Using Linux Containers in Telecom Applications

Csaba Rotter, Lóránt Farkas, Gábor Nyíri
Technology and Innovation, Research
Nokia Networks
Budapest, Hungary
{csaba.rotter,lorant.farkas,gabor.nyiri}@nokia.com

Gergely Csatári, László Jánosi, Róbert Springer
MBB Liquid Core
Nokia Networks
Budapest, Hungary
{gergely.csatari,laszlo.janosi,robert.springer}@nokia.com

Abstract—Container Technology is one of the most hyped virtualization technologies in the last couple of years. It enables not only higher application density on the same HW environment compared to hypervisor-based technologies but it also gives performance benefits regarding to starting and stopping applications. Container technologies are not new, but more recently an entire ecosystem has been built around them and also a lot of synergy has been created with the currently have hypervisor-based cloud technology. Telecom applications exhibit strong performance and high availability requirements, therefore running them in containers requires additional investigations. This article targets to present the container ecosystem from this angle. In particular we are looking at the way how the currently available technology can be used and extended to meet the requirements of telecom applications.

Keywords—component; formatting; style; styling; insert (key words)

I. MOTIVATION ON USING LINUX CONTAINERS

Hypervisor-based virtualization and container technology is similar that perspective that both of them ensure an environment for application isolation. This isolation is important from resource access and usage point of view. This is especially important when two applications are running on the same hardware environment and they are competing for the same type of resources. This resource can be CPU time, memory, disk I/O, network I/O, etc. Hypervisor-based virtualization offers a higher level of isolation especially in case of Type 2 hypervisors when the application is running on the top of guest operating system through the host operating system. This greater isolation is on cost of greater overhead. Reducing this overhead is the biggest advantage of container technology and allows an application to start even 100 or 1000 times faster than in case of hypervisor- based virtualization. Docker and Rocket are container commodity tools, they not only ease container creation but introduce the notion of container image and image repository. By doing this, container commoditization tools make not just the SW delivery but also the application delivery much faster. Continuous application delivery is also a benefit as result of the reduced time of SW delivery. This causes an increased agility based on reducing the provisioning time between development and testing. Container-based technologies offer a great level of flexibility by allowing to containerize either a whole system or just parts of the system together with the application. This is important from portability point of view, which means that the container could be so lightweight to contain only the application.

Containers are lightweight, regarding speed and size. Size is much smaller compared to virtual machines meanwhile the startup speed is noticeably faster as was told before.

It is a relatively new and developing area with all of the children's sickness of it. In some cases tools are not mature enough, or tools are in prototype phase, with poor or even contradicting documentation. Growing popularity and the very attractive features of containers are the driving factors for IT and telco world to build and provision mature and high performance applications on top of them.

In the recent decades Telecom Applications are deployed into more and more generic hardware and are less and less integrated to the hardware. As a result of this tendency there is a demand to be able to deploy the same application into several deployment variants, which is Advanced Telecommunications Computing Architecture (ATCA) or IT hardware or virtual machines in different clouds. Encapsulating the application into one or more Linux Containers provides the possibility to deploy the same application to all of these different options.

II. LINUX CONTAINER TECHNOLOGY LANDSCAPE

It is important to claim that this landscape is based on the evaluations done in the last months of 2015 so some information could be not accurate on the time of reading.

A. Container base technology

Container technology as such is an old technology. Old in that sense that some of the basic ingredients, like, *chroot*, was part of Unix distributions as early as 1979. Chroot [1] is a Unix operation which enables to change the apparent root directory of the current running process. In 2004 Parallels, formerly known as SWsoft, released OpenVZ [2], an advanced container-based virtualization, with sophisticated features like resource management and live migration. The reason of not being so popular was that it required kernel patches which is not that straightforward to perform for a generic Linux user. In 2006, Google released cgroups [3] and that was so successful that in 2007 Google containerized the entire search application and later all the Google applications were ported on top of

containers [4]. In 2011 a Container Unification Agreement was signed at the Kernel Summit where all the major container technology providers agreed to have just one Kernel API and only one underlying technology will stand behind, which is based on cgroups and namespaces. The main reason for this agreement was that technology providers wanted to avoid similar problems occurring in the past caused by missing agreements between KVM and XEN. The first Linux kernel supporting OpenVZ without patches is 3.12 and from this version onwards container technology is included into kernel functionality. Cgroups was started in 2006 by Google engineers and merged into upstream 2.6.24 kernel for larger audience of LXC (LinuX Containers, not to be confused with the LXC user space toolset) usage.

