A Container-based Edge Cloud PaaS Architecture based on Raspberry Pi Clusters

Claus Pahl, Sven Helmer, Lorenzo Miori, Julian Sanin
Free University of Bozen-Bolzano
39100 Bolzano, Italy

Brian Lee
IC4, Athlone Institute of Technology
Athlone, Ireland

***Abstract*—Cloud technology is moving towards multi-cloud environments with the inclusion of various devices. Cloud and IoT integration resulting in so-called edge cloud and fog computing has started. This requires the combination of data centre 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 solution. Containers are furthermore relevant for cloud platform concerns dealt with by Platform-as-a-Service (PaaS) clouds like application packaging and orchestration.

We demonstrate an architecture for edge cloud PaaS. For edge clouds, application and service orchestration can help to manage and orchestrate applications through containers. 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. 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 small single-board devices like Raspberry Pis. 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 as edge cloud. We illustrate key concepts of an *edge cloud PaaS* and refer to experimental and conceptual work to make that case.

***Index Terms*—Container, Cluster, Cloud, PaaS, Edge Cloud, Orchestration, Single-board Computer, Raspberry Pi.**

I. Introduction

Cloud computing is moving from large-scale centralised data centres to more distributed multi-cloud settings. These can consist of networks of larger and smaller virtualised infrastructure runtime nodes that connect to IoT (Internet-of-Things) devices with centralised data centres. To meet the flexibility, elasticity and cost requirements of smaller devices, virtualisation needs to be applied throughout, requiring Internet-of Things (IoT) infrastructures to be integrated. These architectures are often referred to as edge clouds or fog computing architectures [4]. Resulting from smaller devices and distribution, a more lightweight solutions than the current virtual machine (VM)-based virtualisation technology is needed. Furthermore, as another challenge, an architecture supporting the orchestration of lightweight virtualised runtimes is needed.

Firstly, we need virtualisation to achieve elasticity of large-scale shared resources. Virtual machines (VMs) have largely provided the compute infrastructure layer so far. We propose containers, which are a more lightweight virtualisation solution that is less resource and time consuming. Containers are specifically suitable for interoperable application packaging in the cloud and align with PaaS concerns [19]. Furthermore, they are flexible tools for packaging, delivering and orchestrating software infrastructure services as well as applications, i.e., tasks that are typically a PaaS (Platform-as-a-Service) focus. Containers can be used for componentising workloads in-between clouds. The basic ideas of containerisation are: (i) a lightweight portable runtime, (ii) the capability to develop, test and deploy applications to a large number of servers and (iii) the capability to interconnect containers. They also relate to the IaaS level through sharing and isolation aspects.

There is a need for an architecture that bridges between IoT, local compute devices and data centre clouds. Sensors at the core of IoT are often found in remote places, where a robust and low-power infrastructure for local computation and data storage needs to be provided. Small, single-board computers such as the Raspberry Pi (RPi) can be utilised here. With these devices, computation can be brought to the edge of the cloud, rather than data from the Internet-of-Things (IoT) to the cloud. This requires application distribution from the cloud centre rather than data streaming into the centre. Thus, for portable and interoperable software and application orchestration in a distributed edge cloud architecture, we require a lightweight distribution of packaged applications for deployment and management. The solution can be again containerisation, but would need to be extended to deal with orchestration. Thus, we consider managing clusters of containers and their orchestration in a cloud setting.

We discuss key ingredients of an *edge cloud PaaS*, i.e., PaaS middleware features like container and cluster management on Raspberry Pis, which provides a solution to the problems outlined in our edge cloud review [19]. We refer to conceptual orchestration language work as well as experimental work to demonstrate the feasibility of the proposed architecture.

We start with a review of the architectural setting for edge clouds, before introducing container virtualisation in Section III. In Section IV, we look at edge PaaS cloud concerns and Raspberry Pis as the infrastructure architecture. Finally, clustering and orchestration are discussed in Section V and evaluated in Section VI, before ending with some conclusions.

II. Edge Cloud Architectures

Cloud edge computing is moving computing application and data management services away from data centre architectures

to the edges of the network towards IoT infrastructures [5]. The objective is to allow analytics and knowledge generation services to be placed at the source of the data (e.g., sensors). This approach requires leveraging resources that may not be continuously connected. For instance, cloud computing at the edge links into the internet of things. In this setting, the core cloud provides a globalised view, whereas edge cloud nodes provide localised views.

