Containers and Clusters for Edge Cloud Architectures – a Technology Review

Claus Pahl
Irish Centre for Cloud Computing and Commerce IC4 & Lero, the Irish Software Research Centre
Dublin City University
Dublin 9, Ireland

***Abstract*—Cloud technology is moving towards more distribution across multi-clouds and the inclusion of various devices, as evident through IoT and network integration in the context of edge cloud and fog computing. Generally, lightweight virtualisation solutions are beneficial for this architectural setting with smaller, but still virtualised devices to host application and platform services, and the logistics required to manage this.**

Containerisation is currently discussed as a lightweight virtualisation solution. In addition to having benefits over traditional virtual machines in the cloud in terms of size and flexibility, containers are specifically relevant for platform concerns typically dealt with Platform-as-a-Service (PaaS) clouds such as application packaging and orchestration. For the edge cloud environment, application and service orchestration can help to manage and orchestrate applications through containers as an application packaging mechanism. We review edge cloud requirements and discuss the suitability container and cluster technology of that arise from having to facilitate applications through distributed multi-cloud platforms build from a range of networked nodes ranging from data centres to small devices, which we refer to here as edge cloud.

Keywords: Container, Cluster, Virtualisation, Cloud Computing, PaaS, Multi-cloud, Edge Cloud, Orchestration, Topology.

I. Introduction

Cloud computing is moving from centralised, large-scale data centres to a more distributed multi-cloud setting comprised of a network of larger and smaller virtualised infrastructure runtime nodes. Virtualising reaches the network and allows Internet-of Things (IoT) infrastructures to be integrated. These architectures and their setting are often referred to as edge clouds, edge computing or fog computing [3]. As a challenge resulting from distribution, we need a more lightweight solutions than the current virtual machine (VM)-based virtualisation technology. Furthermore, as another challenge, the orchestration of lightweight virtualised runtimes is needed.

Regarding the first challenge, the cloud relies on virtualisation techniques to achieve elasticity of large-scale shared resources. Virtual machines (VMs) have been at the core of the compute infrastructure layer providing virtualised operating systems. We will investigate containers here, which are a similar, more lightweight virtualisation concept, i.e., less resource and time consuming. VMs and containers are both virtualisation techniques, but solve different problems. Containers have been suggested as a solution for more interoperable application packaging in the cloud and should therefore address the PaaS concerns.

Contrary to VMs, containers can be seen as more flexible tools for packaging, delivering and orchestrating both software infrastructure services and applications, i.e., tasks that are typically a PaaS (Platform-as-a-Service) focus. Containers built on recent advances in virtualisation, allowing a more portable way aiming at more interoperability [15] while still utilising operating systems (OS) virtualisation principles. VMs on the other hand are about hardware allocation and management (machines turned on/off and provisioned). With them, there is an IaaS (Infrastructure-as-a-Service) focus on hardware virtualisation. Containers as a replacement for VMs are only a specific use case where the allocation of hardware resources is done through containers by 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. Containers address concerns at the cloud PaaS level. They also related to the IaaS level through sharing and isolation aspects that exemplify the evolution of OS and virtualisation technology.

Regarding the second challenge, for portable and interoperable software applications in a distributed cloud architecture, we need a lightweight distribution of packaged applications for deployment and management [5]. Again, the solution can be containerisation, but would need to be extended to deal with the orchestration needs. In order to assess the concerns here, we look into managing clusters of containers and their orchestration in a cloud setting.

This article reviews the suitability of container technology for edge clouds and similar settings, starting with summarsing the virtualisation principles behind containers and identifying key technical requirements of edge cloud architectures. The relevance of the new container technology for PaaS cloud concerns with application packaging and orchestration concerns shall be specifically investigated.This research shall clarify how containers can change PaaS clouds as a platform technology. As we consider distributed clouds, the resulting requirements for application packaging and interoperable orchestration over clusters of containers are central. We discuss what is needed to evolve PaaS technology further as a distributed cloud software platform resulting in a discussion of achievements and limitations of the state-of-the-art. In order to illustrate the technologies in question, we will refer to sample technologies they exemplify technology trends. As part of this investigation, we abstract concepts from various technology platforms and condense them into sets of common principles, supplemented by architecture frameworks that represent best

practice.

