# Containerization Using Docker Technology

Alexandru Eftimie
University POLITEHNICA of Bucharest, Bucharest, Romania
Department of Telecommunication, University "Politehnica" of Bucharest, Romania
E-mail: alexandru.eftimie@gmail.com

Eugen Borcoci
University POLITEHNICA of Bucharest, Bucharest, Romania
Department of Telecommunication, University "Politehnica" of Bucharest, Romania
E-mail: eugen.borcoci@elcom.pub.ro

***Abstract*— This paper aims to provide a clearer view of container technology, such as its advantages and disadvantages, how it can cooperate with Openstack, and document the improvements made by this cooperation. In terms of containerization technology, this paper will illustrate the components of Docker and the impact it has compared to the already classic technology of virtual machines. Another element to be addressed in this paper is the importance of moving some of the computing power to the periphery of networks. This can be done using existing peripheral devices to the extent that the computing resources on this equipment allow. This move is necessary to provide an infrastructure capable of supporting services with low latency needs, minimal delay (traffic sensor, vehicle sensors) and at the same time an infrastructure that will free the core of networks from a large volume of data. This paper will follow the comparison of household equipment that can play an active role in computing at the periphery of networks to highlight what types of applications or calculations can be performed on them. We will follow in this paper the comparison of household equipment that can play an active role in computing at the periphery of networks to highlight what types of applications or calculations can be performed on them.**

***Keywords-container; Docker; Network Function Virtualization (NFV).***

## I. INTRODUCTION

Currently, typical network architectures have three main areas: access, transport, and core. Most of computational resources for applications are in the cloud, far away from the end user. Another element which will be addressed in this paper is the importance of moving some of the computing power to the periphery of networks. This can be done using existing peripheral devices, like routers, dedicated gateways, servers, to the extent that the computing resources on this equipment allow [1]. This move is necessary to provide an infrastructure capable of supporting services with low latency needs, minimal delay and at the same time an infrastructure that will free the core of networks from a large volume of data. We will follow in this paper the comparison of household equipment that can play an active role in computing at the periphery of networks to highlight what types of applications or calculations can be performed on them.

Recently, we have seen a trend called "fog computing" (an architecture that uses edge devices to carry out a substantial amount of computation) and, therefore, the need for processing on the periphery of networks. Hence, there is a need to implement computing and processing machines in this area, a need we have not encountered so far. Given that the services that are liable to be moved to the edge of the network ("edge computing") are diverse and offered by various providers, a possible solution could be the implementation of OpenStack on the periphery of networks by Internet providers / transport providers.

In Section 2 an overview of the containerization technology is going to be described, illustrating the advantages it has compared with virtual machines. Section 3 will cover the Docker technology illustrating its components, the architecture and how the isolation is achieved. In Section 4 we will describe the cooperation between Network Function Virtualization (NFV) and containers and compare current approaches and platforms used to migrate computational resources to the edge of the network.

## II. CONTAINERS AND MICROSERVICES

Containerization [2] has become a major trend in software development as an alternative or companion to virtualization [3]. This involves encapsulating or packaging the software code and all its dependencies so that it can run smoothly and consistently on any infrastructure. The technology has matured rapidly, leading to measurable benefits for developers and operations teams, as well as general software infrastructure.

Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, the code is developed in a specific computing environment which, when transferred to a new location, often leads to bugs and errors.

Containers are often referred to as "lightweight", which means that they share the core of the machine's Operating System (OS) and do not require the association of an operating system within each application. Containers are inherently smaller than a virtual machine and require less

 ISBN: 978-1-61208-837-2 

start-up time, allowing many more containers to run on the same computing power as a single Virtual Machine (VM). This leads to higher server efficiencies and, in turn, reduces server and licensing costs.

Simple containerization allows applications to be "written once and run anywhere." This portability is important in terms of the development process and supplier compatibility. It also offers other notable advantages, such as fault isolation and ease of management and security [4].

