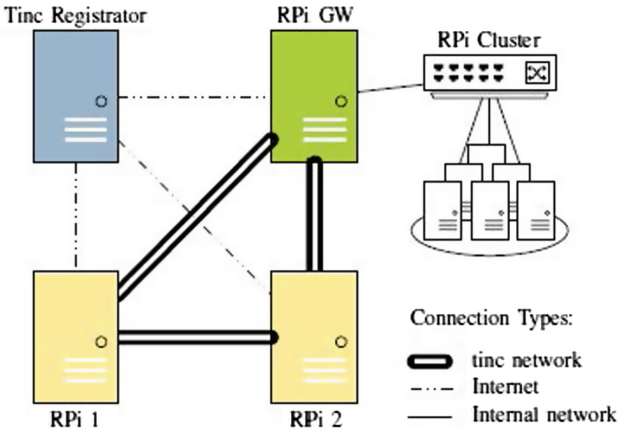


**Fig. 5.** Interconnection of fog cells.

The realized initialization and configuration management processes of the clustering service are adaptive and enable self-configuration. An IEEE 802.1Q VLAN mesh with a predefined ID for every node has been integrated in such a way that the existing network is not polluted by broadcast messages of the cluster. A DHCP server is configured, such that the IP addresses of nodes may be allocated dynamically. To initialize a cluster service running on top of the VLAN mesh a temporary fixed IP address of a host is used. If the service already exists, it can be recognized by a specific zeroconf notification service and the temporary IP address is replaced by a new one provided by DHCP.

If the address configuration is omitted, HCL will boot and directly use the tunnel interface. Then it reconfigures the Docker Engine to advertise the functionality of Docker Swarm Mode and initiates a Consul container on a master node. Subsequently, all other nodes join the existing Consul service by starting their own Consul containers. The master node acts as key-value store with a built-in DNS server. It is able to discover and track the liveliness of various services across a given infrastructure.

Further details on the used configuration scripts and some performance issues have been discussed in [9].

As outcome all nodes of a cluster connect with a Docker Swarm container and start another instance as a manager or replica of the cluster. The realized self-configuration management process guarantees that important components of the Docker framework such as Docker Compose, Consul, and Docker Swarm can be used to run user services and applications on a configured cluster. The feasibility of the approach has been demonstrated by a private, secured, and geographically distributed cluster running binned services across secured links (see [8,9] for more details).

Currently, the adaptation of OpenFlow switching using Open vSwitch[3] and SDN controllers to an ARM architecture and the integration of the basic *ovs-docker*[4] framework into the HCL stack is on the way and first results are available in an experimental prototype as indicated in Fig. 4.

## 3 Orchestration of Fog Computing Scenarios

In Subsect. 2.2 we have illustrated that fog computing requires in IoT scenarios the interworking of distinct fog cells and their management nodes, i.e. the cluster controllers. By these means layered architectures including trees and meshes of fog cells can be constructed and the interconnection of containers has to be realized. Therefore, the scalable, automatic deployment and efficient, fault tolerant operation of new services and customer applications in a lightweight virtualization environment requires tools for the clustering of Docker hosts, the deployment of containers, the monitoring and event management, as well as the powerful orchestration of processes.

In recent years several platforms offering cluster computing with LINUX containers have been developed mainly for powerful data centers, e.g. Apache Mesos[5] or Google's Kubernetes[6]. Docker has integrated Docker Swarm Mode as Engine clustering tool to enable similar orchestration features and to support container scheduling and management tasks across interconnected hosts (cf. [11, 13]). As HCL relies on those frameworks, it can enable the same functionality on cheap, energy efficient ARM-based hardware like RPi, Pine A64, or ClearFog SBC's.

[3] See URL: http://openvswitch.org/.
[4] See URL: https://github.com/openvswitch/ovs/blob/master/utilities/ovs-docker.
[5] See URL: http://mesos.apache.org/.
[6] See URL: http://kubernetes.io/.