We start with a more detailed review of the architectural setting in Section II and discuss the resulting virtualisation and management needs to Section III. We then introduce container-based virtualisation in Section IV. In Section V, we focus the investigation on PaaS cloud concerns. Finally, clustering and orchestration are discussed in Section VI, before ending with some conclusions.

II. Towards Edge Clouds

Cloud edge computing is pushing computing applications, data, and services away from centralized cloud data centre architectures to the edges of the underlying network [4]. The objective is to allow analytics and knowledge generation services to be placed at the source of the data. Cloud computing at the edge of the network links into the internet of things (IoT). The core cloud provides a globalised view; edge clouds are responsible for localised views (to host services close to or at endpoints of networks), on-device, private cloud like infrastructures. We can classify distributed clouds into three architectural models, ranging from tightly coupled to highly dispersed ones:

- Multi-datacentre clouds with multiple, but tightly coupled data centers under control of the same provider.
- Loosely coupled multi-service clouds combine services from different cloud providers.
- Decentralized edge clouds utilize edge resources to provide data and compute resources in a highly dispersed manner.

A. Edge Cloud Technology Requirements

In order to support specifically the edge clouds, we need for instance location-awareness and computation placement, replication, and recovery. For example, consider a content analysis application that processes digital multimedia content hosted throughout the Internet. Edge resources would be required for both computation and data storage to address the wide data distribution. The necessary edge resources could be dedicated resources spread across content distribution networks.

Various virtualised resources exist in this setting that can support edge clouds [11] – all programmable, but different in size and type, such as nodes and edges (the latter are actually nodes of the network itself). This results in different resource restrictions calling for lightweightness with respect to the virtualisation approach [20]. In centre and edge clouds, but also the IoT objects linked to, compute and storage resources, platform services and applications need to be managed, i.e., packaged, deployed and orchestrated, see Figure 1. For the network, virtualisation capacity is required as well – cf. work on software-defined networks (SDN). We need to support data transfer between virtualised resources and to provide compute, storage, network resources between end devices and traditional cloud computing data centres.

- Concrete requirements arising from this are location awareness, low latency and mobility support to manage cloud end points with rich (virtualised) services.

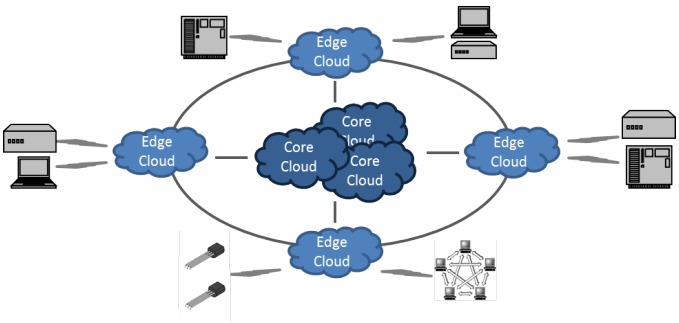

Fig. 1. Resources Architecture as Cluster-based Container Architecture

- SDN-virtualised nodes and also virtualised edges and connectors require a lightweight virtualisation technology of efficiently deploy portable services at edges.
- This type of virtualised infrastructure might provide end-user access and IoT links – through possibly private edge clouds (which are technically miniclouds).

These need to be configured and updated in a secure way – this particularly applies to the service management. We would also need a development layer on top to provision and manage applications on these infrastructures. Solutions here could comprise common topology patterns, controlling application lifecycle and an easy-to-use API. The right abstractions for edge cloud oriented management at a typical PaaS layer would be beneficial.

B. Architectural Principles

An architecture addressing the challenges can be organised into layers (bottom to top):

- at the bottom a smart things network (smart sensors network, wireless sensor and actuator networks, mobile and ad-hoc networks – possibly with a MQTT protocol on top with a pub/sub model),
- a field area network (such as 3/4G, LTE, WIFI) and then the IP core infrastructure, and
- a virtual compute/storage/network cloud with applications on top.

