A Performance Exploration of Architectural Options for a Middleware for Decentralised Lightweight Edge Cloud Architectures

David von Leon, Lorenzo Miori, Julian Sanin, Nabil El Ioini, Sven Helmer and Claus Pahl
Free University of Bozen-Bolzano, 39100 Bolzano, Italy

Keywords: Internet-of-Things, Edge Cloud, Container, Middleware, Orchestration, Single-board Computer.

Abstract: The integration of Cloud and IoT (Internet-of-Things) resulting in so-called edge clouds has started. This requires the combination of data centre management technologies with much more constrained devices. Lightweight virtualisation solutions such as containerisation can be used to distribute, deploy and manage edge cloud applications on clusters. Leightweightness also applies to the devices, where we focus here on small-board devices such as Raspberry Pis in our concrete case. These small-board devices are particularly useful in situations where a mix of robustness due to environmental conditions and low costs is required. We discuss different architectural solutions for the distribution of computation to edge cloud devices based on containers and other management approaches and evaluate these in terms of cost, power consumption and performance.

1 INTRODUCTION

Cloud technology is moving towards multi-cloud environments with the inclusion of various devices and sensors. Cloud and IoT integration resulting in so-called edge cloud and fog computing has started (Chandra et al., 2013). This requires the combination of data centre management technologies with much more constrained devices, but still using virtualised solutions to deal with scalability, flexibility and multi-tenancy concerns.

Lightweight virtualisation solutions do exist for this architectural setting with smaller, but still virtualised devices to provide application and platform technology as services. Containerisation is a solution component for lightweight virtualisation (Pahl, 2015). Containers furthermore address platform concerns relevant for Platform-as-a-Service (PaaS) clouds like application packaging and orchestration.

We will compare a container middleware platform for edge cloud computing with other platforms (own-build or OpenStack based) that all use small-board devices for low-cost, robust settings.

We will discuss these architectural options for edge cloud middleware. For edge clouds, application and service orchestration can help to manage and orches-trate applications through containers (Pahl & Lee, 2015; Pahl et al., 2016). In this way, computation can be brought to the edge of the cloud, rather than data from the Internet-of-Things (IoT) to the cloud (Kratzke, 2014), thus increasing performance and security by not transferring large amounts of data.

A key constraint of edge cloud settings, in particular in the vicinity of sensors in the IoT and cyber-physical systems (CPS) space are resource constraints in terms of computing power, storage capability, reliable connectivity or power supply.

We address this constrained environment by focusing on small single board computers as the deployment platform of the container solution (Abrahamsson et al., 2013). We specifically focus on clusters of Raspberry Pi devices, see (https://www.raspberrypi.org/). Local cluster management is important to manage for example a number of locally distributed sensors.

We show that edge cloud requirements such as cost-efficiency, low power consumption, and robustness can be met by implementing container and cluster technology on single-board devices like Raspberry Pis (Miori, 2014; Sanin, 2016; von Leon, 2016). This architecture can facilitate applications through distributed multi-cloud platforms built from a range of nodes from data centres to small devices, which we refer to here as edge cloud. Other architectural options are less suitable.

von Leon, D., Miori, L., Sanin, J., El Ioini, N., Helmer, S. and Pahl, C.
A Performance Exploration of Architectural Options for a Middleware for Decentralised Lightweight Edge Cloud Architectures.
DOI: 10.5220/0006677400730084
In *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018)*, pages 73-84
ISBN: 978-989-758-296-7

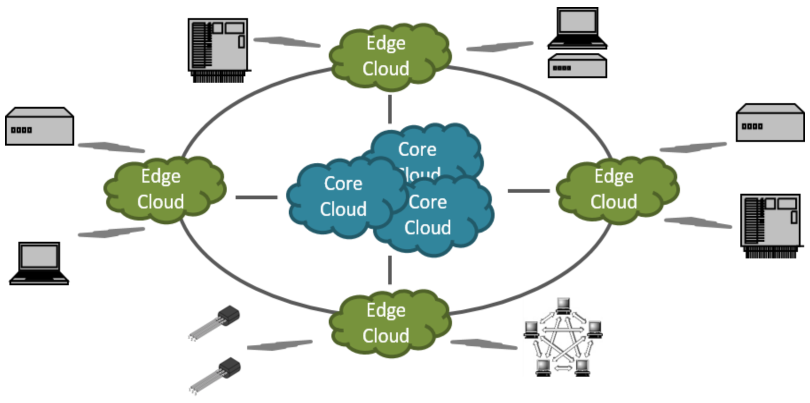

Figure 1: Edge Cloud Architecture.

Our objective here is to discuss three different architectural scenarios in terms of their implementation and experimental evaluation results, covering installation, power consumption, cost and performance concerns.

Our contribution will be organised as follows. We determine requirements and review technologies and architectures for edge cloud computing. We discuss the key ingredients of edge cloud architectures based on containers as the packaging and distribution mechanism. We specifically discuss storage, orchestration and cluster management for distributed Raspberry Pi clusters in edge cloud environments. We report on experimental results with Raspberry Pi clusters to validate the proposed architectural solution. The settings included are:

– Own-build storage and cluster orchestration
– Openstack storage
– Docker container orchestration
– IoT/sensor integration

We pay particular attention to practical concerns such as installation and management efforts, since RPi edge clusters are meant to be run in remote areas without expert support.

2 DETERMINATION OF REQUIREMENTS

We determine requirements and review technologies and architectures for edge cloud computing.