Comparing the architecture of hypervisor-based virtualization to container-based virtualization reveals the lightweight nature of containers. Hypervisor-based virtualization is based on emulating the hardware but containers are based on different approach. Containers are based on sharing the operating system like in Fig. 1. This means that instead of running a hypervisor and a guest operating system and the applications on top of the guest operating system, the applications running in container share the kernel of the host operating system. They can also share additional user space libraries and binaries, this is just a matter of configuration.

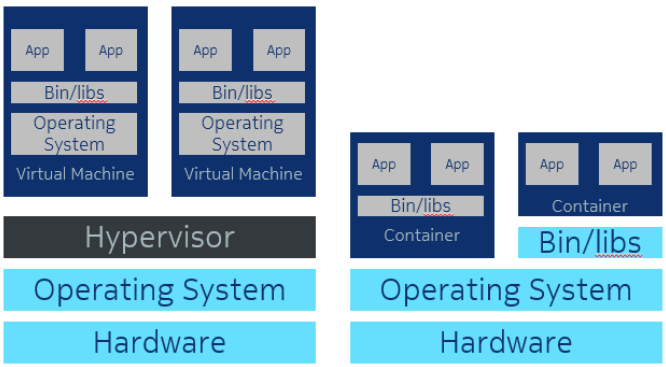

Fig. 1. Hypervisor based versus container based technology

Since in case of container-based virtualization there is no need to start an additional guest operating system and additional libraries and binaries but only to use and share an already running kernel, the startup speed of the applications is very high. According to [5] Docker equals or exceeds KVM performance in every case tested. Regarding to comparison it is worth to mention that meanwhile containers offer high level of density and allow dynamic resource allocation and have almost native bare metal performance, still it is a great disadvantage that one hardware cannot run applications together that share different kernel versions or kernel modules, because all applications share exactly one kernel, that of the host operating system.

1) cgroups

As it was shown cgroups is at the basis of any current container technology and it is included into modern Linux kernels. Cgroups also represent the basis of system resource management. In order to understand the cgroups model we need to understand the Linux process model [6].

In the traditional Linux process model all the processes are child processes of the *init* process and *init* is executed by the kernel at boot time, which means the process model is single hierarchy. Init process always has one as process identifier and this cause problems when starting several init processes. Cgroups implementation allows to a started init process to believe that it is running with a process id (pid) equal to one but the original init process will see it running with a different pid. This means that the cgroups model is similar to the process model because it is a single hierarchy too but the fundamental difference is that many hierarchies can exist at the same time with several separate unconnected trees. Detailed cgroups description can be found on kernel.org cgroups documentation [7].

2) Namespaces

The Linux kernel provides process level isolation by creating separate namespace for containers. Namespaces allow to create an abstraction of a global system resource to a process and makes it to appear as a separate instance to a process running on the namespace. As result several containers can use the same resources at the same time without any conflict [8]. Processes in the same namespace internally perform as if they were the only processes in the system, they don't see other processes outside their namespace.

Linux Security Modules [9] and Mandatory Access Control [10] are also key components of container creation especially with regards to security. With Linux Capabilities [11] we can set per process privileges to system call access. This also increases the container security.

In order to build a container we need to add resource management functionality to a process or process group by using cgroups, we need process isolation by using namespaces, to change the apparent root directory and we need to enforce security settings. This can be done by using command line tools and this is not very straightforward for an average Linux user. This process has been successfully automated and encapsulated by Docker developers so that the details are hidden from the average Linux user. As a result Docker container creation became as easy as a virtual machine creation with hypervisor-based technology, one of the key reasons Docker gained such a significant traction.

3) LXC Toolset

LXC is a low level but still flexible set of tools, templates, libraries and language bindings and it covers almost every containerization feature supported by upstream kernel. LXC Project [32], project supported by Canonical Ltd provides tools to manage containers, networking and storage functions. The only disadvantage is that it has heavy support focus on Ubuntu, without extensive documentation and as such the cross distribution functionality is not always straightforward.

4) Docker

As was mentioned earlier the basic container technology is not really new. The question is then how could Docker make container technology so popular. Docker developers realized that not just container creation is difficult but also it is not easy

to persistently save a container content like in case of virtual machines. The below figure shows the basic difference between Linux containers (left side) and Docker containers (right side), which is a Linux container created by Docker.