### 3.1 Orchestration Tools for ARM Clusters

Clustering of hosts with 64-bit ARM architecture is a prerequisite for fog computing scenarios and has been studied in recent years (cf. [7,12,21]). To the best of our knowledge, in spring 2012 researchers at the University of Glasgow have started the first public project on cloud computing with Raspberry Pi 1, bundling 56 Raspberry Pi's and utilizing LXC[7] container virtualization (cf. [19,21]). After that many related projects followed (see e.g. [6,14,20]), while the largest cluster with 300 RPi's has been built at Bolzano University [2]. In mid-2015 researcher at the University of Glasgow started with in-depth experiments using Kubernetes and Docker on a cluster using the second version of Raspberry Pi's. Official support for Kubernetes on ARM has been performed by L. Kaldström's project 'Kubernetes on ARM'[8] among others (see [7,12,15,17]).

However, the performance of Kubernetes and Docker Swarm Mode has not been evaluated thoroughly by detailed measurements on those hardware platforms in any of the aforementioned experiments. The latter ended when all nodes were up and running. For instance, no experiment measured the performance characteristics of load balancing by a simple load test.

In addition, the first evidence of testing fault-tolerance capabilities of Kubernetes is provided by a talk of Tsang and Wassink at Devoxx, Belgium, in 2015[9]. Studies by Nissen and Jensen [16,17] provide the most versatile investigation about Kubernetes on Raspberry Pi's. Although they are the first to actually perform a load test on a RPi cluster, their presented measurements still don't cover other important resource consumption aspects.

In this regard our investigations on the suitability of Docker Swarm Mode and its versatile alternative Kubernetes on an ARM infrastructure clustered by HCL have been stimulated by the gaps of those studies and the evaluation is based on a hardware environment similar to that one used by Nissen and Jensen [17].

### 3.2 Performance Comparison of Container Orchestration

It is our objective to illustrate the comparison of major performance characteristics w.r.t. Docker Swarm Mode v1.12.2 and 1.12.3, respectively, with HypriotOS-v1.0.1[10] and the Kubernetes protocol stack on top of clustered Docker hosts with HypriotOS-v0.8 and interconnected Docker 1.11.1 containers. The latter suites have been used as basic orchestration tools for a scalable deployment and operations on ARM architectures based on the PicoCluster kit[11] due to code stability issues.

[7] See URL: https://linuxcontainers.org/.
[8] See URL: https://github.com/luxas/kubernetes-on-arm/releases, https://luxaslabs.com/.
[9] See URL: https://www.youtube.com/watch?v=kT1vmK0r184.
[10] See URL: https://blog.hypriot.com/downloads/.
[11] See URL: https://www.picocluster.com/collections/starter-picocluster-kits/products/pico-5-raspberry-pi-starter-kit?variant=29344698892.

Our test bed is provided by a cluster with 5 Raspberry Pi 3 Model B boards and SD cards Samsung EVO Plus, 32 GByte, with high performance of 20 MBps writing and 80 MBps reading speed. Hosts have been interconnected by a router with 100 Mbps port speed that can be utilized up to 94.4 % by RPi's and provides a DHCP service. Hosts synchronized by NTP were also interconnected to a desktop as external management and master control node with M/Monit as process monitoring server to store and visualize the performance data collected by Monit 5.11.0 clients at the RPi's (see Fig. 6).

The experiments were performed to validate whether the claims regarding the efficiency of container processing and memory access as well as the high-availability guarantees of the cluster orchestration tools Docker Swarm Mode and Kubernetes still hold in the resource constrained environment of a SBC cluster.