2.1 Edge Cloud Architectures

Edge computing mechanisms are needed for both computation and storage to address data collection, its pre-processing and further distribution. These edge resources could be dedicated (possibly smaller) resources spread across distributed networks. In order to support edge cloud architectures, we need the following features:

– location-awareness and computation placement,
– management services for data storage, replication, recovery.

Virtualised resources can support edge cloud architectures (Manzalini, 2014). Due to smaller device sizes, this can result in different resource restrictions, which in turn requires some form of lightweightness of the virtualisation technique (Zhu, 2013). In edge cloud architectures with integrated IoT objects, we need compute and storage resources used by applications and managed by platform services, i.e., packaged, deployed and orchestrated (Figure 1). Even for the network, virtualisation capacity is required as well (cf., recent work on software-defined networks (SDNs). Thus, we need to support data transfer between virtualised resources and to provide compute, storage, and network resources between end devices and traditional data centres.

Concrete requirements arising are location awareness, low latency and software mobility support to manage cloud end points with rich (virtualised) services. This type of virtualised infrastructure might provide end-user access and IoT links - through possibly private, edge clouds. These are technically micro-clouds, providing different services, but on a small scale (Helmer et al., 2016).

These need to be configured and updated - this particularly applies to service management. We also need a development layer to provision and manage applications on these infrastructures. Solutions here could comprise common topology patterns, controlling application lifecycles, and an easy-to-use API. We need to find the right abstraction level for edge cloud management at a typical PaaS layer.

2.2 A Use Case

We motivate our approach with a use case taken from our local region: modern ski resorts operate extensive IoT-cloud infrastructures. Sensors gather a variety of data:

– weather: air temperature/humidity, sun intensity
– snow: quality (snow humidity, temperature)
– people: location and numbers

With the combination of these data sources, two sample functions can be enabled:

– People management: through mobile phone apps, skiers can get recommendations regarding snow quality and possible overcrowding at lifts and on slopes. A mobile phone app can use the cloud as an intermediary to receive data from, but the performance of the architecture would benefit from data pre-processing at sensor location to reduce the data traffic into the cloud.
– Snow management: snow groomers (snow cats) are heavy-duty vehicles that rely on sensor data (ranging from tilt sensors in the vehicle and GPS location all the way to snow properties) to provide an economic solution in terms of time needed for the preparation of slopes, while at the same time allowing a near-optimal distribution of snow. This is a real-time system where cloud-based computation is not feasible (due to unavailability of suitable connectivity) and thus local processing of data is required for all data collection, analysis and reaction.

As we can see, performance of the architecture is a critical concern (Heinrich et al., 2017; Pahl et al., 2018) that can be alleviated by more local computation, avoiding high volumes of data to be transferred into centralised clouds. Local processing of data, particularly for the snow management where data sources and actions resulting through the snow groomers happen in the same place, is beneficial, but needs to be facilitated through robust technologies that can operate in remote areas under difficult environmental conditions.

Clusters of single-board computers such as Raspberry Pis are a suitable, robust technology. The architecture is dynamic as only necessary components (containers) should remain on local devices. For instance, a sensor responsible for people management during daytime could support snow management during the night.

Furthermore, the solution would benefit from flexible platform management with different platform and application services deployed at different times in different locations. Containers can help here, but need to be supported by advanced orchestration support.

To illustrate this, two orchestration patterns emerge:

– data pre-processing for people management: reducing data volume in transfer to the cloud is the aim. Analytics services packaged as containers that filter and aggregate data need to be deployed on selected edge nodes.
– fully localised processing in clusters (organised around individual slopes with their profile): full computation on board and locally between snow groomers is required, facilitated by the deployment of analysis, but also decision making and actuation features, all as containers.

2.3 Edge Cloud Architectures

We discuss the key ingredients of an edge cloud architecture based on containers as the packaging and distribution mechanism. A number of concerns needs to be addressed:

– Application construction
– Application orchestration
– Resource scheduling
– Distributed systems services
– Programming model
– (Edge) cloud-native architecture

This requires a combination of lightweight technology platforms – single-board devices as lightweight hardware combined with containers as a lightweight software platform. Container technologies can take care of the application management based on constrained resources in a distributed, clustered edge computing context.

3 LIGHTWEIGHT EDGE CLOUD CLUSTERS

3.1 Raspberry Pi Clusters

We specifically discuss orchestration for distributed Raspberry Pi clusters in edge cloud environments. We were inspired to implement our edge cloud architecture on Raspberry Pi clusters by previous work showing that clusters consisting of 300 or more RPis can be built (Abrahamsson et al., 2013). These small single-board computers create both opportunities and challenges. A Raspberry Pi (RPi) is relatively cheap (they cost around 30$) and has a low power consumption, which makes it possible to create an affordable and energy-efficient cluster suitable for demanding environments for which high-tech installations are not feasible. Since a single RPi lacks computing power, in general we cannot run computa-