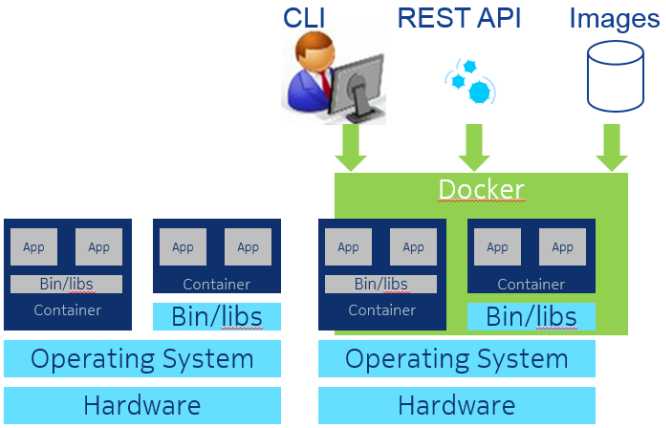

Fig. 2. Linux Container versus Docker Container

As we can observe from the Fig. 2 Docker added a command line interface to container creation, a REST API and most importantly an image repository to store modified or preinstalled container images. These images conceptually are very similar to VM images as they store a snapshot of the installed application but the main difference is that when a user modifies this image and wants to store the modified image, then only the modification is stored and not the entire modified image. This is important from that perspective that an image modification can be used quickly not just on the place of modification but pushing it to a central repository is much easier because the total size of the content committed to repository is much smaller than committing the full modified image. This process can be seen in Fig. 3. There are public and private registries in order to upload and download container images, the public Docker registry is called Docker Hub. A Docker container holds everything that is needed for an application in order to run.

Docker consists of a Docker daemon and one or more Docker clients, Docker clients can initiate operations like pull or run Docker images by Docker daemon and the daemon itself will actually create the containers on the host where the daemon is installed. The user does not directly interact with the daemon but through the Docker client. The Docker client is the Docker binary and is the primary user interface to Docker.

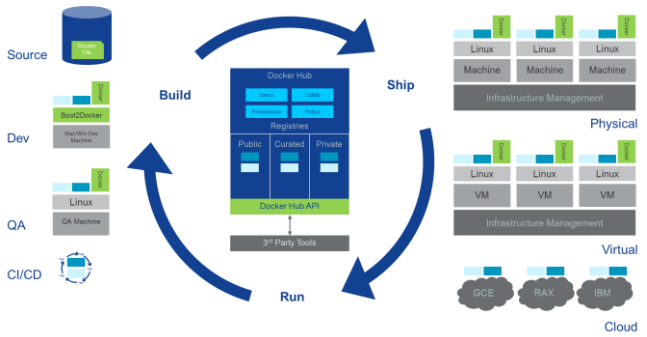

Fig. 3. Docker image registry usage [39]

In early development of Docker container creation LXC was the default container creation interface [37]. Both LXC and also Docker development was very intensive lately and Docker decided to go independent from LXC and started to use libcontainer, which is a native Go implementation of Linux container creation based on cgroups, namespaces, etc.. LXC is just an option to use in Docker. As can be seen in Fig. 4 Docker can use LXC, libvirt or recently developed libcontainer, which become the default option from Docker 0.9 onwards. It is important to see that lot of big players like RedHat, Canonical, Parallels and Google also embraced libcontainer as the default container access interface.

Wide industry support also contributed to growing Docker popularity, Docker becomes a de-facto industry standard on container management.

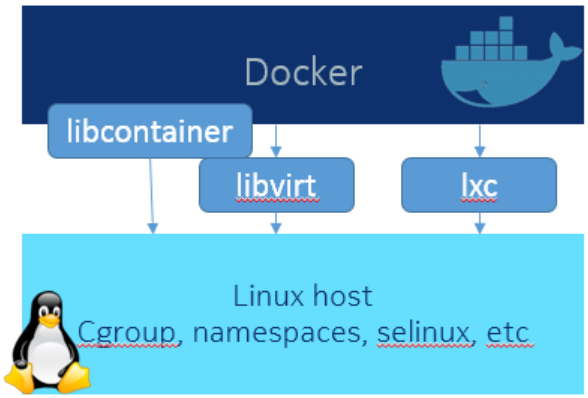

Fig. 4. Docker libcontainer usage

In early stages of Docker development the biggest concern from the industry was container security. As it was written also by The Register [38], Gartner says “Linux containers are mature enough to be used as private and public PaaS”.

As the kernel API towards container operations is open, gives the opportunity to other industry players to create their own container commoditization software. One other very important player in this topic is Rocket

5) Rocket (rkt)

When CoreOS [13] started to invest in rkt development they addressed composability, security and speed as a key differentiator factor to other container management tools. Rocket is a very modern but fresh tool and it is integrated to systemd and to cluster orchestration tools. Rocket architecture is simpler than Docker, does not have client server entity, but rkt binary in every node. In addition Rocket claims compatibility to other container software, for example rkt can run Docker images. Rocket is not yet mature, according to their roadmap [14] in 2016 February is planned to be released the 1.0 version of rkt.

B. Orchestration

In case of an application, which is running in multiple containers it is important that an entity will keep track of the

containers belonging to the same application and to deal with network connection between them. In case of Docker and within one node Docker takes care of network connection but in case of hundreds or thousands of nodes a reliable orchestration is needed.