Distributed clouds are often classified into three architectural models, ranging from tightly coupled to highly dispersed ones: (i) Multi-datacentre clouds with multiple, but tightly coupled data centres under one control. (ii) Loosely coupled multi-service clouds combining services from different providers, and (iii) decentralised edge clouds utilising edge resources to provide highly dispersed data and compute resources. The third category is the one we target.

A. Edge Cloud Architecture Requirements

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

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

Virtualised resources can support edge cloud architectures [12] which are programmable or configurable, but these differ in size and type, such as nodes and edges. This results in different resource restrictions, which in turn requires some form of lightweightness of the virtualisation technique [26]. In edge cloud architectures with IoT objects integrated, we need (i) compute and storage resources and (ii) platform services and applications to be managed, 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 (SDN)). 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 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. 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 oriented management at a typical PaaS layer.

B. A Motivational Use Case

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

- weather: air temperature, air humidity, sun intensity
- snow: quality (snow humidity, temperature)
- people: location and number

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

- People management: through apps (cf. the go2ski Trentino app), skiers can get recommendations regarding snow quality and possible over-crowding at lifts and on slopes. This mobile phone app uses the cloud as an intermediary to receive data from, but the performance of the architecture would benefit from data preprocessing 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 the 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 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 in different times at 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.

C. Edge Cloud Architecture Principles

An architecture that addresses the requirements described and illustrated above can be organised into three layers: At

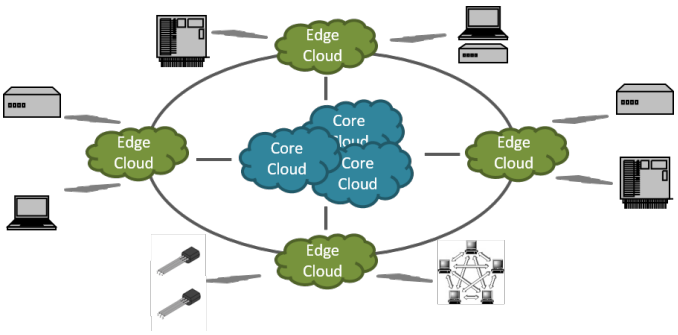


Fig. 1. Resources Architecture as Cluster-based Container Architecture

the lowest level, we locate a smart things network (e.g. smart sensor, wireless, actuator, mobile, and ad-hoc networks – possibly with an MQTT protocol on top employing a pub/sub model). On the middle level we have a field area network (e.g., 3/4G, LTE, WIFI) and the IP core infrastructure. Finally, on the top level, a virtual compute and storage cloud. The operation and management of this architecture is based on centralised providers to push out (deploy) services in application packages (such as the suggested containers) to clustered edge clouds. Docker container architectures for clouds exist [24], but orchestration and topology management require more attention. Solutions in this space include Kubernetes, but, as we will see, need more exploration on smaller devices.

Architecture. IoT integration is the key concern here. In well supported, stable environments, powerful network edge gateways can be used to interface sensors with cloud computing infrastructures. These would provide analytics capabilities and extensive input/output (I/O) options as standard.

- However, with prices starting from more than $1,000 for e.g., a Dell Edge Gateway 5000 series, the costs are often prohibitive.
- Furthermore, a more localised solution with many nodes, possibly clustered, is necessary.
- Another challenge is the need to adapt computational capabilities dynamically.

We propose to replace powerful gateways with clusters of single-board computers such as Raspberry Pis or Arduinos. These are overall cheaper, consume less energy, and allow a flexible and localised placement in remote or not well-supported environments. By clustering the devices we do not just get more computational power, we also gain robust solutions that are resilient against power failures or environmental challenges (like changing temperatures).

Development and Operations. We assume a multi-cloud deployment that requires lightweight application packaging, distribution and support of topology specification and management due to the dispersed nature of the smaller device clusters.

We need to allow various services such as security and analysis services deployed on these resources in addition to based data collection and processing [8]. Furthermore, the allocation of these services might change over time [9]. Therefore, the management of these architectures needs to be supported through an orchestration technique based on topology patterns reflecting common and reference architectures [25].

Since these aspects are typical platform or middleware features, we need edge cloud PaaS capabilities. Several technologies in the container technology space exist that might contribute to this specific PaaS solution:

- Application packaging through containerisation: Containers can be used to distribute service and applications to the edge. Docker is a good example for this.
- Programmability: Orchestration can be supported using topology specification based on topology patterns [3]. Overall, service composition (i.e., orchestration) needs to encompass the component or service life-cycle – deploy, patch, shutdown. A possible solution that we will introduce later is TOSCA [3].