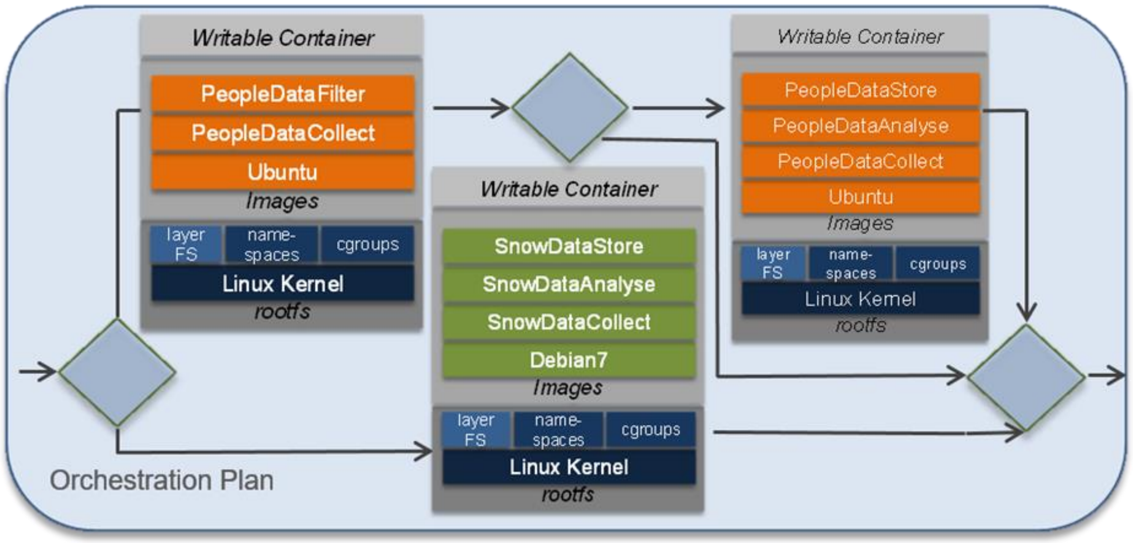

Figure 2: Simplified Container Orchestration Plan for the Case Study.

tionally intensive software on it. Nevertheless, this drawback can be remedied (to a certain degree) by combining a larger number into a cluster. This also allows the creation of differently configured and customised, and at the same time robust, platforms.

Creating and managing clusters are typical PaaS functions, including setting up and configuring hardware and system software, or to monitoring and maintaining the system. Raspberry Pis can also be used to host containers.

3.2 Containerisation

We review an own-build and an OpenStack-based solution and consider containers in a third architectural option. As they are the most recent technology, we introduce the basics here.

Containerisation allows a lightweight virtualisation through the bespoke construction of containers as application packages from individual images (generally retrieved from an image repository). This addresses performance and portability weaknesses of current cloud solutions. Given the overall importance of the cloud, a consolidating view on current activities is important. Many container solutions build on top of Linux LXC techniques. Recent Linux distributions - part of the Linux container project LXC - provide kernel mechanisms such as namespaces and cgroups to isolate processes on a shared operating system. Docker is the most popular container solution at the moment.

Container orchestration deals not only with turning applications on or off (i.e., start or stop containers), but also to move them around between servers. We define orchestration as constructing and managing a possibly distributed assembly of container-based software applications. Container orchestration allows users to define how to coordinate the containers in the cloud when the multi-container packaged application is deployed. Container orchestration defines not only the initial deployment of the containers, but also the management of the multi-containers as a single entity, such as availability, scaling and networking of the containers. Essentially cloud-based container construction is a form of orchestration within the distributed cloud environment.

The orchestration management provided by cluster solutions needs to be combined with development and architecture support.

Multi-PaaS based on container clusters is a solution for managing distributed software applications in the cloud, but this technology still faces challenges. These include a lack of suitable formal descriptions or user-defined metadata for containers beyond image tagging with simple IDs. Description mechanisms need to be extended to clusters of containers and their orchestration as well (Andrikopoulos, 2014). The topology of distributed container architectures needs to be specified and its deployment and execution orchestrated.

There is no widely accepted solution for the orchestration problems. We can illustrate the significance of this problem through a possible reference framework. Docker has started to develop its own orchestration solution (Swarm) and Kubernetes is another relevant project, but a more comprehensive solution that would address the orchestration of complex application stacks could involve Docker orchestration based on the topology-based service orchestration standard TOSCA, which is for instance supported by the Cloudify PaaS.

Figure 2 shows an orchestration plan for the case study. For a container host, it selects either the people management or the snow management as the required RPi configuration. For the people management architecture, it allows an upgrade to more local processing including analysis and local storage. The orchestration engine will actually take care of the deployment of the containers in the right order when needed.

3.3 Network and Data Management Challenges

Clustered containers in distributed systems require advanced network support. Traditionally, containers are exposed on the network via the shared host's address. In Kubernetes, each group of containers (called pods) receives its own unique IP address, reachable from any other pod in the cluster, whether co-located on the same physical machine or not. This requires advanced routing features based on network virtualisation.

Distributed container management also needs to address data storage besides network concerns. Managing containers in Kubernetes clusters can cause flexibility and efficiency problems because of the need for the Kubernetes pods to co-locate with their data. What is needed is a combination of a container with a storage volume that follows it to the physical machine, regardless of the container location in the cluster.

4 ARCHITECTURES AND EXPERIMENTATION

We report on experimental results with Raspberry Pi clusters to validate the proposed architectural solution. We look at the following settings: (i) own-build storage and cluster orchestration in Section 4.1, (ii) OpenStack storage in Section 4.2, (iii) Docker container orchestration in Section 4.3, and (iv) IoT/sensor integration in Section 4.4. For each, we describe the architecture and its evaluation through experiments. The criteria for the evaluation of the architecture are the following four: installation and management effort, power consumption, performance, cost.