1) Kubernetes from Google

The most prominent Docker container orchestration system is Kubernetes [15] and is created and open sourced by Google. Kubernetes is using the concept of labels and pods to group application into logical units for manage them and for easier discovery. According to Google, Kubernetes is "an ocean of user containers" and containers are "Scheduled and packed dynamically onto nodes"

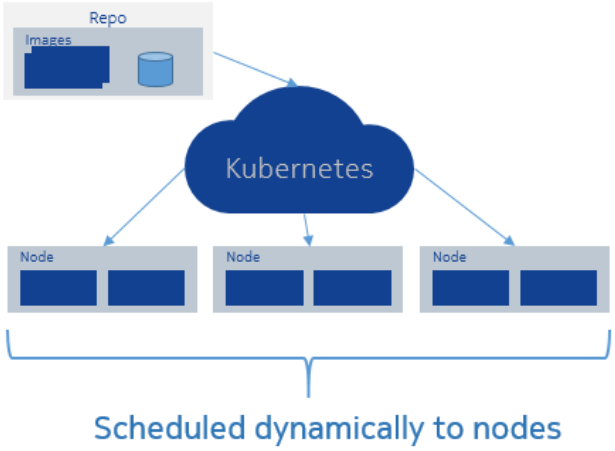

Fig. 5. Kubernetes basic scheduling

Kubernetes supervise and take care the life cycle management of containers. It is important to mention that Kubernetes does not take care of the state of the application. It manages only the container state, in case the application dies restarts the container for example. In case the application state is needed to be considered than that should be handled inside the application. Kubernetes also allows to set affinity and anti-affinity rules, which mean that it is possible to define that containers to be or not to be placed onto one container host. Kubernetes also includes support for Rocket containers.

Dealing with network is not directly done by Kubernetes. As one option Flannel [16] is used to do network connection between Docker hosts by using an overlay network, which is an IP network on top of existing UDP packets. Etcd is used in order to configure and operate Flannel. Etcd [17] is a distributed key-value store written in go and coming from CoreOS, it has a REST interface and it is very simple, secure, fast and reliable. It is important that etcd does not store user data but configuration data for reliable container operation.

2) Fleet from CoreOS

CoreOS also provides an orchestration system, called Fleet [18][19]. The basic philosophy on Fleet is to treat the CoreOS cluster as if it were a single init system. Users are encouraged to write small application units, which can be distributed around cluster of self-updating CoreOS machines. By using Fleet, DevOps team can focus on running containers that provides the service but does not have to worry about individual machines in the cluster. This is similar to Kubernetes from this perspective. Fleet is used in production for some time and can be considered stable [19].

3) Docker Swarm and Compose from Docker

Docker uses slightly different approach for container orchestration. Docker Swarm [20] is a native clustering for Docker containers. It collects together several Docker Engines and makes it look as one Docker Engine to the external world. This comes with a huge benefit of an unchanged Docker API. The disadvantage is that on the time of evaluation the API was not totally unchanged. For example we got HTTP 404 response for command build, rmi, pull because the "distributed" behavior of these commands was not cleared on that time.

Neither of the Container Orchestration system handles networking alone but manages container life-cycle and concentrates on the application running on top of cluster. Resource Management functionalities are also not part of orchestration systems. In order to consider resource management, these systems have to be used with resource managers like Mesos [20], Yarn [21] or DCOS [22] from Mesosphere. Adding resource management functionality is important on cases when multiple applications are running on top of the same infrastructure and they are competing for the same resources. Resource management will ensure that all the application running together receive their guaranteed resource in order to function according to promised performance.

C. Supporting Operating Systems and Ecosystem

Operating System providers also realized the potential in container technology and most of them designed a lightweight version of operating system, which offers minimal functionality but full support to containers, orchestration, and networking and for the full ecosystem. Project Atomic [23] sponsored by Red Hat offers an application centric IT infrastructure by providing a wide range of container support. Project Atomic hosts inherit the full feature of their base distribution, like systemd and journalctl. Project Atomic targets enterprise customers who are already running their application on Red Hat Enterprise, they offer Red Hat experience.

Ubuntu Core [24] is a system from Canonical for container deployment especially designed for Docker. Snappy Ubuntu Core is a mix of Ubuntu Core and the gathered experiences from Ubuntu's phone efforts [35]. Offers a high level of security due to AppArmor [25] to enforce strong isolation between applications. AppArmor is a Mandatory Access Control [10] system to set minimal required resource usage for applications.

The ChromeOS fork, CoreOS [26] is an open-source lightweight operating system designed for clustered deployments. ChromeOS has committed to Rocket support but also support Docker. As a container orchestration it uses Fleet as it was already mentioned.