Containerization offers significant benefits to developers and development teams. These include the following:

- Portability: A container creates an executable software package that is abstracted (unattached or unattended) from the host operating system and, therefore, portable and able to run smoothly and consistently on any platform or cloud.
- Agility: The open source Docker engine for running containers has started the industry standard for containers with simple tools for developers and a universal presentation approach that works on both Linux and Windows operating systems. The container ecosystem has shifted to engines managed by the Open Container (OCI) initiative.
- Speed: Containers are often referred to as “lightweight,” which means they share the core of the machine’s OS. Not only does this lead to higher server efficiency, but it also reduces server and licensing costs, while speeding up startup times because there is no operating system to boot.
- Defect isolation: Each containerized application is isolated and operates independently of the others. The failure of one container does not affect the continuous operation of other containers. Development teams can identify and correct any technical issues in one container without having to consider other containers.
- Efficiency: Software running in containerized environments shares the machine's operating core and application layers in a container can be shared across containers. Thus, the containers are inherently smaller than a VM and require less start-up time, allowing many more containers to run on the same computing power as a single VM.
- Easy to manage: A container orchestration platform automates the installation, scaling and management of containerized tasks and services. Container orchestration platforms can make management tasks easier, such as scaling containerized applications, running newer versions of applications, and providing monitoring, recording, and debugging, among other functions.

Software companies, large and small, accept microservices as a superior approach to application development and management, compared to the previous monolithic model that combines a software application with the associated user interface and the underlying database in a single unit on a single server platform. With the help of microservices, a complex application is divided into a series of smaller, more specialized services, each with its own database and its own business logic. Microservices then communicate with each other through common interfaces (such as APIs) and REST interfaces (such as HTTP). Using microservices, development teams can focus on updating certain areas of an application without impacting it, leading to faster development, testing, and implementation.

Containers, microservices and cloud computing work together to bring application development and delivery to new levels, which are not possible with traditional methodologies and environments. These next-generation approaches add agility, efficiency, reliability, and security to the software development lifecycle - all leading to faster application delivery and improvements to end users and the marketplace [4].

## III. DOCKER

Docker is an open platform for developing, transporting, and running applications. Docker allows applications to be separated from the infrastructure so that software can be delivered quickly. With Docker, the infrastructure can be managed the same way the applications are managed. By taking advantage of Docker's methodologies for fast code forwarding, testing, and implementation, the delay between writing code and running it in production can be significantly reduced.

Docker offers the ability to wrap and run an application in an isolated environment called a container. Isolation and security allow multiple containers to run simultaneously on a given host. Containers are light because they do not need to be overloaded by a hypervisor, but run directly into the core of the host machine. This means that more containers can be run on a given hardware combination than if virtual machines are used. One can even run Docker containers in host machines that are virtual machines.

Docker provides tools and a platform to manage the container lifecycle:

- The application and its support components can be developed using containers.
- The container becomes the unit for distributing and testing the application.
- Implementing the application in the production environment, as a container or an orchestrated service. This works the same whether the production environment is a local data center, a cloud provider, or a hybrid of the two.

Docker Engine is a client-server application with the following major components, depicted also in Figure 1:

- A server that is a type of long-term program called a daemon process.
- REST API that specifies the interfaces that programs can use to talk to the daemon and instruct it on what to do.
- A Command Line Interface (CLI) client.

 ISBN: 978-1-61208-837-2 

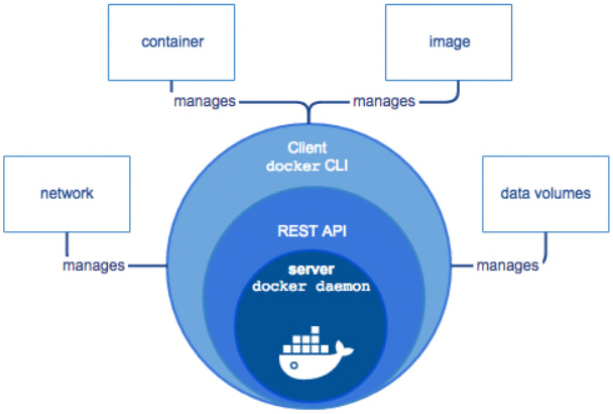


Figure 1. Docker components [5]

The Docker container-based platform enables portable workloads. Docker containers can run on the developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a hybrid environment. Docker's portability and lightweight nature make it easy to dynamically manage workloads, extend or eliminate applications and services, as required, in real time.

## A. Docker arhitecture

Docker uses a client-server architecture. As illustrated in Figure 2, the Docker client speaks to the Docker daemon, which makes it difficult to lift the construction, run, and distribute Docker containers. The Docker client and daemon can run on the same system, or a Docker client can connect to a remote Docker daemon. The Demon client and daemon communicate using a REST API, through UNIX sockets, or a network interface.