These address the general suitability of the proposed architectures in terms of performance, but also take specifically practical concerns such as physical maintenance, power and cost into account.

4.1 Own-Build Cluster Storage and Orchestration

4.1.1 Storage and Orchestration Architecture

Our Raspberry Pi 1 (RPi 1) cluster can be configured with up to 300 nodes. The core of an RPi 1 is a single board with an integrated circuit with an ARM 700 MHz processor (CPU), a Broadcom VideoCore graphics processor (GPU) and 256 or 512 MB of RAM. There is also an SD card slot for storage and I/O units for USB, Ethernet, audio, video and HDMI. Power is provided via a micro-USB connector. As operating system, Raspbian is a version of the widely used Linux distribution Debian, which is optimised for the ARMv6 instruction set.

Our cluster uses a star network topology. One switch acts as the core of the star and other switches then link the core to the RPIs. A master node and an uplink to the internet are connected to the core switch for connectivity reasons.

We use a Debian 7 image to support core middleware services such as storage and cluster management. (Abrahamsson et al., 2013) have investigated basic storage and cluster management for an RPi cluster management solution.

In addition to deploying existing tools such as Swarm or Kubernetes, we also built our own dedicated tool for low-level configuration, monitoring, and maintenance of the cluster as an architectural option. This for instance provides flexibility for monitoring the joining and leaving of nodes to and from the cluster that we expect for dynamic edge cloud environments. The master handles (de)registration here.

4.1.2 Use Case and Experimentation

The suitability of an RPi for a standard application (responding to HTTP requests) was investigated. The total size of a sample file was 64.9 KB. An RPi (model B) was compared to a 1.2 GHz Marvell Kirkwood, a 1 GHz MK802, a 1.6 GHz Intel Atom 330, and a 2.6 GHz dual core G620 Pentium. All tested systems had a wired 1 GB Ethernet connection (which the Raspberry, having a 10/100 Mbit Ethernet card, could not utilize fully). ApachBench2 was used as the benchmark. The test involved a 1000 requests

Table 1: Speed and Power Consumption of the Raspberry Pi cluster, from R. van der Hoeven, “Raspberry Pi Performance", [Online Resource; accessed on 19th December 2017] Available at http://freedomboxblog.nl/raspberry-pi-performance/.

Device	Page/Sec	Power
RPi	17	3W
Kirkwood	25	13W
MK802	39	4W
Atom 330	174	35W
G620	805	45W

with 10 running concurrently. The following page/sec and power consumptions were measured, see Table 1 for details.

This has demonstrated the suitability of RPis for sensor integration and data processing in an environment subject to power supply problems, but where robustness is required.

4.2 Openstack Storage

4.2.1 Storage Management Architecture

(Miori, 2014) has investigated Openstack Swift as a distributed storage device that we ported onto RPis. This extends our earlier self-built storage approach by adopting an open-source solution.

Storage needs to be distributed over a whole cluster in our application context. Using a network storage system helps to improve the performance in a common filesystem for the cluster. We used here a four-bay Network Attached Storage (NAS) from QNAP Systems. However, we have also demonstrated that more resource-demanding Openstack Swift is a feasible option.

The Swift cluster provides a mechanism for storing objects such as application data as well as system data. Data is replicated and distributed among different nodes. We evaluated different topologies and configurations. This again demonstrates feasibility, but performance remains a key concern and further optimisation work is required.

4.2.2 Use Case and Experimentation

We have run several benchmarks (Miori, 2014) based on the Yahoo! Cloud Serving Benchmark (YCSB) and the SwiftStack Benchmark (ssbench). For single node installations, these show that a severe bottleneck emerges around data uploads. A single server cannot handle the traffic. Basically the server is so overloaded that either the cache (memcached) stops working or the container server stops working.

A slightly different picture emerges for clustered file storage. A real-world case study has been carried out using the ownCloud cloud storage system. We have installed a middleware layer on a Raspberry cluster that has been configured and benchmarked. Nonetheless, we could demonstrate the utility of it by running an application on top (ownCloud) enabling the cluster to provide a cloud storage service to the user. Performances are acceptable, yet further optimizations can be achieved.

We use a FUSE (filesystem-in-userspace) module called cloudfuse that is able to connect to a Swift cluster and display its content, as if it were a traditional directory-based filesystem. Each ownCloud instance has access to the Swift cluster via cloudfuse. OwnCloud is working well. The only limitations arise from cloudfuse. It is not possible to rename folders and it is not always fast. A direct implementation or improvement of the built-in Swift support is preferable. The application GUI itself loads quite fast, file listing takes a bit more time.

Swift is a scalable application: the addition of more Raspberry Pi predictably results in better performances. We cannot say yet if the trend is linear or not, thus further scaling up is needed (Jamshidi et al., 2016; Arabnejad et al., 2017).

The cluster costs are acceptable, see Table 2, in particular in comparison with modern gateway services such as the Dell Gateway 5000 series, which would cost a multiple including all hardware.

Table 2: Approximate costs of the Raspberry Pi cluster.

Component	Price	Units	Total
Raspberry Pi	35 €	7	245 €
PoE module	45 €	7	315 €
Cat.5e SFTP Cable	3 €	7	21 €
Aruba 2530 8 PoE+	320 €	1	320 €
Total			901 €

The PoE (Power over Ethernet) add-on boards and PoE managed switches we used are not essential to the project and could easily be replaced by a cheaper solution that involves a separate power supply unit and a simple unmanaged switch without having a negative impact on the system's performance.