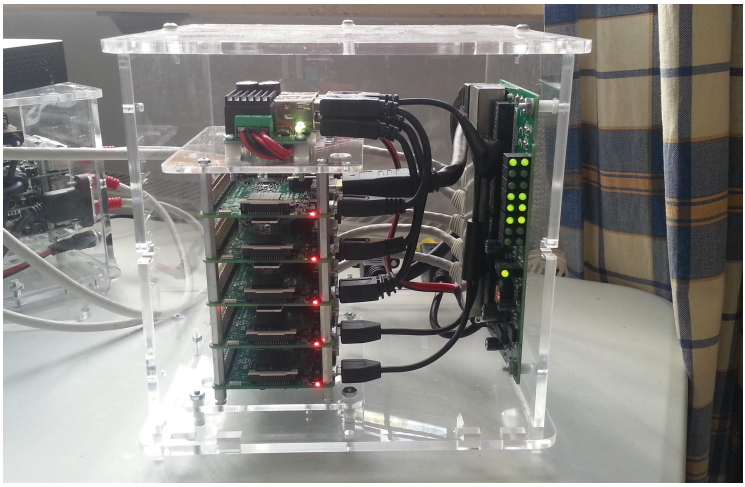

**Fig. 6.** A fog gateway based on a PicoCluster of RPi's controlling a cluster of fog cells.

The evaluation has considered the resource usage of the instances regarding both the manager and worker nodes of the interconnected containers hosted in a cluster as well as load balancing and resilience characteristics. Details of the used configuration scripts are publically available[12] to enable further validation tests.

In the first step, we have illustrated the exemplary resource consumption of native service clustering based on three different scenarios:

– Scenario 1 considers a fresh installation where only the service monitor *Monit* and the Docker daemon are running to get some ground-truth on the basic process load of Docker containers on the RPi's.

[12] See URL: https://blog.hypriot.com/post/high-availability-with-docker/.

– Scenario 2 represents a container cluster after starting the Docker Swarm Mode.
– Scenario 3 considers a cluster after starting Kubernetes as orchestration suite of the interconnected Docker containers.

In all scenarios the achieved performance results stemming from a sampling frequency of 1 Hz of the manager node (first entry) and worker nodes (second entry) have been recorded. Table 1 illustrates a typical outcome of a resource consumption test.

**Table 1.** Measurements of exemplary resource consumption by Docker and Kubernetes instances in our HCL-BaFog test bed.

| Sce. | CPU usage [%] Manager/-Worker | Memory usage [%] Manager/-Worker | Process CPU Docker [%] Manager/Worker | Process memory Docker [%] Manager/Worker |
|---|---|---|---|---|
| 1 | 0.44 | 4.21 | 0.05 | 2.47 |
| 2 | 1.43/0.83 | 7.87/7.4 | 1.1/0.44 | 57.45/2.42 |
| 3 | 14.74/2.01 | 34.21/12.68 | 5.76/1.31 | 479.51/160.77 |

As seen in Fig. 7 Docker Swarm Mode v1.12.2. is already able to balance the incoming traffic load very well among all worker nodes (node-black, node-green, node-blue, node-orange) while the master (node-red) is able to manage all aggregated traffic load.

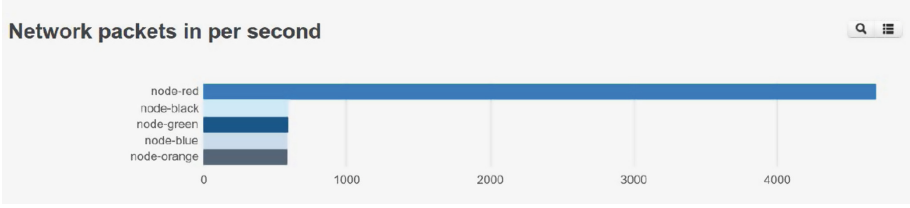


**Fig. 7.** Balancing incoming network traffic of the RPi cluster with Docker v1.12.2. (Color figure online)

Considering both type of nodes, one can recognize that, as expected, Kubernetes requires much more resources than Docker in Swarm Mode. Subtracting the realized consumption of scenario 1 from those of scenario 2 and 3, the resource consumption of Docker and Kubernetes can be isolated in terms of a resulting difference-in-difference model (see Table 2). Considering the CPU usage on the Raspberry Pi hosts in this exemplary cluster scenario, Kubernetes requires 14.4 and 4 times, respectively, more resources w.r.t. manager and worker nodes than

**Table 2.** Difference-in-difference model of an exemplary resource consumption.

| Sce. | CPU usage [%] Manager/Worker | Memory usage [%] Manager/-Worker | Process CPU Docker [%] Manager/Worker | Process memory Docker [%] Manager/Worker |
|---|---|---|---|---|
| 2 | 0.99/0.39 | 3.75/3.28 | 1.05/0.39 | 54.98/0 |
| 3 | 14.3/1.57 | 30.09/8.561 | 5.71/1.26 | 477.04/158.3 |

Docker in Swarm Mode. Regarding the memory usage, Kubernetes allocates 8 times respectively 2.61 times more memory than Docker Swarm Mode. In terms of process CPU seizure by Docker, Kubernetes requires 5.4 times respectively 3.23 times more CPU time than Docker Swarm Mode. Regarding process memory allocation, the resource consumption of Kubernetes is even worse. In summary, the prototypical experimental setup has revealed that Kubernetes requires on average 9.1 times more resources for the master node and 41.9 times more for worker nodes.