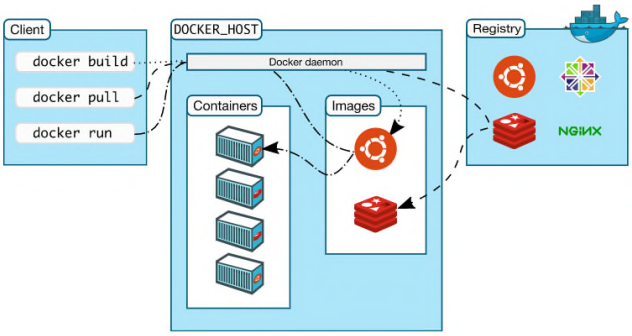


Figure 2. Docker architecture [5]

An image is a read-only template with instructions for creating a Docker container. Often, one image is based on another image, with some additional customizations. For example, one can build an image that is based on the ubuntu image, but installs the Apache web server and application, as well as the configuration details needed to run the application.

One can create new images or use only those created by others and published in a register. To build a new image, a Docker file is created with a simple syntax to define the necessary steps to create the image and run it. Each statement in a Docker file creates a layer in the image. When the Docker file is changed and the image is rebuilt, only those modified layers are rebuilt. This is part of what makes images so light, small and fast compared to other virtualization technologies.

A container is an executable instance. One can create, start, stop, move, or delete a container using the Docker API or CLI and it is possible to connect a container to one or more networks, attach its storage, or even create a new image based on its current state [5].

## B. Isolation in Docker technology

Docker isolates different containers by combining four main concepts:

- Groups.
- Namespaces.
- Stackable image layers and copy writing.
- Virtual network bridges.

Control groups are a way of assigning a subset of resources to a particular process group. This can be common, for example, if we make sure that even if the processor is very busy with Python scripts, the PostgreSQL database still receives dedicated CPU and RAM. Figure 3 illustrates this in an example scenario with 4 processor cores and 16 GB RAM:

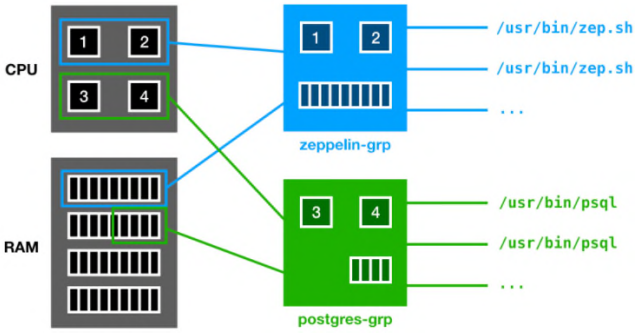


Figure 3. Allocation of resources to control groups [6]

Figure 4 illustrates the parts of a typical process tree in which the init process started a logging service (syslogd), a scheduler (cron), and a connection shell (bash). Within this tree, each process can see all other processes and can send signals (for example to request that the process be stopped) if desired. Using PID namespaces virtualizes the PIDs for a specific process and all of its subprocesses, leading it to believe that it has PID 1. It will also not be able to see any process other than its own children.

```
1
/sbin/init
          +-- 196 /usr/sbin/syslogd -s
          +-- 354 /usr/sbin/cron -s
          +-- 391 login
              +-- 400 bash
                  +-- 701 /usr/local/bin/pstree
```

Figure 4. Process tree [6]

 ISBN: 978-1-61208-837-2

To achieve file system isolation, the namespace will map a node from the file system tree to a virtual root inside that name. By searching for the file system in that namespace, Linux will not allow user to go beyond the virtualized root. Figure 5 shows a part of a file system that contains several roots of the "virtual" file system in the / drives / xx folders, each containing different data.

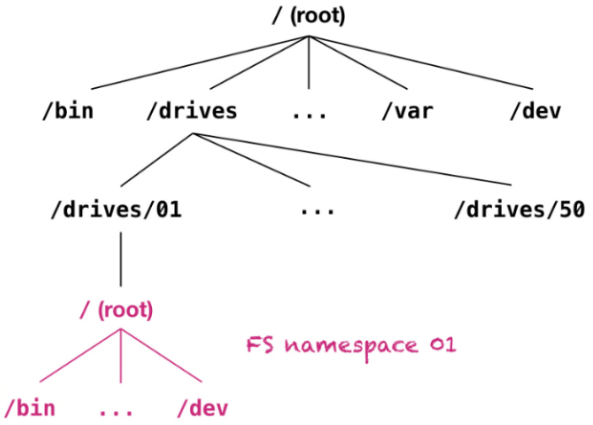


Figure 5. Part of a file system that contains multiple "virtual" file system roots [6]

Docker has a persistence of images in stackable layers. A layer contains changes to the previous level. For example, if one installs Python first and then copy a Python script, the image will have two additional layers: one that contains Python executables and one that contains the script. In Figure 6 a Zeppelin, a Spring and a PHP image are showed.