4.3 Docker Orchestration

Docker and Kubernetes have been put on Raspberry Pis successfully (Tso, 2013), demonstrating the feasibility of running container clusters on RPis. We focus here on the edge cloud requirements. Our work specifically explores key features for a (middleware) platform for the edge cloud.

Fig. 3 describes the complete orchestration flow.

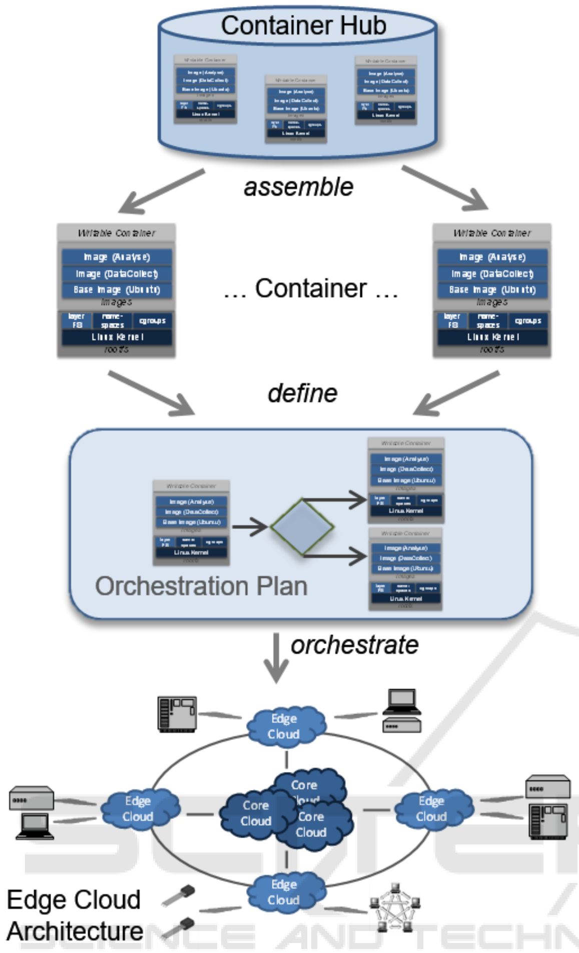

Figure 3: Overall Orchestration Flow.

It starts with the construction of the container from individual images from a container hub (an open repository of images). Different containers for specific processing needs are assembled into an orchestration plan. The plan is then enacted on the defined edge cloud topology.

The platform work described above has implemented core elements of a PaaS-oriented middleware platform. We have demonstrated that an edge cloud PaaS is feasible. We dedicate more space to containerisation as it overcomes some of the problems of the earlier two solutions.

4.3.1 Docker Orchestration Architecture

The RPi as an intermediate layer for local data processing is a feasible, cost-effective solution. A possible solution for an edge cloud architecture is to build a reliable, low-energy, low-cost computing device that is powerful enough to perform data-intense tasks.

Implementation – Hardware and Operating System

The test configuration we used is composed of seven Raspberry Pis that are connected to a switch with cables that aside from carrying the signals are also responsible for delivering the power to the devices. Each unit needs to be fitted with an additional PoE module. This add-on board is connected to the Raspberry Pi. It also replicates the GPIO interface, allowing further modules to be connected. A connection to a LAN or WAN is established by connecting the switch through a remaining Ethernet port. The switch can be configured to connect to an existing DHCP (dynamic host configuretion protocol) server which is responsible for distributing the network configuration parameters such as the IP (internet protocol) addresses. Alternatively it can create subnets via VLANs (virtual LANs).

Hypriot OS, a dedicated distribution of Debian, is the operating system. The distribution already contains Docker software. Note, that we replaced the default insecure authentication by a public-key authentication during the cluster setup process. This eliminates the need for a password-based authentication, the SSH daemon on the remote machine is configured to accept only public-key authentication, creating a more secure environment.

Swarm Cluster Architecture and Security

One node is selected to become the user's gateway into the cluster. The cluster is set up by creating Docker Machines on the gateway node and configuring both the OS and the Docker daemon on all Raspberry Pis that will be part of the cluster. Docker Machines allow the management of remote hosts by sending the commands from the Docker client over a secured connection to the Docker daemon on the remote machine. When the first Docker Machine is created, new TLS certificates are generated on the local machine and then copied over to the remote machines in order to create a trusted network.

While normal nodes run just one container that identifies them as a Swarm node, Swarm Managers deploy an additional container that provides the managerial interface. In addition, Swarm Managers can be configured in a redundant manner that improves the fault tolerance in case of a partial breakdown. In such a constellation, the Swarm Managers run as replicas. Furthermore, the Swarm Managers share their knowledge about the Swarm, and commands sent to a non-leading manager are propagated to the one in charge. This behaviour avoids inconsistencies in the Swarm that could lead to potential misbehaviour due to inconsistent data.

Service Discovery

Multi-host networks such as a Docker Swarm require a key-value store that holds information about the network state, including discovery, networks, endpoints and IP addresses. When deploying the Swarm image, the information about the service itself and how it can be reached needs to be provided. As a key-value store, we chose Consul. Consul does not require a continuous internet connection and allows redundant Swarm Managers, which is why it is ultimately selected. Additionally, it supports replicas of its own, increasing the fault tolerance on the discovery service side. Also Consul elects a leader amongst the instances that are part of the cluster, and propagates its information to every Consul node.