The operation and management of this architecture will see service providers push out (i.e., deploy) service in suitable application packages (such as containers) to clustered edge clouds. While some solutions like Docker container architectures for clouds exist [18], there is still a need for a topology specification and a derived orchestration/choreography plans. Existing solutions in this space include Kubernetes, but, as we will see, leave some orchestration questions unanswered.

C. Challenges – Development and Operations

We assume the requirements to include multi-cloud deployment (via lightweight application packaging, distribution and support of topology specification and management. The aim is to allow, in virtualised form, various services such as security and analysis services deployed on these resources [7]. Specifically, the development of these architectures needs to

be supported through orchestration based on topology patterns reflecting common and reference architectures.

Several technologies exist that might contribute to the solution:

- Application packaging through containerisation: Containers can be used to distribute service and applications (sometimes called appliances) to the edge. Docker has already been used to do this (providing plugins link to agents which could be Docker runtimes).
- Programmability: Orchestration can be supported through topology specification based on TOSCA topology patterns [1]. Overall, service composition (orchestration) needs to cover the whole life-cycle – deploy, patch, shutdown. Operations are mapped to cloud infrastructure management and a TOSCA engine runs on top of here edge cloud infrastructure [2].

We now specifically look at edge clouds and fog computing from a PaaS perspective taking lightweight application packaging and topology specification into account.

III. Virtualisation Principles

Virtualisation is an answer to the need for scheduling processes as manageable container units. Processes and resources in this context are at the operating systems level the file system, memory or the network. We review these technology principles in order to allow us to judge the capability of the technology for edge clouds later on.

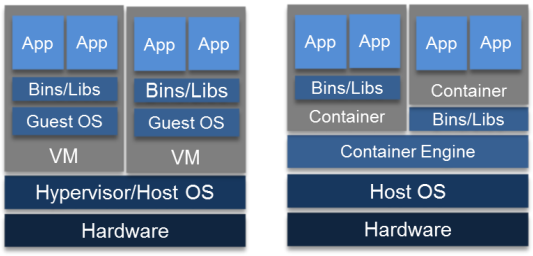

Fig. 2. VM (left) and Container (right) Virtualisation Architecture.

In the cloud as a virtualised architecture, virtual machines (VMs) are the core virtualisation mechanism. We briefly review the development of virtualisation over the years.

VMs have been improved over the years by enhancing scheduling, packaging and resource access (security). VM instances as guests use isolated large files on their host to store their file system and to run a single, large process on the host. Here, some concerns such as security are addressed through isolation. However, limitations remain. For instance, full guest OS images are needed for each VM in addition to binaries and libraries necessary for the applications, which is a space concern that means additional RAM and disk storage requirements. It also causes performance issues as this is slow on startup (boot), see Fig. 2. Furthermore, multi-tenant clouds require the sharing of disk space and CPU. In a virtualised environment, this has to managed such that the underlying platform and infrastructure can be shared in a secure, but also portable and interoperable way [14].

At the platform service level, packaging and application management is an additional requirement. Containers can match these requirements, but a more in-depth elicitation of specific concerns is needed.

Container technology is a development that meets the needs. A container is essentially a packaged self-contained, ready-to-deploy set of parts of applications, that might even include middleware and business logic in the form of binaries and libraries to run the applications [17], see Fig. 2. A typical example is a Web interface component with a Tomcat server. Container tools like Docker are frameworks built around container engines [18]. They make containers a portable way to package applications to run in containers. In terms of a tiered application, a container includes an application tier (or node in a tier). Two challenges remain, however:

- Managing dependencies between containers in multi-tier applications is a problem that emerges. As discussed, an orchestration plan can describe components, their dependencies and their lifecycle. A PaaS cloud can then enact the workflows from the plan through agents (which could be a container runtime engine). Software platform services can support packaging and deployment of applications from containers.
- The second challenge is to define, deploy and operate cross-platform capable cloud services using a lightweight virtualisation mechanism such as containers [13]. There is also a need to transfer cloud deployments between cloud providers in a distributed context, which requires lightweight virtualised clusters for container orchestration. Some PaaS are lightweight virtualisation solutions in this sense.