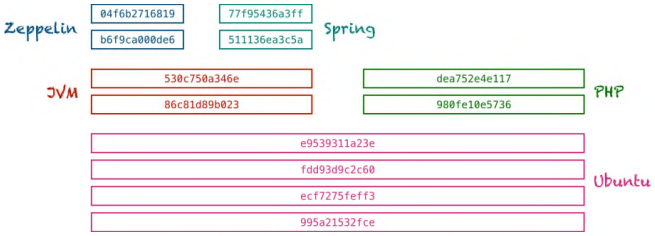


Figure 6. Ubuntu-based Zeppelin, Spring and PHP images [6]

In order not to store Ubuntu three times, the layers are immutable and shared. Docker uses copy-on-write to make a copy of a file only if there are changes. When an image-based container is started, the Docker daemon will provide all the layers contained in that image and place it in an isolated file system namespace for that container. The combination of stackable layers, copy-on-write namespaces, and file system allows a container to run completely independent of things "installed" on the Docker host without wasting much space. This is one of the reasons why containers are lighter compared to virtual machines.

## IV. CONTAINERS IN NFV

To cope with the growing use of networks, driven by new mobile customers, and to meet the demand for new network services and performance guarantees, telecom service providers leverage virtualization on their network by implementing network services in virtual machines. They are disconnected from traditional hardware. This effort, known as NFV, reduces operating expenses and offers new business opportunities. At the same time, new mobile networks, new enterprise and IoT networks introduce the concept of "computing capabilities" that are pushed to the edge of the network, in the immediate vicinity of users. However, the strong footprint of current NFV platforms prevents them from operating at the edge of the network [6].

Data consumption is growing exponentially in today's communications networks. This irreversible trend is determined by the increase in the number of end users and the widespread penetration of new mobile devices (smartphones, portable devices, sensors, etc.). In addition, the consumption of mobile data is also accelerated by the increased capabilities of mobile customers (e.g., higher-resolution screens and HD cameras) and the user's desire for high-speed, always-on, multimedia-oriented connectivity. At the same time, the Telecommunications Service Provider (TSP) market is becoming competitive as the number of “on top” service providers increase, reducing user subscription fees.

As a result, telecom service providers have begun to lose existing revenue, while suffering increased capital expenditures and operating costs that cannot be offset by rising subscription costs. To meet the challenges mentioned above, service providers have begun to migrate network infrastructure to software. By virtualizing traditional services providers can save operational and capital costs and meet user requirements for personalized services. This transformation, called network virtualization, is transforming the way operators build their network architecture to disconnect network functionality from physical locations for faster and more flexible network provisioning.

In Table 1, some popular marginal devices are presented along with their release date, architecture, CPU, and memory parameters. The list includes residential equipment for large-scale customers. As can be seen from Table 1, recent CPE devices and home routers are equipped with powerful computing capabilities (e.g., processors up to 1.6 GHz) and a considerable amount of RAM (up to 1 GB) for run a Linux-based operating system (OpenWRT or DD-WRT).

TABLE I. EDGE DEVICES SPECIFICATION [2]

| Customer Device | Architecture | CPU | Memory |
|---|---|---|---|
| **Residential CPE home routers** | | | |
| Virgin SuperHub 3(Arris TG2492s) | Intel Atom | 2x1.4 GHz | 2x256 MB |
| Google Fiber Network Box GFRG110 | ARM v5 | 1.6 GHz | N/A |
| Orange Livebox 4 | Cortex A9 | 1 GHz | 1 GB |
| **Commodity wireless routers** | | | |
| TP-LINK Archer C9 home router | ARM v7 | 2x1 GHz | 128 MB |
| Ubiquity Edge Router Lite 3 | Cavium MIPS | 2500 MHz | 512 MB |
| Netgear R7500 Smart Wifi Router | Qualcomm Atheros | 2x1.4 GHz | 384 MB |
| **IoT edge gateways** | | | |
| Dell Edge Gateway 5000 | Intel Atom | 1.33 GHz | 2 GB |

| | | | |
|---|---|---|---|
| NEXCOM CPS 200 Industrial IoT Edge Gateway | Intel Celeron | 4x2.0 GHz | 4 GB |
| HPE Edgeline EL4000 | Intel Xeon | 4x3.0 GHz | Up to 64 GB |

As demonstrated in "Container Network Functions" [2], even a home TP-Link router with a 560 MHz processor and 128 MB of RAM can be used to run multiple VNFs using Linux containers. In addition to low-cost marginal devices, such as home routers and residential Customer Premises Equipments (CPE), some vendors have also introduced IoT gateways with state-of-the-art processors and up to 64 GB of RAM to host new services, such as intelligent analysis at the edge of the network.