We look at edge clouds from a PaaS perspective taking lightweight application packaging and topology specification into account. However, we first start with the proposal of using container-based Raspberry Pi clusters as the infrastructure.

D. Raspberry Pi Clusters for Edge Cloud

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 [1]. These single-board computer create challenges, but also opportunities. A Raspberry Pi (RPi) is relatively cheap (with 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 in computing power, in general we cannot run computationally intensive software on it. Nevertheless, this drawback can be remedied (to a certain degree) by combining a larger number into a clusters. This also allows the creation of differently configured and customised 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; in the next section we introduce the container principles first, before returning to the edge cloud context later on.

III. Containers

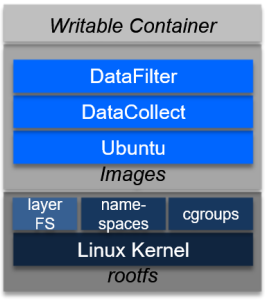
Virtualisation helps with scheduling processes as manageable container units. Multi-tenant clouds require sharing of resources such as disk space and CPU [17]. This underlying platform and infrastructure has to be shared in a secure, but also portable and interoperable way [20].

A. Container Principles

At the PaaS level, packaging and application management is an additional requirement and containers address exactly these requirements. A container is a packaged self-contained, ready-to-deploy set of parts of applications that can include both middleware and application logic [22], [18] (see Fig. **??**).

The Linux container project LXC uses kernel mechanisms that isolate processes in shared environments [22]. Containers are virtualisation mechanisms suitable for application management in PaaS clouds. A container is represented by lightweight images – VMs are also based on images, but full monolithic ones. Processes running in a container are almost fully isolated. Container images are the building blocks from which containers are launched. Docker is a container building on top of Linux LXC. A Docker image is made up of file systems layered over each other.

- Booting: Docker mounts the rootfs as read-only (as in a traditional Linux boot), but instead of changing the file system to read-write mode, it uses a union mount to add a writable file system on top of the read-only file system.
- Mounting: This allows multiple read-only file systems to be stacked on top of each other. Only the top layer (container) is writable.



Complete Docker images form portable application containers that built around container engines for container execution [24]. This is called *lightweight* as single images can easily be changed and distributed.

Containers are used for PaaS Clouds. For instance, Warden provides an API in Cloud Foundry for managing a collection of containers. Containers can be limited in terms of resource access. Garden is a re-coding of Warden that provides technology for Diego (container architecture for Cloud Foundry).

Another example is Rocket, a new container runtime from the CoreOS project (CoreOS is a Linux derivate for massive server deployments). Rocket is an alternative to Docker, specifically designed for composability, security, and speed – important properties in the edge cloud domain, reflecting ongoing concerns in this area.

Two problems remain around containers: Firstly, managing dependencies between containers in multi-tier, distributed applications is a problem. Something like an orchestration plan can describe containerised components, their dependencies and their lifecycle. A PaaS cloud can then enact orchestration workflows from a plan through a container engine. PaaS services support packaging and deployment of containers.

Secondly, to define, deploy, and operate cross-platform capable cloud services in a lightweight way that suits the proposed single-board computing platform [14]. This results in a need to transfer cloud deployments between cloud providers in a distributed context, i.e., to orchestrate services in a customised way. Some PaaS are already lightweight virtualisation solutions in this sense, which we will see later on.

B. Application Containerisation and Container Management

Containers can encapsulate a number of application components through the image layering and extension process. A container solution consists of two main components – (i) an application container engine to run images and (ii) a repository/registry that is operated via push and pull operations to transfer images to and from host-based container engines.

- *Container repositories* play a central role in providing possibly reusable private and public container images.
- The *container API* supports life-cycle operations like creating, composing, distributing containers, starting and running commands in images.
- Containers are *created* by assembling them from individual images, possibly extracted from the repositories.

Storage and network management are two specific platform/middleware services that are needed to support containers as application packages for distributed edge clouds:

- Docker manages data through data volumes and data volume containers. Data storage operations can add data volumes to any container. A data volume is a designated directory within one or more containers that bypasses the union file system. This allows to provide persistent or shared data. Volumes can then be shared and reused between containers (Fig. 2). A data volume container enables sharing persistent data between application containers through a dedicated, separate data storage container.
- Secondly, network management is based on two methods for assigning ports on a host – through network port mappings and container linking. Applications can connect inside a Docker container via a network port. Container linking allows linking multiple containers together and sending information between them. Linked containers can transfer their data using environment variables.