IV. Container Virtualisation for Lightweight Application Packaging

The evolution of virtualisation has resulted in more lightweight solutions. This is specifically relevant for application packaging at a software platform and application level. Recent virtualisation advances have improved multi-tenancy capabilities, i.e., the capability to share a resource in a cloud.

A. LXC Linux and Docker Containers

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 [17]. These are examples of OS virtualisation advances.

- Namespace isolation allows groups of processes to be separated. This ensures that they cannot see resources in other groups. Different namespaces are the used for process isolation, network interfaces, access to inter-process communication, mount-points or for isolating kernel and version identifiers.
- cgroups (control groups) manage and limit resource access for process groups through limit enforcement, accounting and isolation, e.g., limiting the memory available to a specific container. This enables better isolation between isolated applications on a host. This restricts containers in multi-tenant host environments.

Control groups allow sharing available hardware resources between containers and, if required, setting up limits and constraints.

Containers are, as a consequence of these properties, virtualisation techniques 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.

There is still an ongoing evolution of OS virtualisation and containerisation, aiming at providing OS support through standard APIs and tools for container management, network management and making resource utilisation more visible and manageable.

Docker is a container solution that builds on top of Linux LXC techniques. Docker is the most popular container solution at the moment and shall be used to illustrate containerisation. A Docker image is made up of file systems layered over each other, similar to the Linux virtualisation stack, using the LXC mechanisms, see Fig. 3. A container-aware daemon, called systemd, starts containers as application processes. It plays a key role as the root of the user's process tree.

- Boot process: In a traditional Linux boot, the kernel first mounts the root file system as read-only, before checking its integrity. It then switches the rootfs volume to read-write mode. 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. Using union mount, several file systems can be mounted on top of each other. This property can be used to create new images by building on top of base images. Each of these file system layers is a separate image loaded by the container engine for execution.
- Container: Only the top layer is writable, which is the container itself. The container can have state and is executable. It is a kind of directory for everything needed for execution. While they are normally stateful, containers can be made into stateless images to be reused in more complex builds.

A typical layering could include, from top to bottom (Fig. 3), a writable container image for applications, an Apache image and an Emacs image as sample platform components, a Linux image (a distribution such as Ubuntu), and the rootfs kernel image. Containers are based on layers composed from individual images built on top of a base image that can be extended. Complete Docker images form portable application containers. They are also building blocks for application stacks. This approach is called *lightweight* as single images can easily be changed and distributed.

B. Application Containerisation and Container Management

Fig. 3 shows a sample containerised application, the writable container. Containers can encapsulate a number of application components through the image layering and extension process. Different user applications and platform components can be combined in a container. Fig. 4 is an illustration of different scenarios using the container capability of combining images for platform and application components.

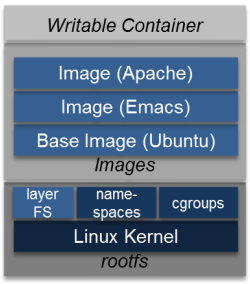

Fig. 3. Container Image Architecture.

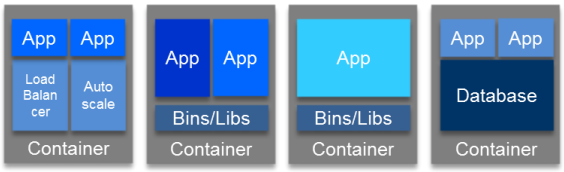

Fig. 4. Container-based Application Architecture.

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.

- These *repositories* play a key role in providing access to possibly reusable private and public container images, which might be tens of thousands. Examples of popular images are platform components such as MongoDB or Node.js.
- The *container API* supports life-cycle operations like creating, defining, composing, distributing containers, running/starting images and running commands in images.
- *Container creation* for applications is done by assembling them from individual images, which can be base images extracted from repositories.

In addition to these basic features, storage and network management are two specific functions that containers as application packages for distributed edge clouds also require.

Firstly, there are two ways data is managed in Docker – 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 features for persistent or shared data. Volumes can then be shared and reused between containers (Fig. 5). 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

TABLE I. VM VS. CONTAINER COMPARISON.