The reason is that the orchestration capabilities of Kubernetes are much more sophisticated and customizable compared with the current Docker Swarm Mode. Additionally, Kubernetes has a much richer feature set because it offers by default advanced operation services, such as service monitoring, centralized logging and a dashboard. The latter allows to control the cluster almost purely by the associated graphical user interface.

Thus, it is a challenging future development task to enhance the orchestration capabilities of Docker Swarm Mode while maintaining its efficient resource consumption on SBC clusters.

### 3.3 On the Performance of Resilience and Recovery Strategies

To cope with the high-availability and fault-tolerance requirements of IoT services in fog computing scenarios, we investigate the performance of the application and infrastructure level resilience of a service and the related recovery strategies implemented by the orchestration, scheduling and deployment procedures of Docker Swarm Mode.

The study has been inspired by Nissen and Jensen [16] who have evaluated the resilience at application and infrastructure levels of cloud computing on an ARM platform using Kubernetes. Here their infrastructure resilience experiment based on a simple Docker service has been repeated in the cluster environment of our test bed HCL-BaFog. As exemplary service of the resilience test a HTTP server is used by the swarm manager node acting as leader. When the Docker Engine is operated in Swarm Mode, a corresponding Docker service can be created to deploy the related image of the replicated HTTP application among the clustered Docker Engines of the swarm. The swarm manager schedules the service tasks onto the interconnected hosts in the swarm as one or more replica tasks which are the atomic scheduling entities. The latter instantiates a container for each

task which is created and allocated by the orchestration function to an available worker node of the swarm. All tasks are then running independently of each other on the available nodes. The orchestration function performs permanent health checking of the running containers in the cluster to guarantee the level of replicas and creates a new replica task that spawns a new container at some node in case of a node crash.

The number of replicas of the container is a major tunable parameter for the level of fault tolerance provided to the service by the fog computing platform. This number of replicas in the cluster affects under varying processing loads the recovery time of a fault-tolerant distributed application running under Docker Swarm Mode and determines its response time to a requestor outside the cluster.

Therefore, a cluster recovery after a crash of a single node has been emulated with a MTBF of 5 min by the following small-scale availability test procedure of a replicated web service:

1. Docker runs a web service that is reachable from an external device, i.e. the desktop computer.
2. Variable background load is generated in terms of HTTP GET requests to the cluster using the HTTP benchmark and stress testing tool *Vegeta*[13].
3. To test the fault tolerance and recovery capabilities after a single point failure, a stopping of all processes related to the Docker service with a prescribed level of task replication and a hard rebooting of the whole cluster is performed. The following replication scenarios of the Docker service have been covered:
   (a) The Docker service runs with 1 replica.
   (b) The Docker service runs with 2 replicas.
   (c) The Docker service runs with 5 replicas.
   (d) The Docker service runs with 10 replicas.
4. During step 3, we have permanently measured the request rate processed by the cluster per seconds, the number of activated replicas, and the latency of the service response.

For comparison the results of similar experiments on a Kubernetes cluster have been adopted from Nissen and Jensen [16, Sec. 8.2, Fig. 8.5, p. 81]. The exemplary outcome of the recovery test by the Docker framework on our test bed is illustrated in Fig. 8. It is recognized that there is a sharp breaking point in the latency following a M/G/N response time pattern, e.g. beyond a request rate of 900 for $N = 10$ replicas. Then the load balancing after recovery reaches a high utilization level during the redistribution and restart of replicated containers. The improvement of the service robustness due to an increasing number of replicas constitutes a positive outcome of the applied resilience strategy. Regarding the given latency threshold of 50 ms, we realize, however, that Docker Swarm Mode has still a lot of potential for an improvement beyond its current low recovery rates.