IV. Edge Clouds – Container and Orchestration

Containers appear as a highly suitable technology for application packaging and management in edge clouds that are more flexible and lightweight than VMs as the format to provision platform and application components.

What we propose is an edge cloud PaaS built on containers, suitable for clusters of single-board computers. PaaS provide mechanisms for deploying applications, designing applications for the cloud, pushing applications to their deployment environment, using services, migrating databases, mapping custom domains, IDE plugins, or a build integration tool. PaaS exhibit features like built farms, routing layers, or schedulers that dispatch workloads to VMs [7].

A. Evolution of PaaS – towards Edge Cloud PaaS

Container frameworks address the application deployment problems through interoperable, lightweight and virtualised packaging. Containers for application building, deployment and management (through a runtime) provide interoperability. Containers are interoperable – those produced outside a PaaS can be migrated in since the container encapsulates the application. Some PaaS are now aligned with containerisation and standardised application packaging. Many PaaS use Docker and some have their own container foundation for running

platform tools. This development is part of an evolution of PaaS, moving towards container-based, interoperable PaaS.

1) Proprietary: The first PaaS generation included fixed proprietary platforms, e.g., Azure or Heroku.
2) Open-Source: The second PaaS generation included open-source solutions, e.g., Cloud Foundry or OpenShift, allowing users to run their own PaaS (on-premise or in the cloud), many with built-in support of containers. Openshift moves from its own container model to Docker. Cloud Foundry does the same through Diego. However, these two PaaS platforms treat containers differently. Cloud Foundry supports state-less applications through containers, but lets stateful services run in VMs. Openshift does not distinguish between them.
3) Micro-PaaS: The current third generation of PaaS includes for example Deis or Flynn. These are built on Docker from scratch and are deployable on own servers or on public IaaS clouds. Flynn and Deis, for examples, have created a micro-PaaS concept where small PaaS can be run on limited hardware with little overhead. They have adopted elements of CoreOS for a clustered, distributed architecture management. This builds on lightweight, decoupled services facilitated by Docker. This specifically benefits distributed multi-tenancy cloud on reduced capability resources (such as RPis).
4) Edge Cloud PaaS: We forsee a fourth generation of edge cloud PaaS that provide PaaS features for edge cloud environments, i.e., develop micro-PaaS further into edge environments, focusing more on clustering and orchestration across micro-computer infrastructures.

V. Clustering and Orchestrating Containers

In order to satisfy edge cloud requirements, the single container host concept needs to be expanded into clusters of container hosts to run containerised edge cloud platform services as well as applications over multiple clusters in multiple clouds in order to meet the edge cloud requirements [10]. The interoperability of containers makes this possible.

A. Container Clusters

A cluster architecture groups hosts into clusters [10]. Fig. 2 illustrates an architectural framework based on common container and cluster concepts. Container hosts are linked into a cluster configuration. Central concepts are clusters, containers, application services, volumes and links. A *cluster* consists of several (host) nodes. Each (host) node holds several containers with common platform services such as scheduling, load balancing and applications. Each *container* in a cluster can hold provided services such as payload services, which are once-off services (e.g., print), or functional (middleware service) components. *Application services* are logical groups of containers from the same image. Application services allow scaling an application across nodes. *Volumes* are used for applications that need data persistence. Data stored in these data volumes mounted by containers persists. Finally, *links* allow two or more containers to connect and communicate.

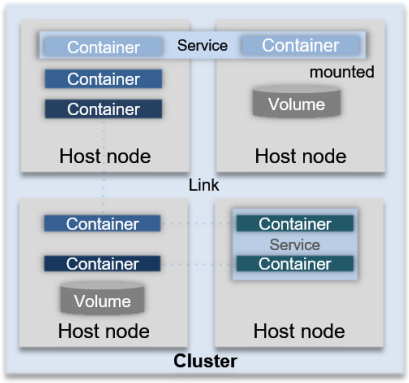


Fig. 2. Container-based Cluster Architecture – an architectural framework.

Resulting from this architectural scenario is an abstraction layer for cluster-based service management that is different from the container features provided by Docker. A cluster management architecture has the following components: the service node (cluster), an API, a platform service manager, a lifecycle management agent and a cluster head node service.