Photon OS [27] is coming and open-sourced from VMware and clearly targets VMware customers. It is optimized for

vSphere and it has strong container support, supports Docker and Rocket too.

Fig. 6 summarize the container ecosystem, starting with container supporting Kernel features, Container commoditization tools and applications running on top of orchestration systems.

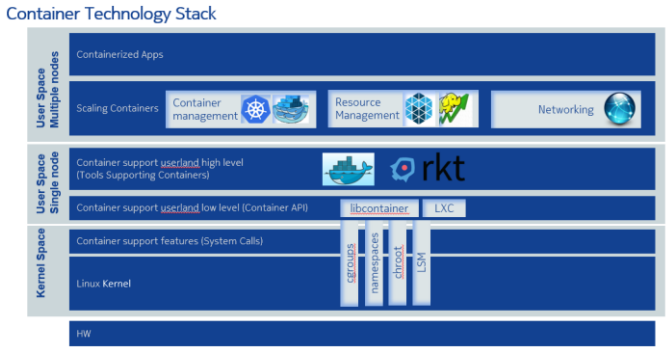

Fig. 6. Container Technology Stack

III. TELECOM APPLICATIONS AND THEIR PROPERTIES

With a slight modification of our [33] previous definition of Mobil Core Network Elements for Cloud it is possible to create a generic definition of a Telecom Application. A Telecom Application in the telecommunication world is a single or multiple node application responsible for a well-defined task in the telecommunication network. A Telecom Application uses standardized interfaces to connect to other network elements and implements standardized functions. On top of the standardized functions a telecom application can have vendor specific functionality and can use vendor specific interfaces. One Telecom Application can integrate one or more standard functions.

A. Properties of Telecom Application

There is a set of properties which are generic to all Telecommunication Applications. The following list presents the most typical properties.

High availability: telecom networks need to operate with high availability. To achieve a network level availability either pooling of Telecom Applications is needed or the distinct Telecom Applications have to implement high availability features. As pooling of Telecom Applications is not standardized for every type of application and the pooling is not economical in small networks the Telecom Applications need to implement high availability features.

High capacity: Telecom Applications normally serve millions of users and millions of telecommunication session initiations per hour. To provide service continuity even in case of an overload situation Telecom Applications implement certain overload control mechanisms.

Application Cluster: To provide the required high capacity using reasonably small (virtual) computers and to implement the high availability features in most of the cases Telecom Applications are implemented as a cluster of (virtual) servers. The basic mechanism of High Availability is the capability to continue the execution of a functionality on a healthy server whenever the original place of execution fails. To support this failover capability there is a need to guarantee that two software components are not executed on the same (virtual) server, this need is defined in an anti-affinity rule.

Network separation: Due to network reliability reasons some telecom protocols are using SCTP (Stream Control Transmission Protocol) with multihoming feature as a transport protocol which requires the usage of multiple NIC-s (Network Interface Card) in a node. Also there are regulatory requirements to separate certain traffic types from each other what also requires the usage of multiple NIC-s.

IP Address Allocation: the operators of Telecom Applications use strict IP planning with preallocated IP addresses for certain functions. As a consequence of this it shall be possible to allocate fixed IP addresses to certain NIC-s. Also as IP replanning and reallocation of IP addresses might happen it should be possible to change the fixed IP address of a NIC.

B. Deployment alternatives with container orchestration

When Linux Containers with orchestration are used there is a possibility to use the container orchestration framework's features to implement the cluster management tasks of the Telecom Application. These cluster management tasks can be the following:

Membership management and service discovery: Some container management software, like Docker and some container orchestration software, like Kubernetes are emitting notifications related to the lifecycle events of the Containers. These events can be used to register or de-register the service implemented by the Container to the service discovery system of the Application or to execute some recovery action in case of a failure event.

Monitoring: Some container orchestration software, like Kubernetes has the capability to monitor both the state of the Container and the Application running in the Container.

Deployment of the Linux Containers and orchestration means the way how the software implementing, managing and orchestrating the Linux Containers is organized in the production environment. It defines the responsibility split between the company or organization delivers the software components related to Linux Containers and the company or organization delivers the Telecom Application executed on top of the Container related software components. In the Telecom industry there are two deployment alternatives of the Linux Container orchestration. We discuss in the subsequent the benefits and drawbacks of both deployment alternatives.

Embedded deployment: all container related software components are part of the Telecom Application, delivered by the provider of the Telecom Application who takes full responsibility for both the Telecom Application and the Container related software components. The provider of the Telecom Application provides the full stack to execute the Telecom Application. The responsibility of the Telecom

Application provider is marked with dashed line on the following figure (Fig. 7).