Swarm Handling

After the Docker Machines are set up successfully, the Swarm nodes communicate their presence to the Consul server, as well as the Swarm manager. The users are able to interact with the Swarm manager and also with each Docker Machine separately. This is done by requesting Docker-specific environment variables from the Docker Machine. When the shell is set up accordingly, the Docker client tunnels into the manager and executes the commands there. This way it is possible for the users to retrieve Swarm related information and perform Swarm related tasks such as launching a new container. The manager will consider each node of the Swarm and deploy it according to given constraints and to the selected Swarm strategy.

4.3.2 Docker Experimentation

The evaluation of the project focuses on the complexity to build and handle it and its costs, before concentrating on the performance and power consumption (von Leon, 2016).

Installation Effort / Costs. Assembling the hardware for the Raspberry Pi cluster does not require special tools or profound skills. This makes the architecture suitable to be installed in remote areas without expert support. Once running, handling the cluster is straightforward. Interacting with it is not different from handling a single Docker installation. The only aspect that has to be kept in mind is that ARM software and images are not always available for, so they might have to be created on purpose.

Performance & Power consumption. To evaluate the performance, we performed a stress test on the swarm manager by deploying many containers of a certain image over a short period of time, looking at the time to deploy the images as well as the launch time for containers. The test configuration deploys 250 containers on the Swarm with 5 requests at a time. To determine the efficiency of the Raspberry Pi cluster both the time to execute the analysis and the power consumption are measured and put into perspective with a Virtual Machine Cluster on a desktop computer and a Single Raspberry Pi. The desktop computer is a 64bit Intel Core 2 Quad Q9550 @2.83GHz Windows 10 machine with 8GB Ram and a 256GB SSD.

Table 3: Time comparison - listing the overall, the mean and the maximal time of container.

	Launching	Idle	Load
Raspberry Pi cluster	228s	2137ms	9256ms
Single Raspberry Pi node	510s	5025ms	14115ms
Virtual Machine Cluster	49s	472ms	1553ms
Single Virtual Machine Node	125s	1238ms	3568ms

Table 4: Comparison of the power consumption while idling and under load.

	Idle	Load
Raspberry Pi cluster	22.5W	25-26W
Single Raspberry Pi node	2.4W	2.8W
Virtual Machine Cluster	85-90W	128-132W
Single Virtual Machine Node	85-90W	110-114W

Table 5: Power consumption of the Raspberry Pi cluster while idling and under load.

	Idle	Load
Single node	2.4W	2.7W
All nodes	16W	17-18W
Switch	5W	8W
Complete system	22.5W	25-26W

In comparison, we can note a lack of performance for the Raspberry Pi cluster that is due to its limited single board architecture. The I/O of the micro SD card slot is relatively slow in terms of reading and writing, with maximally 22MB/s and 20MB/s, respectively. On the other hand, the network connectivity is only provided by 10/100Mbit/s Ethernet. Furthermore, with 26W (2.8W per unit) under load, the modest power consumption of the Raspberry Pi cluster puts its moderate performance into perspective and gives reason to assume the suitability of such systems in robustness requiring edge computing settings.

4.4 IoT Integration

We also need to evaluate the suitability of the proposed platform for IoT applications. For this, we chose a health care application using sensor integration: in the health care domain, we worked with health status sensing devices that were integrated using a Raspberry Pi device (Sanin, 2016).

A specific focus of this investigation has been on power management. While protocols have emerged that help to bridge between the sensor world and Internet-enabled technologies such as MQTT, this experimental work has also shown the need for dedicated power management to prevent overheating and reduce consumption.

5 DISCUSSION – TOWARDS AN EDGE CLOUD PaaS

Some PaaS have started to address limitations in the context of programming (such as orchestration) and DevOps for clusters. The examples used above allow some observations.

– Containers are largely adopted for PaaS clouds.
– Standardisation by adopting emerging de-facto standards like Docker or Kubernetes is also happening, though currently at a slower pace.
– Development and operations are still at an early stage, particularly if complex orchestrations on distributed topologies are in question.

We have shown the need for an Edge Cloud PaaS, and have implemented, experimented with and evaluated some core ingredients of these Edge Cloud PaaS, showing that containers are the most viable options over for instance an OpenStack attempt.

We can observe that cloud management platforms are still at an earlier stage than the container platforms that they build on. While clusters in general are about distribution, the question emerges as to which extent this distribution reaches the edge of the cloud with small devices and embedded systems. Whether devices running small Linux distributions such as the Debian-based DSL (which requires around 50MB storage) can support container host and cluster management is a sample question. Recent 3rd-generation PaaS are equally lightweight and aim to support the build-your-own-PaaS idea that is a first step. Edge Cloud PaaS then form the fourth generation bridging between IoT and Cloud technology.

An important concern for edge architectures is security. We have discussed some ID management concerns. IoT networks are distributed environments, in which trust between sensor owners and network and device providers does not necessarily exist. In order to support important orchestration activities from a security perspective, we want to record the provenance of sensor data or the fact that certain processing and interaction steps have actually been carried out (Gacitua and Pahl, 2017; Pahl, 2002; Gruhn et al., 1995). Blockchain technology is a solution for this in an untrusted environment. Many security related problems can be addressed using the decentralized, autonomous, and trusted capabilities of blockchain. Blockchain provides inherent security mechanisms capable of operating in an unreliable network, without relying on a central authority. Blockchain is a tamper proofed, distributed and shared database where all participants can append and read transactions but no one has full control over it. Every added transaction is digitally signed and timestamped, this means that all operations can be traced back, and their provenance can be determined (Dorri et al., 2017). The security model implemented by blockchain insures data integrity using consensus-driven mechanisms to enable the verification of all the transactions in the network, which makes all records easily auditable. This is particularly important since it allows tracking all sources of insecure transactions in the network (e.g., vulnerable IoT devices) (Nir, 2017). Additionally, blockchain can strengthen the security of edge components in terms of the identity management and access control and prevent data manipulation.