The deployment of distributed applications as containers is supported using a virtual scalable *service node (cluster)*, supporting scaling, load balancing, failover. An *API* allows operating clusters from the creation of services and container sets to other life-cycle functions. A *platform service manager* looks after the software packaging and management. An *agent* manages the container life-cycles. A *cluster* head node service is the master that receives commands from the outside and relays them to container hosts. This allows development of an edge cloud architecture without consideration of underlying network topology and avoids manual configuration [7].

A cluster architecture is composed of engines to share service discovery and orchestration/deployment (load balancing, monitoring, scaling, and also file storage, deployment, pushing, pulling). Requirements for these cluster architectures have been summarised [11]. A lightweight virtualised cluster architecture building on containerisation should provide a number of management features as part of the abstraction on top of the container hosts:

- Hosting containerised services and providing secure communication between these services,
- Auto-scalability and load balancing support,
- Distributed scalable service discovery and orchestration,
- Migration of service deployments between clusters.

Mesos is a cluster management platform – it binds distributed hardware resources into a pool to be used by applications to manage workload distribution. The Mesos kernel runs on all machines in the cluster and provides applications with resource management and scheduling across clouds.

An example of clustering management at a higher level than Mesos is Kubernetes, which can be configured to orchestrate Docker containers on Mesos. Kubernetes is based on processes that run on Docker hosts. These bind hosts into clusters and manage the containers. Openshift is a PaaS example that has adopted Kubernetes. Kubernetes competes

with platform-specific evolution towards container-based orchestration. Cloud Foundry is such an example that uses Diego as an orchestration engine for containers.

B. Network and Data Management Challenges

Clustered containers in distributed systems require advanced network support. Traditionally, containers are exposed on the network via the shared hosts 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.

C. Orchestration and Topology

The management solution 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 [2]. 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 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.

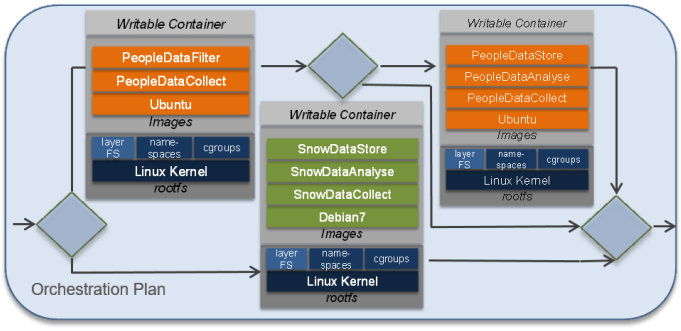


Fig. 3. Orchestration Plan for the Case Study.

In Figure 3, we show an orchestration plan for the case study. For a container host, it select 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.

VI. Implementation and Validation

Our Raspberry Pi cluster has been installed, equipped with key platform services such as storage and cluster management and has been used in different experiments that aim to address platform evaluation as well as the feasibility of sensors in an IoT context. We also report here on the development of a dedicated topology and orchestration language on top of the platform services. We also cover related work in this context.

A. Implementation – Hardware and PaaS Services

Our Raspberry Pi 1 (RPi 1) cluster can be configured with up to 300 nodes [1]. The core of an RPi 1 is a single board with an integrated circuit with an ARM 700 MHz processor (CPU)[1], 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.[2] The RPi 1 comes in two variants, A and B, with the latter offering 512 MB of RAM (instead of 256 MB) and an on-board Ethernet port. RPIs can be powered using mobile phone chargers with micro-USB interfaces. As operating system, Raspbian is a version of the well-known Linux distribution Debian, 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.

- cluster management: [1] has investigated basic storage and cluster management for an RPi cluster management solution. Rather than deploying Kubernetes, we built our own dedicated tool for low-level configuration, monitoring, and maintenance of the cluster. This 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.
- storage management: [13] has investigated Openstack Swift as a distributed storage device 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. 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

[1] 64-bit or OpenPower architecture not yet supported.

[2] In our case we use old modified power supplies of desktop PC, as powering hundreds of RPis with phone chargers is impractical.

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. Currently, we are working on a configuration involving the more powerful Raspberry Pi 2.

A real-world case study has been carried out using the *ownCloud* cloud storage as a use case.