[13] URL: https://github.com/tsenart/vegeta, https://hub.docker.com/r/mistobaan/vegeta/.

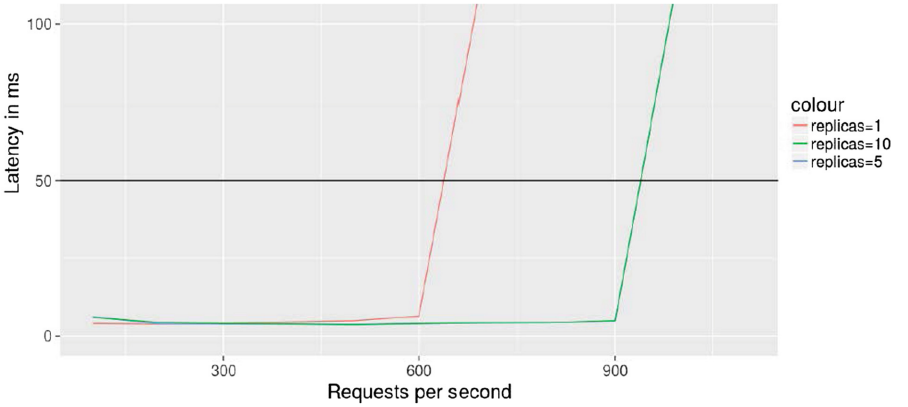


**Fig. 8.** Latency of the service response for Docker Swarm Mode in a recovery test.

Considering the performance of Kubernetes and Docker in Swarm Mode on more powerful systems with Intel or AMD designs or other low-cost multi-kernel ARM platforms, e.g. ONDROID-XU4 with its Octa core processor and 2 GByte RAM, of course more extensive studies are required to identify an optimal solution w.r.t the large variety of different scenarios arising from exciting IoT applications. They can provide the basis for a broader comparison on the scalability and suitability of the processing and memory units of the used hardware platforms. Regarding a monitoring and alerting application of fog computing by means of a HCL-like Raspberry Pi platform in the library depots of University of Bamberg, our current investigation already indicates that the microSD storage units may become a dependability bottleneck. In conclusion, extensive investigations of those performability issues will represent an important topic of our future research.

## 4 Conclusions

At present mobile edge and fog computing constitute a challenging effort to establish the distributed processing concepts and services of cloud computing at the edge of converging wireless networks and wired high-speed backbones. The need to integrate advanced transport and computing technology will increase with the rapid deployment of technology from 5G mobile networks and the fast evolution of Internet-of-Things including diverse applications from different domains such as smart energy, smart transport, smart city, smart home, and e-health.

In our paper we have discussed the concepts of our fog computing platform HCL-BaFog derived from Hypriot Cluster Lab (HCL). The latter has been developed for a resource constrained 64-bit ARM architecture, like Raspberry Pi's or ClearFog single board computers, and includes software-defined networking and lightweight LINUX container virtualization based on the open source framework

Docker. We have presented the design principles of the fog computing platform and discussed its usage to create secure overlays among Docker containers.

To guarantee high-availability features and to provide failsafe data processing, we have also elaborated on the integration of Docker Swarm Mode and Kubernetes as basic orchestration tools and drawn some conclusions on their suitability regarding fog computing on resource constrained hardware platforms.

Our results have revealed the feasibility of our approach to deploy and orchestrate automatically a small, energy efficient virtualized edge cloud on cheap commodity hardware. Regarding the cluster architecture, up to now HCL-BaFog's single point of failure is given by the master (or leader) node managing a Docker Swarm Mode. Thus, our future work will focus on fast, secure failover mechanisms based on the recently published resilience mechanisms of Docker Swarm Mode and its enhancement to scalable master-slave scenarios. Regarding the automation of a SDN deployment and traffic routing across Docker containers, further extensions of the ovs-docker framework are required on the ARM architecture such that an application can connect to the Docker REST API and start a new container with a specialized service, e.g. a caching proxy, while the container is attached to an ovs instance.