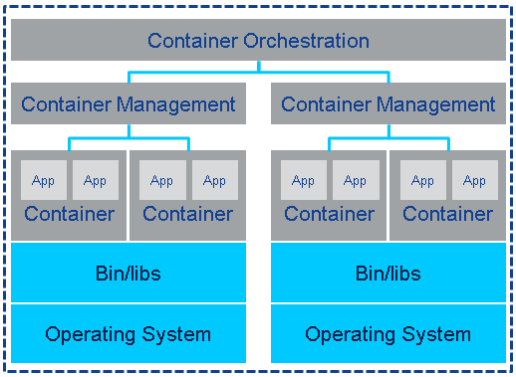

Fig. 7. Container Management and Orchestration

Provided deployment: container management and orchestration related software are provided by the company or organization which operates the Telecom Application. The provider of the Telecom Application provides only the Linux Containers implementing the functionality of the Telecom Application. The responsibility of the Telecom Application provider is marked with dashed line on the following figure (Fig. 8).

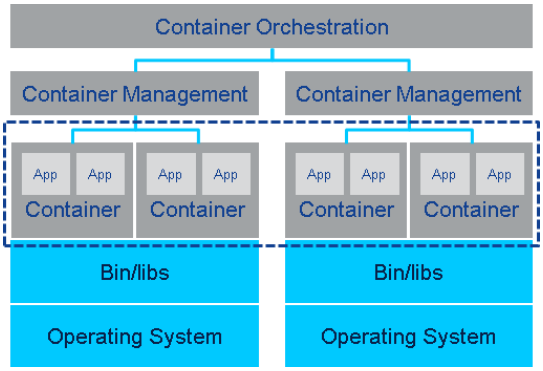

Fig. 8. Telco operator provided Application deployment

There are network operators in the Telecom Industry who are planning to build their own Linux Container execution environments and willing to provide the Container management and orchestration software components. These operators select the tools for these tasks and define their own architecture principles for Container management and orchestration. Telecom Applications executed in these provided deployment environments either implement the support for all variants of Container management and orchestration or implement their membership management, service discovery and monitoring features without the support of container management and orchestration software components.

Due to the high velocity of changes in the Linux Container management and orchestration arena Nokia Open TAS (Telephony Application Server) uses embedded deployment with an architecture which prepares for the support of more Container management and orchestration software components. During the transition to a Linux Container based architecture Nokia Open TAS also moves towards the direction of Microservices based architecture. Usage of Linux Containers fits very well into Microservices architectures and continuous delivery, but the usage of these technologies and technologies have no strict dependency on each other.

Our intention with this paper is not to introduce any novelty in Linux Container implementation, management or orchestration, but to show the place of the ecosystem to run complex telecom applications with strong emphasis on multinode environment.

C. *Deployment alternatives with virtualisation containers in ETSI Network Function Virtualisation (NFV)*

As [34] describes the different virtualisation technologies, like hypervisor based virtualisation and Operating System (OS) virtualisation, such as Linux Containers can be nested into each other.

Hypervisor based virtualisation on top of bare metal: This deployment alternative is the mostly used in the Telecommunication Virtual Network Functions (VNF). In this case the VNF Manager manages the lifecycle of the hypervisor based virtual machines.

OS virtualization on top of bare metal: As of today this setup is not supported by Telecom Applications in the ETSI NFV architecture due to the limited support in the NFV Infrastructures (NFVI). On the other hand the "bare metal" variants of telecom applications use this approach to hide the differences of the different deployment options from their internal architecture. The lifecycle management of the Linux Containers is either managed by the application or a central container infrastructure depending on if the Embedded or the Provided deployment model is used.

OS virtualization on top of hypervisor based virtualisation: With this deployment alternative it is possible to introduce the benefits of Linux Containers into the ETSI NFV infrastructure even without a Linux Container support from the NFVIs. Both NFVI-s and the VNFM recognizes and manages only the hypervisor based virtual machines. The lifecycle of the Linux Containers are managed by the application or a central container infrastructure.

Hypervisor based virtualization on top of OS virtualization: According to our current view this setup brings no benefit for Telecommunication Applications, therefore it is not used.

IV. Requirements for the Linux Container Ecosystem

As a consequence of the properties of the Telecom Applications listed in Chapter III there are requirements which are hitting the Container base technology, Container management and Container orchestration.