Docker and Kubernetes have been put on Raspberry Pis successfully [23], demonstrating the feasibility of running container clusters on RPis. We focus here on the edge cloud requirements. Our work specifically explores middleware platform service need for the edge cloud. Fig. 4 describes the complete 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.

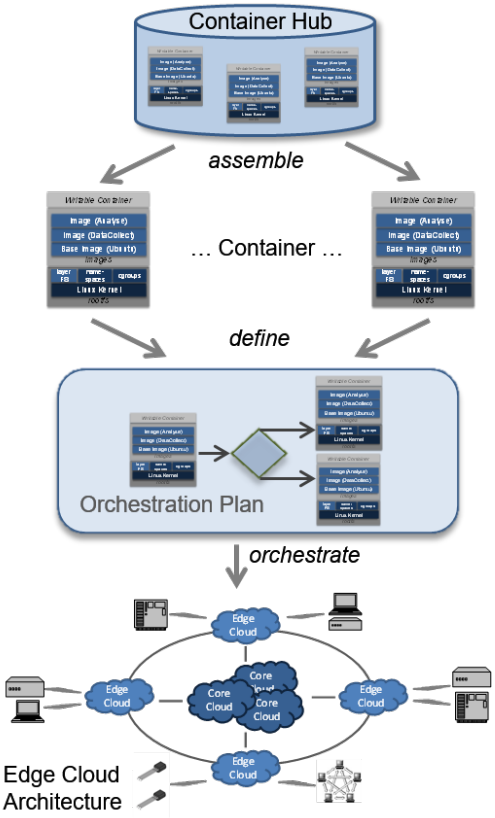


Fig. 4. Overall Orchestration Flow.

B. Experimentation

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 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 [21]. A specific focus here has been on power management. While protocols emerge 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.

We investigated the suitability of an RPi for a standard application (responding to HTTP requests). 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 with 10 running concurrently. The following page/sec and power consumptions were measured: RPi: 17, 3W; Kirkwood: 25, 13W; MK802: 39, 4W; Atom 330: 174, 35W; G620: 805, 45W.

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.

We have also looked at this from the cost perspective, where often pricing prevents technologies to be widely adopted. The RPi as an intermediate layer for local data processing is a feasible, cost-effective solution.

C. PaaS-level Topology and Orchestration Specification

In an effort to support more comprehensive PaaS service, better specification of management aspects like orchestration is needed. To better support the orchestration of containers in edge cloud environments, we have suggested a TOSCA-based orchestration language for Docker-based containers [19]. This abstract language helps in defining common orchestration patterns for the cloud as templates, see Figure 5 where the TOSCA framework is applied to container topology specification (left) and orchestration plans (bottom right).

While basic clustering and orchestration support exists for containers, within Docker or through additional mechanisms like Kubernetes, better programming support (towards more edge cloud PaaS development support) is needed to specify container orchestration for edge clouds. As our motivational use case above demonstrates, different orchestration patterns emerge, that can ideally be supported through orchestration templates as provided by TOSCA for complex topologies supporting different architectural patterns and styles [16].

D. 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. Firstly, containers are largely adopted for PaaS clouds. Secondly, standardisation by adopting emerging de-facto standards like Docker or Kubernetes is also happening, though currently at a slower pace. Thirdly, development and operations are still

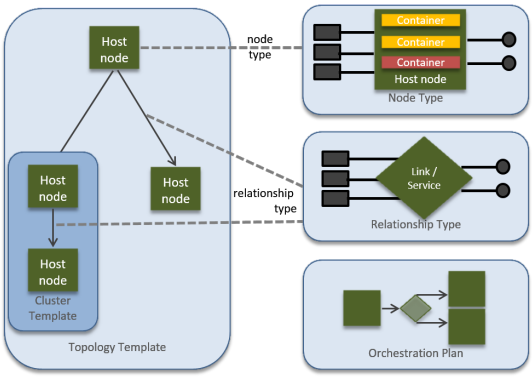


Fig. 5. TOSCA-based Cluster Topology and Orchestration Specification.

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.

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.

VII. Conclusion

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 service 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 use recently emerging container technology and container cluster management to determine the suitability of these approaches for edge clouds built on single-board affordable device clusters. The observations here support the current strong trend in this technology, but have also identified some limitations and aspects that need further investigation.

Container technology has the potential 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 based on better semantic descriptions [15]. More work is also needed on improved performance management.