	VMs	Containers
Standardisation	Fairly standardised system images with capabilities similar to bare-metal computers.	Not well standardised, OS- and kernel-specific with varying degrees of complexity.
Host/guest architecture	Can run guest kernels that are different from the host, with consequent more limited insight into host storage and memory management.	Run host kernels at guest level only, but can do so possibly with a different package tree or distribution such that the container kernel operates almost like the host.
Boot process	Started through standard boot process, resulting in a number of hypervisor processes on the host.	Can start containerised application directly or through container-aware init daemon like systemd. These appear as normal processes on the host.

TABLE II. CONTAINER MODELS [ADAPTED FROM [9]].

OS	Container Models
Linux	Docker, LXC Linux containers, OpenVZ (and others for variants such as BSD, HP-UX, Solaris)
	We have discussed LXC and Docker in detail. OpenVZ is also container-based virtualization for Linux. OpenVZ puts more emphasis on security, but in contrast to LXC requires additional patches to the (vanilla0 kernel to operate.
Windows	Sandboxie
	- Provides application isolation support for Windows environments. A sandbox is a container placed around an application. In this sense, virtual machines are sandboxes that emulate a complete host computer.
Cloud PaaS	Warden/Garden (in Cloud Foundry):
	- Warden provides an API for managing containers (isolated environments). Warden provides a service for managing a collection of containers and defines a protocol for clients to send requests to and receive responses from the server. Containers can be limited in terms of CPU usage, memory usage, disk usage, and network access. Warden builds on Ruby.
	- Garden is a re-coding of Warden in Go. Garden provides the container technology for Diego (the future architecture for Cloud Foundry).
	- LXC and Docker (in Openshift):
	Openshift gears area kind of containers. Openshift gears are not traditional LXC style Linux containers that e.g., Docker relies on. Openshift combines gears, SELinux and Docker to put a stronger emphasis on security (avoiding identification problems in namespaces)

and container linking. Applications connect to an application running 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.

C. Comparison

In order to summarise traditional VMs and containers, we compare the two technologies in Table I.

We have used Docker earlier on to illustrate some container concepts, but a range of other container technologies exist for different operating systems types. For widely used OS like Linux and Windows several ones exist, but also specific solutions for PaaS, see Table II) [9]. Common to all is providing isolation for applications to address security problems.

The tool landscape supporting containerisation is equally in evolution. As one example, Rocket is a new container runtime from the CoreOS project (CoreOS is a Linux derivate for massive server deployments). Rocket is an alternative to the Docker runtime. It is specifically designed for composability, security, and speed – important properties in the edge cloud domain. The concerns specifically addressed by Rocket are good examples of ongoing concerns.

V. PAAS CLOUDS AND CONTAINERISATION

VMs are today the format to provision platform and application components at the infrastructure layer. Containers, however, appear as a highly suitable technology for application packaging and management in PaaS clouds. 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 [6].

A. Evolution of 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, 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.

- The first PaaS generation included classical fixed proprietary platforms such as Azure or Heroku.
- The second PaaS generation was built around open-source solutions such as Cloud Foundry or OpenShift that allow users to run their own PaaS (on-premise or in the cloud), already with a built-in support of containers. Openshift moves now from its own container model to the Docker container model, as does Cloud Foundry through its internal Diego solution.
- The current third generation of PaaS includes platforms like Dawn, Deis, Flynn, Octohost and Tsuru, which are built on Docker from scratch and are deployable on own servers or on public IaaS clouds.

However, open PaaS platforms like Cloud Foundry and Openshift treat containers differently. Cloud Foundry supports stateless applications through containers, but stateful services run in VMs. Openshift on the other hand does not distinguish between them.

Flynn and Deis, as two examples of the third generation PaaS, have created a micro-PaaS concept where small PaaS can be run on limited hardware with very little overhead. They have adopted elements of CoreOS for their clustered, distributed architecture management, building on the mechanism of lightweight, decoupled services facilitated by Docker. This aids distributed multi-tenancy cloud on reduced capability resources.