6 RELATED WORK

Container-based operating systems virtualisation has been demonstrated to be a viable option to hypervisors (Soltesz, 2007). This is a benefit for smaller devices due to their reduced sizes (Pahl et al., 2017).

For clusters of smaller devices, be that in constrained or mobile environments, the functional scope of a middleware layer needs to be suitably adapted (Qanbari et al., 2014). There is a need to provide robustness through mechanisms that deal with failure of connections and nodes. Flexible orchestration and load balancing are such functions. Also, security in the form of identity management is in unsecured environments a must. While we have added some security discussion in Section 5, further security related concerns such as data provenance or smart contracts accompanying orchestration instructions need to be investigated.

De Coninck et al. (2016) also approach the problem from a middleware perspective. Dupont et al. (2017) look at a specific concern in IoT settings – container migration (Jamshidi et al., 2017) to enhance the flexibility of the setting.

Bellavista and Zanni (2017) investigate, as we do, infrastructure based on Raspberry Pis to host Docker container. Their work also confirms the suitability of single-board devices. Work at the University of Glasgow (Tso et al., 2013) also explores Raspberry Pis for edge cloud computing. Their work involves lessons learned from practical applications of RPis in real-world settings. We have added in the solution presented here a comparative evaluation of different cluster-based architectures to their observations.

7 CONCLUSIONS

Edge clouds move the focus from heavy-weight data centre clouds to more lightweight resources, distributed to bring specific services to the users. They do, however, create a number of challenges. We have identified lightweight virtualisation and the need to orchestrate the deployment of these services as key challenges. We looked at platform (PaaS) specifically as the application service packaging and orchestration is a key PaaS concern (through of course not limited to PaaS).

Our aim was to compare recently emerging container technology and container cluster management and other architectural options such as OpenStack or bespoke solutions to determine the suitability of these approaches for edge clouds built on single-board affordable device clusters. Our observations support the current trend in container technology, but have also identified some limitations and aspects that need further investigation.

Container technology has a better potential than the other options to substantially advance PaaS technology towards distributed heterogeneous clouds through lightweightness and interoperability on, for instance, Raspberry Pis.

We can also conclude that significant improvements are still required to deal with data and network management aspects, as is providing an abstract development and architecture layer. Orchestration, as far as it is supported in cluster solutions, is ultimately not sufficient and needs to be extended to include better analysis and decision support (Fang et al., 2016). Suitable architecture that include coordination and brokerage options shall be considered (Fowley et al., 2016).

More work is also needed on improved performance management (Heinrich et al., 2017) and the adoption of microservices as an architectural principle (Pahl et al., 2016). Another concern that needs more attention is security. We are planning to use blockchain technology for provenance management.

ACKNOWLEDGEMENTS

The research reported here has been in part supported by the Free University of Bozen-Bolzano through the ECO and ECORE projects.

REFERENCES

P. Abrahamsson et al. (2013). Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment. IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom).

V. Andrikopoulos, S. Gomez Saez, F. Leymann, and J. Wettinger (2014). Optimal distribution of applications in the cloud. In Advanced Information Systems Engineering, pp. 75-90. Springer.

H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada (2017). A Comparison of Reinforcement Learning Techniques for Fuzzy Cloud Auto-Scaling. Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing.

P. Bellavista, and A. Zanni. (2017). Feasibility of fog computing deployment based on docker containerization over raspberrypi. Proceedings of the 18th International Conference on Distributed Computing and Networking. ACM.

T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner (2013). OpenTOSCA - a runtime for TOSCA-based cloud applications. In Service-Oriented Computing, pp. 692-695.

F. Bonomi, R. Milito, J. Zhu, and S. Addepalli (2012). Fog computing and its role in the internet of things. Workshop Mobile Cloud Computing.

A. Chandra, J. Weissman, and B. Heintz (2013). Decentralized Edge Clouds. IEEE Internet Computing.

C. Dupont, R. Giaffreda, and L. Capra (2017). Edge computing in IoT context: Horizontal and vertical Linux container migration. In Global Internet of Things Summit (GIoTS), pp. 1-4. IEEE.

E. De Coninck, S. Bohez, S. Leroux, T. Verbelen, B. Vankeirsbilck, B. Dhoedt, and P. Simoens (2016). Middleware Platform for Distributed Applications Incorporating Robots, Sensors and the Cloud. IEEE International Conference on Cloud Networking.

A. Dorri, S. Salil, S. Kanhere, and R. Jurdak (2017). Towards an Optimized BlockChain for IoT. Intl Conf on IoT Design and Implementation. ACM.

D. Fang, X. Liu, I. Romdhani, P. Jamshidi, and C. Pahl (2016). An agility-oriented and fuzziness-embedded semantic model for collaborative cloud service search, retrieval and recommendation. Future Generation Computer Systems 56, 11-26.

F. Fowley, C. Pahl, P. Jamshidi, D. Fang, and X. Liu (2016). A classification and comparison framework for cloud service brokerage architectures. IEEE Transactions on Cloud Computing.