References

[1] P. Abrahamsson et al., "Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment," International Conference on Cloud Computing Technology and Science (CloudCom), pp. 170-175. 2013.
[2] V. Andrikopoulos, S. Gómez Sáez, F. Leymann, and J. Wettinger, "Optimal distribution of applications in the cloud," In Advanced Information Systems Engineering, pp. 75-90. Springer, 2014.
[3] T. Binz, U. Breitenbcher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner, "OpenTOSCA – a runtime for TOSCA-based cloud applications," In Service-Oriented Computing, pp. 692-695, 2013.
[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," Workshop Mobile Cloud Computing, 2012.
[5] A. Chandra, J. Weissman, and B. Heintz, "Decentralized Edge Clouds," IEEE Internet Computing, 2013.
[6] F. Fowley, C. Pahl, and L. Zhang, "A comparison framework and review of service brokerage solutions for cloud architectures," 1st International Workshop on Cloud Service Brokerage (CSB'2013). 2013.
[7] O. Gass, H. Meth, A. Maedche, "PaaS Characteristics for Productive Software Development: An Evaluation Framework," IEEE Internet Computing, vol. 18, no. 1, pp. 56-64, 2014.
[8] A. Gember, A Krishnamurthy, S. St John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," Duke University, Tech Report, 2013.
[9] P. Jamshidi, M. Ghafari, A. Ahmad, and C. Pahl, "A framework for classifying and comparing architecture-centric software evolution research," European Conference on Software Maintenance and Reengineering, 2013.
[10] V. Koukis, C. Venetsanopoulos, and N. Koziris, "∼okeanos: Building a Cloud, Cluster by Cluster," Internet Computing, 17(3), pp. 67-71, 2013.
[11] N. Kratzke, "A Lightweight Virtualization Cluster Reference Architecture Derived from Open Source PaaS Platforms," Open Journal of Mobile Computing and Cloud Computing vol. 1, no. 2, 2014.
[12] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, and A. Campi, "Clouds of virtual machines in edge networks," Communications Magazine, IEEE 51(7): 63-70. 2013.
[13] L. Miori, "Deployment and evaluation of a middleware layer on the Raspberry Pi cluster," BSc thesis, Univ of Bozen-Bolzano. 2014.
[14] T.H. Noor, Q.Z. Sheng, A.H.H. Ngu, and S. Dustdar, "Analysis of Web-Scale Cloud Services," IEEE Internet Computing, 18(4), pp. 55-61, 2014.
[15] C. Pahl, "An ontology for software component matching," International Journal on Software Tools for Technology Transfer, 9(2):169-178. 2007.
[16] C. Pahl, S. Giesecke, and W. Hasselbring, "Ontology-based modelling of architectural styles," Information and Software Technology, 51(12). pp. 1739-1749. 2009.
[17] C. Pahl and H. Xiong, "Migration to PaaS Clouds - Migration Process and Architectural Concerns," International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, 2013.
[18] C. Pahl, "Containerisation and the PaaS Cloud," IEEE Cloud Computing, 2 (3). pp. 24-31, 2015.
[19] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures - a technology review," 3rd International Conference on Future Internet of Things and Cloud (FiCloud-2015). 2015.
[20] S. Qanbari, F. Li, and S. Dustdar, "Toward portable cloud manufacturing services," Internet Computing, IEEE 18, no. 6: 77-80. 2014.
[21] J. Sanin, "Evaluation and Development of a Biometric Measurement Platform with a Raspberry Pi," BSc thesis, Univ of Bozen-Bolzano. 2016.
[22] S. Soltesz, H. Pötzl, M.E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," ACM SIGOPS Operating Systems Review, vol. 41, no. 3, pp. 275-287, 2007.
[23] P. Tso, D. White, S. Jouet, J. Singer, and D. Pezaros, "The Glasgow Raspberry Pi cloud: A scale model for cloud computing infrastructures," 1st Int. Workshop on Resource Management of Cloud Computing. 2013.
[24] J. Turnbull, "The Docker Book," http://www.dockerbook.com/. 2014.
[25] M.X. Wang, K.Y. Bandara, and C. Pahl, "Integrated constraint violation handling for dynamic service composition," IEEE International Conference on Services Computing SCC'2009, 2009.
[26] J. Zhu, D.S. Chan, M.S. Prabhu, P. Natarajan, H. Hu, and F. Bonomi, "Improving web sites performance using edge servers in fog computing architecture," Intl Symp on Service Oriented System Engineering, 2013.