B. Service/Microservice Orchestration

Recently, microservice architectures have been discussed, which aim to break up monolithic application architectures into SOA-style independently deployable services, which are well supported by container architectures. Services are loosely

coupled, independent software components that can be rapidly called and mapped to any business process are required.

The microservices architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms. Microservices are independently deployable, usually supported by a fully automated deployment and orchestration framework. They require the ability to deploy often and independently at arbitrary schedules, instead of requiring synchronized deployments at fixed times.

The microservice development and architecture concerns are central PaaS concerns. Containerisation provides an ideal mechanism for their flexible deployment schedules and orchestration needs, particularly, if these are to be PaaS-provisioned.

VI. Clustering and Orchestrating Container

The next concern is to facilitate the step from a single container host to clusters of container hosts to run containerised applications over multiple clusters in multiple clouds in order to meet the edge cloud requirements [10]. The built-in interoperability of containers can make this possible.

A. Container Clusters

A container-based cluster architecture groups hosts into clusters [8]. Fig. 5 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.

Each *cluster* consists of several (host) nodes – where nodes are virtual servers on hypervisors or possibly bare-metal servers. Each (host) node holds several containers with common services such as scheduling, load balancing and applications. Each *container* in a cluster can hold continually provided services such as their payload service, so-called jobs, which are once-off services (e.g., print), or functional (middleware service) components. Next, *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 require data persistence. Containers can mount volumes. Data stored in these volumes persists, even after a container is terminated. Finally, *links* allow two or more containers to connect and communicate.

Resulting from this architectural scenario is an abstraction layer for cluster-based service management that is different from the container features provided by for instance 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 through containers is supported using a virtual scalable *service node (cluster)*, with high internal complexity (supporting scaling, load balancing, failover) and reduced external complexity. 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 (at each host). A *cluster* head node service is the master that receives commands

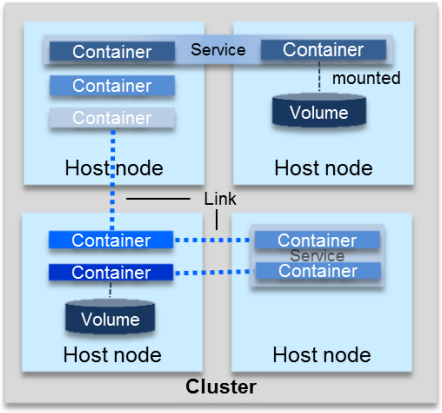

Fig. 5. Container-based Cluster Architecture – an architectural framework based on common concepts.

from the outside and relays them to container hosts. This allows development of for instance edge cloud architecture without consideration of the underlying network topology and avoids manual configuration [6].

A cluster architecture is composed of engines to share service discovery (e.g., through shared distributed key value stores) and orchestration/deployment (load balancing, monitoring, scaling, and also file storage, deployment, pushing, pulling). This satisfies some requirements put forward for these cluster architectures by Kratzke [9]. A lightweight virtualised cluster architecture building on containerisation should, according to Kratzke, 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 and scalable service discovery and orchestration,
- Transfer/migration of service deployments between clusters.

Similar to Docker, Diego, Warden and others in the container space, several products have emerged in the cluster space. One such cluster management platform is Mesos – an Apache project that binds distributed hardware resources into a single pool of resources. These resources can be used by application frameworks to manage workload distribution. It is a distributed systems kernel following the same principles as the Linux kernel, but here at a different level of abstraction. The Mesos kernel runs on all machines in the cluster. It facilitates applications with APIs for resource management and scheduling across cloud environments. In terms of interoperability, it natively supports LXC and also Docker.

Another clustering management solution, albeit at a higher level than Mesos, is the Kubernetes architecture. Kubernetes, which is supported by Google, 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. So-called minions are container hosts that run pods, which are sets of containers on the same host.

Openshift is a PaaS example that has adopted Kubernetes. Kubernetes competes here with the platform-specific evolution towards container-based orchestration. Cloud Foundry, for instance, uses Diego as a new orchestration engine for containers.