V. Gacitua-Decar and C. Pahl (2017). Structural Process Pattern Matching Based on Graph Morphism Detection. International Journal of Software Engineering and Knowledge Engineering 27(2).

O. Gass, H. Meth, and A. Maedche (2014). PaaS Characteristics for Productive Software Development: An Evaluation Framework. IEEE Internet Computing, vol. 18, no. 1, pp. 56-64.

A. Gember, A Krishnamurthy, S. St John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar (2013). Stratos: A network-aware orchestration layer for middleboxes in the cloud. Duke University, Tech Report.

V. Gruhn, C. Pahl, M. Wever (1995). Data model evolution as a basis of business process management. OOER'95: Object-Oriented and Entity-Relationship Modeling, 270-281.

R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L.E. Lwakatare, C. Pahl, S. Schulte, and J. Wettinger (2017). Performance Engineering for Microservices: Research Challenges and Directions. Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion.

S. Helmer, C. Pahl, J. Sanin, L. Miori, S. Brocanelli, F. Cardano, D. Gadler, D. Morandini, A. Piccoli, S. Salam, A.M. Sharear, A. Ventura, P. Abrahamsson, and T.O. Oyetoyan (2016). Bringing the Cloud to Rural and Remote Areas via Cloudlets. Proceedings of the 7th Annual Symposium on Computing for Development, 14.

P. Jamshidi, M. Ghafari, A. Ahmad, and C. Pahl (2013). A framework for classifying and comparing architecture-centric software evolution research. European Conference on Software Maintenance and Reengineering.

P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, and G. Estrada (2016). Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures. 12th International ACM Conference on Quality of Software Architectures (QoSA).

P. Jamshidi, C. Pahl, and N.C. Mendonça (2017). Pattern-based multi- cloud architecture migration. Software: Practice and Experience 47 (9), 1159-1184.

N. Kratzke (2014). A Lightweight Virtualization Cluster Reference Architecture Derived from Open Source PaaS Platforms. Open Journal of Mobile Computing and Cloud Computing vol. 1, no. 2.

N. Kshetr (2017). Can Blockchain Strengthen the Internet of Things? IT Professional 19.4:68-72.

A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, and A. Campi (2013). Clouds of virtual machines in edge networks. IEEE Communications.

L. Miori (2014). Deployment and evaluation of a middleware layer on the Raspberry Pi cluster. BSc thesis, Univ of Bozen-Bolzano.

T.H. Noor, Q.Z. Sheng, A.H.H. Ngu, and S. Dustdar (2014). Analysis of Web-Scale Cloud Services. IEEE Internet Computing, 18(4), pp. 55-61.

C. Pahl (2002). A formal composition and interaction model for a web component platform. Electronic Notes in Theoretical Computer Science, vol 66(4), pp. 67-81.

C. Pahl and H. Xiong (2013). Migration to PaaS Clouds - Migration Process and Architectural Concerns. International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems.

C. Pahl (2015). Containerisation and the PaaS Cloud. IEEE Cloud Computing.

C. Pahl and B. Lee (2015). Containers and clusters for edge cloud architectures - a technology review. 3rd International Conference on Future Internet of Things and Cloud (FiCloud-2015).

C. Pahl and P. Jamshidi (2016). Microservices: A Systematic Mapping Study. Proceedings CLOSER Conference, 137-146.

C. Pahl, S. Helmer, L. Miori, J. Sanin and B. Lee (2016). A container-based edge cloud PaaS architecture based on Raspberry Pi clusters. IEEE International Conference on Future Internet of Things and Cloud Workshops (FiCloudW).

C. Pahl, A. Brogi, J. Soldani and P. Jamshidi (2017). Cloud Container Technologies: a State-of-the-Art Review. IEEE Transactions on Cloud Computing.

C. Pahl, P. Jamshidi and D. Weyns (2017). Cloud architecture continuity: Change models and change rules for sustainable cloud software architectures. Journal of Software: Evolution and Process 29 (2).

C. Pahl, P. Jamshidi, and O. Zimmermann (2018). Architectural principles for cloud software. ACM Trans. on Internet Technology (TOIT).

J. Sanin (2016). Evaluation and Development of a Biometric Measurement Platform with a Raspberry Pi. BSc thesis, Univ of Bozen-Bolzano.

D. von Leon (2016). Implementing an Edge Cloud Architecture with a Raspberry Pi Cluster. BSc thesis, Univ of Bozen-Bolzano.

S. Qanbari, F. Li, and S. Dustdar (2014). Toward portable cloud manufacturing services. Internet Computing, IEEE 18, no. 6: 77-80.

S. Soltesz, H. Potzl, M.E. Fiuczynski, A. Bavier, and L. Peterson (2007). Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. ACM SIGOPS Operating Systems Review, vol. 41, no. 3, pp. 275-287.

P. Tso, D. White, S. Jouet, J. Singer, and D. Pezaros (2013). The Glasgow Raspberry Pi cloud: A scale model for cloud computing infrastructures. Int. Works Resource Management of Cloud Comp.

J. Turnbull (2014). The Docker Book. Online at http://www.dockerbook.com/.

J. Zhu, D.S. Chan, M.S. Prabhu, P. Natarajan, H. Hu, and F. Bonomi (2013). Improving web sites performance using edge servers in fog computing architecture. Intl Symp on Service Oriented System Engineering.