A. *Fixed IP addresses for containers*

It should be possible to define fixed IP address to a NIC of a container and it should be possible to change this fixed IP address later preferably without the restarting of the Container. To implement this the container orchestration shall be able to store the IP address in the descriptor of the system and shall be

able to provide the IP address to the container management layer. The container management layer shall be able to set the IP address to the interface. To implement the changing of the IP address without restarting the container, the container management shall be able to reconfigure the container and the surrounding networks to use the new IP address. The application specific IP address configuration should be handled by the application itself. Late IP Address provisioning

For Telecom Applications with centralized CM (Configuration Management) database and centralized CLI (Command Line Interface) or GUI (Graphical User Interface) the capability of late IP address provisioning to a container is also needed. Late IP address provisioning means, that a fixed IP address is going to be set to a NIC, but the IP address is not configured yet, and will be provisioned later. To implement this the Container orchestration shall be able to store an indication that the IP address will be added later and provide this information to the Container management. The Container management shall be capable to start the interface without an IP address and set the IP address later.

B. Multiple network interfaces

It should be possible to define several NIC-s to a Container. To implement this the Container orchestration shall be capable to describe more than one NIC-s in the descriptors of the system and should be able to provide the information about the several NIC-s to the Container management component. The Container management component shall be able to configure the several NIC-s and connect them to the correct network.

C. Dependence between Containers

The container orchestration component shall be able to handle dependences between Containers, thus it should be possible to define a starting order of the Containers. Without this feature the Telco Applications Containers shall be implemented in a way, that they are able to wait until all other Containers are started which from they depend. To implement the dependency feature the Container orchestration component shall be able to store the dependency information between containers in the descriptor of the system and should be able to start the containers according to the dependencies described.

D. Container and application level health monitoring

Due to the high availability requirements Telecom Applications monitor the health status of every layer of their software stack. They use HW watchdog or HW watchdog emulation to monitor the Operating System, use different process monitoring tools to monitor and maintain the started processes, use monitoring tools to monitor the status of distant nodes of the cluster and willing to use monitoring tools to monitor the health status of Containers and the Application components running in the Containers. Some of the Container management components, like Docker already provide a way to monitor the status of the containers [31], while some of the Container orchestration components, like Kubernetes already provide different ways to monitor the status of the applications in the Containers.

E. Container affinity and anti-affinity rules

Due to the Telco Application properties described in Chapter III there is a need for a possibility to define anti-affinity rules between the Containers. Even if Linux Containers are restarted in a milliseconds range the restart of a container is not equivalent with the continuous running of the container. There are containers executing service proxy or infrastructure management tasks which have to be run on every container host. During the restart the container loose its non-persistent data and the detection of the detection of the error, the decision making and the restart of the container can take too long time and cause disturbances in the services of the Telecommunication Application. When an anti-affinity rule is defined between any numbers of Containers, the Container orchestration component ensures that all of the mentioned Containers are started on different Container Hosts.

In case of an affinity rule is defined between Containers the Container orchestration component ensures that all of the Containers are started on the same Container Host.

F. Resource SLA requirements

In order to run multiple applications on top of the same infrastructure it is very important to ensure resource level guarantee for applications in case they are competing for the same resource type. This is especially true in multitenant environment where resource bottleneck could cause performance degradation of the provided service. Currently available resource managers [20][21] gives support mainly of CPU and memory but there are methods [28] on how to extend the currently available resource model to disk I/O and network I/O.

G. Common API-s for container management and orchestration

One way to solve the embedded or provided orchestration dilemma, described in Chapter III.B, would be to define and standardize a common API for all operations what are used on the Container Management and Orchestration software components by the (Telecom) Application. In this way only this common API should be supported by the Telecom Applications to implement the Container management and orchestration assisted membership management, service discovery and monitoring features. To implement this the existing and future Container orchestration and management components should provide the same API for their functionality needed for membership management, service discovery and monitoring features. Alternatively, an API adaptor component can solve this problem, similarly to libvirt (http://libvirt.org/) in case of hypervisors.

H. Support of the ecosystem components

Telecommunication vendors use a supported Linux distribution as the base OS (Operating System) of their products. In the last years the components needed to create, manage and orchestrate Linux Containers have been added to most of these Linux distributions. Also some Container optimized distributions were also created. In the Telecom industry the Applications and therefore their components shall

have a support period of 3-5 years. To enable the usage of the container optimized Linux distributions the support period of these shall be also extended to 3-5 years. Without this change the Telecom Applications are forced to use the "normal mainstream" Linux distributions.

V. INDUSTRY OUTLOOK

Container technology has a very promising future with its attractive performance results but not all components are in production ready state. There are lot of missing features like disk and network I/O guarantees which is especially true when background operations are also consuming the same resources [29]. These problems are expected to be solved in future and the focus is expected to be on scaling the application, on service decomposition of big monolithic applications using microservice architectures. Microservices [30] are the best examples to be used in containerized environment. This is especially true if the environment is extended with dynamic resource management functionalities. Besides the possibility to support several deployment options, the usage of Linux Containers are opening the way towards autonomous software management of the microservice components of the Telecom Applications either from a local or a global container repository.