Clustered containers in distributed systems require advanced network support. Containers provide an abstraction that makes each container a self-contained unit of computation. 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 in addition to the network aspect. Managing containers in Kubernetes clusters might cause difficulties with regard to flexibility and efficiency because of the need for the Kubernetes pods to co-locate with their data. What is needed is to combine a container with a storage volume that follows it to the physical machine, regardless of the container location in the cluster.

B. *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 [19]. The topology of distributed container architectures needs to be specified and its deployment and execution orchestrated, see Fig. 6.

There is currently no widely accepted solution for the orchestration problems. We briefly illustrate the significance of this problem through a possible solution that we want to propose here as 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. Cloudify uses TOSCA (Topology and Orchestration Specification for Cloud Applications [1]) to enhance the portability of cloud applications and services, see Fig. 6. TOSCA supports a number of features:

- the interoperable description of application and infrastructure cloud services – here implemented as containers hosted on nodes in an edge cloud,
- the relationships between parts of the service – here service compositions and links as relationships, as illustrated in Fig. 5,
- the operational behaviour of these services (such as deploy, patch or shutdown) in an orchestration plan.

This has the advantage of being independent of a supplier creating the service and also any particular cloud provider or hosting technology. TOSCA can also be used to associate higher-level operational behaviour with cloud infrastructure management. TOSCA templates can be used to define container clusters, abstract node and relationship types, and application stack templates.

Cloud applications can run in TOSCA containers – enacted through a TOSCA engine based on TOSCA topology and orchestration descriptions [2]. The topology specification is based on nodes (e.g., Web server or sensor/device) and has life-cycle interfaces, which allows an architect to define how to create, configure, start, stop and delete resources. An orchestration plan is defined in YAML. The orchestration plan is used to orchestrate the deployment of applications as well as the post-deployment automation processes. The orchestration plan describes the applications and their lifecycle, and the relationships between components. This includes the connections between applications and where they are hosted – features that we have already discussed as vital for the edge cloud environment. With TOSCA, we can describe the infrastructure, the platform middleware tier and application layer on top of these. A PaaS product like Cloudify supporting TOSCA would take a TOSCA orchestration plan and then enacts this using workflows that traverse the graph of the components and issues commands to agents. These agents create the application components and glue them together. The agents use extensions, called plugins, which are adaptors between for instance a Cloudify configuration and the various infrastructure as a service (IaaS) and automation tool APIs.

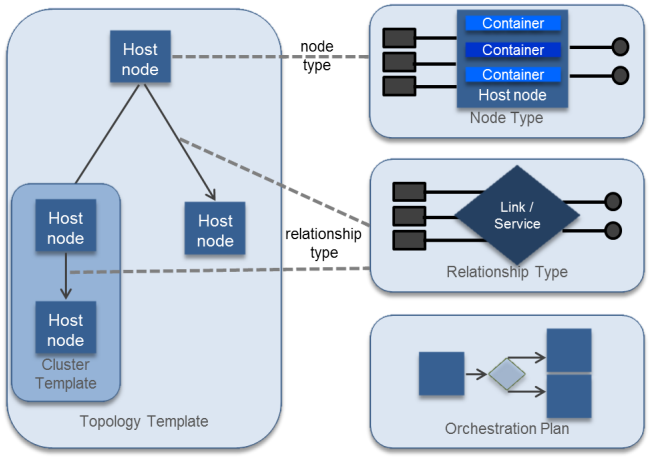

Fig. 6. Reference Framework for Cluster Topology Orchestration [adapted from the TOSCA standard to an edge cloud setting].

TOSCA framework and engine provide an orchestrator tool that allows the description of complex topologies and deployments. A dedicated topology and orchestration specification allows to specify the deployment of each service in the most ideal way. For simple stateless microservices, the best approach, for instance, is to use Docker and Kubernetes. A TOSCA blueprint can be used for more complex topologies that require more orchestration. Examples here could include a replicated and sharded mongo DB cluster or a more complex microservice. Though, a TOSCA blueprint can also be used for the basic container case to, for example, spawn a number of instances of a Docker image.