While positioning at the edge of the network has many advantages, traditional NFV platforms have been built on high-power servers, mainly operating virtual machines (using technologies, such as Xen Project [7] or Kernel Virtual Machine [8]) for VNF. Table 2 summarizes the features supported by some existing solutions. The information presented reflects the public information available at the time of writing. Cloud4NFV [1] is a platform that promises to provide a new service to end customers, based on cloud, defined software networks and WAN technologies. The research projects UNIFY [3] and T-NOVA [7] share a similar vision of the unification of cloud networks and providers by implementing a system of "network functions as a service". The OPNFV Linux project is the most popular open source NFV platform, with support and implementations from many vendors and large vendors. While all these platforms have made important contributions in the field, none of them has so far featured a container-based, network-focused and mobility-focused NFV system.

TABLE 1. SUMMARY OF EXISTING APPROACHES [2]

| | GNF | Cloud NFV [1] | UNIFY | T-NOVA [7] | OPNFV |
|---|---|---|---|---|---|
| Virtualization technology | Container | VM | VM | VM | VM |
| End-to-end service mgmt. | Yes | Yes | Yes | Yes | Yes |
| Distributed infrastructure | Yes | Yes | Yes | Yes | Yes |
| Traffic steering | Yes | Yes | Yes | No | Yes |
| Runs on the network edge | Yes | No | No | No | No |
| SFC support | Yes | Yes | Yes | No | Yes |
| Roaming VNFs | Yes | No | No | No | No |

In [2], there were highlighted some basic features of containerized VNFs that were measured on an Intel i7 server with 16 GB of memory: Delay: maintaining the low delay introduced by VNF is important to implement transparent services and is therefore a key benchmark for VNF technologies. In Figure 2a, the delay introduced by different virtualization platforms by displaying the idle time (round time trip) is expressed. While ClickOS [8] achieves a slightly lower delay than containers, ClickOS is built on top of a modified, specialized hypervisor that optimizes packet forwarding performance. On the other hand, container-based functions use unmodified containers on a standard Linux kernel, allowing deployment on devices that do not support hardware virtualization (for example, all CPE devices and home routers). Other VM-based technologies, such as KVM or Xen VM, result in a much longer delay, which is mainly attributed to copying packages from the hypervisor to the VM.

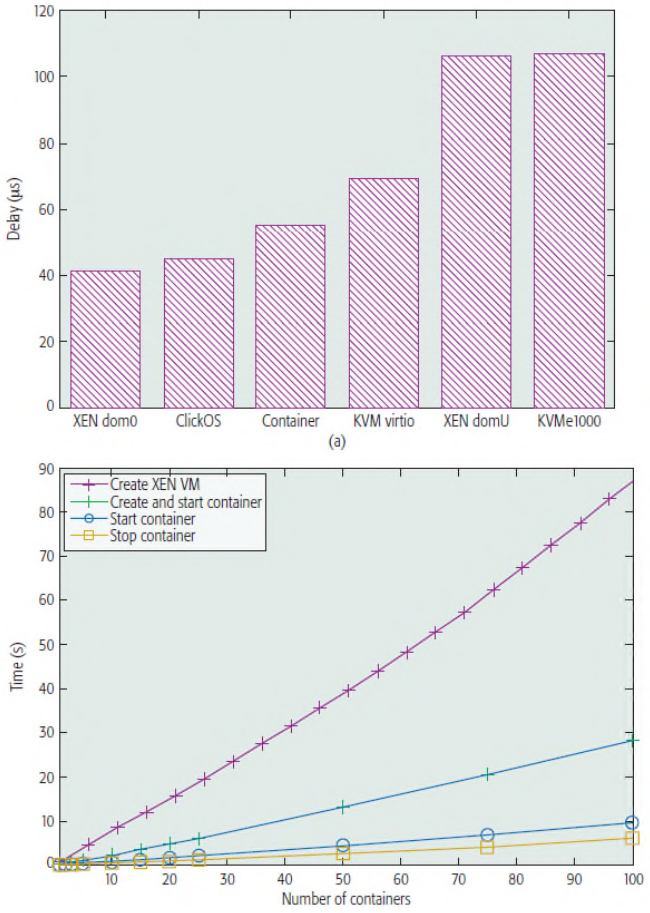


Figure 7. Performances of virtual functions as containers: a) ping delay; b) creation, start and stop times; c) idle memory consumption [2]

## V. CONCLUSION

Containers give a false sense of security. There are many pitfalls when it comes to securing applications. It is wrong to assume that one way to secure them is to place them in containers. Containers do not provide anything in themselves. If one keeps his web application containerized, it could be locked into namespaces, but there are several ways to get rid of it depending on his configuration.