**Acknowledgment.** The authors are very much indebted to those members of the Hypriot Pirate team outside the University of Bamberg, including Govinda Fichtner, Dieter Reuter, and Stefan Scherer, that has developed the HCL platform during spare time and that guarantees its overwhelming success by enormous personal efforts.

## References

1. Aazam, M., Huh, E.-N.: Fog computing and smart gateway based communication for cloud of things. In: International Conference on Future Internet of Things and Cloud (FiCloud) 2014, pp. 464–470, August 2014
2. Abrahamsson, P., et al.: Affordable and energy-efficient cloud computing clusters: the Bolzano Raspberry Pi cloud cluster experiment. In: IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom) 2013, vol. 2, pp. 170–175. IEEE (2013)
3. Al-Fuqaha, A., et al.: Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutor. **17**(4), 2347–2376 (2015). (Fourth Quarter)
4. Atzori, L., et al.: Internet of things: a survey. Comput. Netw. **54**, 2787–2805 (2010)
5. Docker: What Is Docker? Comparing Containers and Virtual Machines. https://www.docker.com/what-docker#/VM. Accessed 04 Oct 2016
6. GCHQ: GCHQ's Raspberry Pi 'Bramble' - exploring the future of computing, 11. https://www.gchq.gov.uk/news-article/gchqs-raspberry-pi-bramble-exploring-future-computing. Accessed 14 Oct 2016
7. Goasguen, S.: Running Kubernetes on a Raspberry Pi, 16. http://sebgoa.blogspot.de/2015/09/running-kubernetes-on-raspberry-pi.html. Accessed 14 Oct 2016
8. Großmann, M., Eiermann, A., Renner, M.: Hypriot cluster lab: an ARM-powered cloud solution utilizing docker. In: 23rd International Conference on Telecommunications (ICT 2016), pp. 16–18, Thessaloniki, Greece, May 2016

9. Großmann, M., Eiermann, A.: Automated establishment of a secured network for providing a distributed container cluster. In: 28th International Teletraffic Congress (ITC28), 13–15 September 2016, Würzburg, Germany (2016)
10. Gubbi, J., et al.: Internet of Things (IoT): a vision, architectural elements, and future directions. Future Gener. Comput. Syst. **29**, 1645–1660 (2013)
11. Holla, S.: Orchestrating Docker. Packt Publishing Ltd., Birmingham (2015)
12. Huß, R.: A Raspberry Pi 3 Kubernetes Cluster, 27. https://ro14nd.de/kubernetes-on-raspberry-pi3. Accessed 14 Oct 2016
13. Kaewkasi, C.: Docker Swarm Mode, 2016. https://medium.com/@chanwit/docker-swarm-mode-fde1e3e392ae#p7w8sxhac. Accessed 14 Oct 2016
14. Kiepert, J.: Creating a Raspberry Pi-based Beowulf Cluster, Boise State University, 22 May 2013. Accessed 14 Oct 2016
15. Mogren, L.: Kubernetes on ARM (2016). Accessed 14 Oct 2016
16. Nissen, K., Jensen, M.: Kubecloud - a small-scale tangible cloud computing environment. Master's thesis, Aarhus University - Department of Engineering, 6 June (2016). http://kubecloud.io/files/kubecloud.pdf
17. Nissen, K., Jensen, M.: Setting up a Kubernetes on ARM cluster, 13. http://kubecloud.io/kubernetes-on-arm-cluster/. Accessed 14 Oct 2016
18. Skarlat, O., et al.: Resource provisioning for IoT services in the fog. In: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA 2016), 4–6 November 2016. Macau, China (2016)
19. Tso, F.P., et al.: The Glasgow Raspberry Pi Cloud: a scale model for cloud computing infrastructures. In: 2013 IEEE 33rd International Conference on Distributed Computing Systems (ICDCS) Workshops, IEEE, pp. 108–112 (2013)
20. University of Southampton. Southampton engineers a Raspberry Pi Supercomputer, 11 September 2012. https://www.southampton.ac.uk/sjc/raspberrypi/Raspberry_Pi_supercomputer_11Sept2012.pdf. Accessed 14 Oct 2016
21. White, D.: Building a Raspberry Pi cloud (2014). Accessed 14 Oct 2016