Currently, TOSCA deals with container technologies, such

as Docker and Rocket, as well as container management technologies, such as Kubernetes and Mesos, in an agnostic way. Due to this abstraction and interoperability, TOSCA is a suitable platform for defining a standard container orchestration specification that is portable across various cloud environments and container providers – which is, despite the current success and leadership of for instance Docker, a valuable property.

In another direction that singles TOSCA out, there is a growing interest in NFV (Network Functions Virtualization) within the TOSCA community – an aspect at the infrastructure level (cf. earlier SDN comments) due to our focus on application and service management at the platform level.

VII. Conclusion

Edge clouds move the focus from heavy-weight data centre clouds to more lightweight virtualised resources, distributed to bring services to the users. They do, however, create 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 here was to start with the recently emerging container technology and container cluster management to determine the suitability of these approaches for edge clouds through a technology review. The observations here support the current enthusiasm for this technology, but have also identified limitations. 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 defacto standards like Docker or Kubernetes is also happening, though currently at a slower pace. Thirdly, development and operations are still at an early stage, particularly if complex orchestrations on distributed topologies are in question.

We can observe that cloud management platforms are still at an earlier stage than the container platforms that they build on [10], [12]. 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. Container technology has the potential to substantially advance PaaS technology towards distributed heterogeneous clouds through lightweightness and interoperability. However, 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 implemented cluster solutions, is ultimately not sufficient.

Acknowledgments

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie) and by the Irish Centre for Cloud Computing and Commerce (IC4), an Irish national Technology Centre funded by Enterprise Ireland and the Irish Industrial Development Authority.

References

[1] T. Binz, G. Breiter, F. Leymann, T. Spatzier, Portable Cloud Services Using TOSCA, IEEE Internet Computing, vol. 16, no. 3, pp. 80-85, 2012.

[2] T. Binz, U. Breitenbcher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner. "OpenTOSCAa runtime for TOSCA-based cloud applications." In Service-Oriented Computing, pp. 692-695. Springer, 2013.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. "Fog computing and its role in the internet of things." In Proceedings MCC workshop on Mobile cloud computing, pp. 13-16. ACM, 2012.

[4] A. Chandra, J. Weissman, and B. Heintz, " Decentralized Edge Clouds", IEEE Internet Computing, 2013

[5] B. Di Martino, "Applications Portability and Services Interoperability among Multiple Clouds", IEEE Cloud Computing , vol. 1, no. 1, pp. 74-77, 2014.

[6] 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.

[7] 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, Technical Report, 2013.

[8] V. Koukis, C. Venetsanopoulos, N. Koziris, "∼okeanos: Building a Cloud, Cluster by Cluster", IEEE Internet Computing, vol. 17, no. 3, pp. 67-71, 2013.

[9] 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.

[10] J.P. Martin-Flatin, "Challenges in Cloud Management", IEEE Cloud Computing, vol. 1, no. 1, pp. 66-70, 2014.

[11] 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.

[12] E. Miluzzo, "I'm Cloud 2.0, and I'm Not Just a Data Center", IEEE Internet Computing, vol. 18, no. 3, pp. 73-77, 2014.

[13] T.H. Noor, Q.Z. Sheng, A.H.H. Ngu, S. Dustdar, "Analysis of Web-Scale Cloud Services", IEEE Internet Computing, vol.18, no. 4, pp. 55-61, 2014.

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

[15] R. Ranjan, "The Cloud Interoperability Challenge", IEEE Cloud Computing, vol. 1, no. 2, pp. 20-24, 2014.

[16] B. Satzger, W. Hummer, C. Inzinger, P. Leitner, S. Dustdar, "Winds of Change: From Vendor Lock-In to the Meta Cloud", IEEE Internet Computing, vol. 17, no. 1, pp. 69-73, 2013.

[17] S. Soltesz, H. Pötzl, M.E. Fiuczynski, A. Bavier, 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.

[18] J. Turnbull, The Docker Book. http://www.dockerbook.com/. 2014.

[19] 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.

[20] 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." In IEEE 7th International Symposium on Service Oriented System Engineering (SOSE), pp. 320-323. IEEE, 2013.