Considering that there are Docker images that require the exposure of more than 20 ports for different applications inside a container, Docker's philosophy is that a container

 ISBN: 978-1-61208-837-2 

should do a single job and user should compose them instead of making them heavier. If one ends up packing all the tools in one container, all the benefits will be lost, user may have different versions of Java or Python inside, and he might end up with a 20GB image that can not be managed.

The paper managed to provide an overview of the current options for implementing "fog computing": whether it is devices already in production at the end user, or we are talking about native equipment, dedicated to perform functions specific to information processing on the periphery network. Observing these things, a variant that can be adopted is the use of dedicated servers at the periphery of the networks on which to build special functions using Docker technology. This has the advantage of a fast implementation of functions, rapid scaling, as well as the advantage of having a platform shared by many entities given the isolation discussed in Section 4. The problem that remains open is where to implement this dedicated server: it is necessary aggregate and respond to a large number of requests to justify the investment and available computing resources. Physical distance from end users (mobile devices, sensors, etc.) must also be considered in order not to lose the main advantage offered by the concept of "fog computing": latency and low delay.

In this paper, we provided an overview of Docker technology and how this technology can contribute to a better exploitation of virtual network functions. This is because Docker provides very good isolation between instances and at the same time does not require the presence of dedicated software (hypervisor), offering greater flexibility than classic virtual machines.

There are a significant number of benefits to using VNFs on containers rather than on the hypervisor. However, if we look at the technological innovation there is no outstanding progress, and this is due to the lack of a model of common guidelines. Now, 5G networks are starting to be implemented and tested in some cities by service providers with the help of top providers. The development will lead to targeting more innovative features that 5G brings, such as network tracing, Mobile Edge Computing (MEC) and cRAN (Cloud Radio Access Networks). These new 5G features will certainly require the dynamism and benefits offered by containers for the highly automated deployment of services to each edge of the 5G network. We can expect all service providers and network solution providers to be aware of the benefits of the container to be used to achieve high efficiency.

Container technologies such as Docker are becoming the leading standards for building containerized applications. They help organizations free from a complexity that limits the agility of development. Containers, container infrastructure and container implementation technologies have proven to be very powerful abstractions that can be applied to several different use cases. Using something like Kubernetes, an organization can deliver a cloud that uses containers exclusively for application delivery.

The growing interest of users and the widespread adoption of Docker and container technology have forced old retailers to deliver at least their first container products, but it should be noted in the long run how these technologies can integrate seamlessly and meet the technical requirements of old systems.

## REFERENCES


[1] S. João, D. Miguel and C. Jorge, "Cloud4NFV: A Platform for Virtual Network Functions," in *in 3rd Intl. Conf. on Cloud Networking (CloudNet). IEEE, 2014*, 2014.

[2] R. Cziva and D. P. Pezaros, "Container Network Functions: Bringing NFV to the Network Edge," in *Communications Magazine. IEEE, June 2017*, 2017.

[3] C. András et al., "Unifying Cloud and Carrier Network," in *Utility and Cloud Computing (UCC), 2013*.

[4] IBM Cloud Education, [Online]. Available: https://www.ibm.com/cloud/learn/containerization. [Accessed 7 March 2021].

[5] Docker Docs, "Docker," [Online]. Available: https://docs.docker.com/get-started/overview/. [Accessed 7 Feb 2021].

[6] F. Rosner, "CodeCentric Blog," [Online]. Available: https://blog.codecentric.de/en/2019/06/docker-demystified/. [Accessed 7 Feb 2021].

[7] X. George et al., "T-NOVA: Network Functions as-a-Service," *IEEE Explore,* 2015.

[8] M. Joao and A. Mohamed, "ClickOS and the Art of Network," in *11th USENIX Symposium on Networked Systems*, Seattle, WA, USA, 2014.


 ISBN: 978-1-61208-837-2 