VI. SUMMARY AND ADDITIONAL ASPECTS

In this paper we show an overview of Linux Container ecosystem and the usage of this ecosystem in Telecom Applications. The intention of this paper is not to show a quantitative analysis of Linux Container technology, but to show the effect of the latest technology on the architectural evolution of multinode Telecommunication Applications. The evolution of standardization of container technology is not touched, however „The Open Container Initiative" [36] is important to be mentioned, as it offers a standardized container format supporting by the most important container ecosystem providers.

REFERENCES

[1] Change root. https://wiki.archlinux.org/index.php/Change_root

[2] OpenVZ, Available: https://openvz.org/Main_Page

[3] Cgroups, Available: https://en.wikipedia.org/wiki/Cgroups

[4] Google:, "Everything at Google runs in a container" http://www.theregister.co.uk/2014/05/23/google_containerization_two_billion/

[5] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio, "An Updated Performance Comparison of Virtual Machines and Linux Containers", IBM Research Report, 2014.

[6] Init wiki, https://en.wikipedia.org/wiki/Init

[7] Kernel.org cgroups documentation, Available: https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[8] RedHat articles, Introduction to Linux Containers, August, 2015, https://access.redhat.com/articles/1353593

[9] Linux Security Modules. Available: https://en.wikipedia.org/wiki/Linux_Security_Modules

[10] Mandatory Access Control. Available: https://en.wikipedia.org/wiki/Mandatory_access_control

[11] Linux Capabilities, Linux Man Page. Available: http://linux.die.net/man/7/capabilities

[12] Libcontainer Github repository. Available: https://github.com/opencontainers/runc/tree/master/libcontainer

[13] Motivation behind Rocket. Available: https://coreos.com/blog/rocket/

[14] Rocket roadmap. Available: https://github.com/coreos/rkt/blob/master/ROADMAP.md

[15] Kubernetes, Available: http://kubernetes.io/

[16] Flannel Github repository. Available: https://github.com/coreos/flannel

[17] Etcd github repository, Available: https://github.com/coreos/etcd

[18] CoreOS Cluster. Available: https://coreos.com/using-coreos/clustering/

[19] Fleet github repository. Available: https://github.com/coreos/fleet

[20] Apache Mesos. Available: http://mesos.apache.org/

[21] Apache Yarn. Available: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[22] DCOS documentation. Available: http://docs.mesosphere.com/

[23] Project Atomic. Available: http://www.projectatomic.io/docs/introduction/

[24] Ubuntu Core. Available: http://www.ubuntu.com/cloud/tools/snappy

[25] AppArmor. Available: https://wiki.ubuntu.com/AppArmor

[26] CoreOS Homepage. Available: https://coreos.com/using-coreos/

[27] CoreOS wiki. Available: https://en.wikipedia.org/wiki/CoreOS

[28] T. V. Do, B. T. Vu, H. N. Do, L. Farkas, C. Rotter, and T. Tarjanyi,"Building Block Components to Control a Data Rate *in the Apache Hadoop Compute Platform," in Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, Feb 2015, pp. 23–29.

[29] Xuan Thi Tran and Tien Van Do and Nam H. Do and Lorant Farkas and Csaba Rotter, "Provision of Disk I/O Guarantee for MapReduce Applications", in Trustcom-BigDataSE-ISPA 2015

[30] Microservices, Available: https://en.wikipedia.org/wiki/Microservices

[31] Docker Runtime Metrics, Docker Documentation. Available: https://docs.docker.com/articles/runmetrics/

[32] LXC Project Homepage: Available: https://linuxcontainers.org/

[33] G. Csatari and T. Laszlo, "NSN Mobile Core Network Elements in Cloud," *in Communications Workshops (ICC), 2013 IEEE International Conference on*, June 2013, pp. 251–255

[34] ETSI NFV DGS/NFV-EVE004: https://docbox.etsi.org/ISG/NFV/Open/Drafts/EVE004_Virtualisation_technologies_Report/NFV-EVE004v050.zip

[35] Ubuntu Phone resources: http://www.ubuntu.com/phone/developers

[36] The Open Container Initiative,https://www.opencontainers.org/

[37] Docker libcontainer unifies Linux container powers, Zdnet article, Available: http://www.zdnet.com/article/docker-libcontainer-unifies-linux-container-powers/

[38] Docker's just a bit dodgy, but ready for rollout says Gartner, The Registers, Available: http://www.theregister.co.uk/2015/01/12/docker_security_immature_but_not_scary_says_gartner/

[39] Announcing Docker Hub and Official Repositories, Docker Blog, Available: https://blog.docker.com/2014/06/announcing-docker-hub-and-official-repositories/