

The Headend Evolution: Design Considerations for Deploying vCCAP and Other VNFs

A Technical Paper prepared for SCTE•ISBE by

Patricio Sebastian Latini
Regional VP - CALA
CASA Systems
100 Old River Rd. – Andover, MA
+1 (305) 504-9250
patricio.latini@casa-systems.com

Table of Contents

Title	Page Number
1. Introduction.....	4
2. Virtualization	4
2.1. Virtualization Introduction.....	4
2.2. The Hypervisor	5
2.3. Virtual Machines	6
2.4. Network Function Virtualization.....	7
2.5. Management and Orchestration.....	7
3. Containers	8
3.1 – Containers and Microservices.....	10
3.2 – Containers Orchestration.....	11
3.3 – Continuous Integration/Continuous Deployment	12
4. The Clouds	13
4.1. Private Cloud.....	13
4.2. Public Cloud	14
4.3. Hybrid Cloud	15
4.4. Platform as a Services (PaaS).....	15
5. Virtualizing the CCAP.....	17
5.1. Control and User Plane Separation (CUPS).....	17
5.2. Virtual CCAP Core and Remote PHY.....	18
5.3. Multi-access Edge Computing (MEC)	20
5.4. Virtual MAC Manager and Remote MAC/PHY.....	21
6. Future and Convergent Core Functions.....	22
7. Conclusion	22
Abbreviations.....	23
Bibliography & References	24

List of Figures

Title	Page Number
Figure 1 – Virtual Machines	5
Figure 2 - Hypervisor Types.....	6
Figure 3 - Network Function Virtualization examples	7
Figure 4 - NFV Paradigm.....	8
Figure 5 - NFV MANO Framework	8
Figure 6 - Containers Architecture	9
Figure 7 - Monolithic vs Microservices Architectures	11
Figure 8 - Kubernetes Architecture	12
Figure 9 - CI/CD Process.....	13
Figure 10 - Cloud Services Models	15
Figure 11 - Cloud Native Hybrid Architecture	16
Figure 12 - Integrated CCAP vs virtual CCAP Functions	18
Figure 13 - vCCAP Node to Container Mapping.....	19
Figure 14 - vCCAP Kubernetes Managed Architecture.....	19
Figure 15 - vCCAP Container Based Redundancy	20
Figure 16 - vCCAP Deployments Models	20

Figure 17 - Integrated Remote MAC-PHY Device	21
Figure 18 - Decoupled Remote MAC - Remote PHY Devices.....	21
Figure 19 - Fixed Mobile Convergence Evolution	22

List of Tables

Title	Page Number
Table 1 - Hypervisor Examples	6
Table 2 - Container Engines	9
Table 3 - Private Cloud Platforms	13
Table 4 - Cloud Service Models	14
Table 5 - PaaS Platforms.....	16

1. Introduction

The telecommunications industry is well underway to moving to virtualizing network functions, and the cable industry is no exception. This paper focuses on providing understanding of virtualized solutions and technology, by analyzing design aspects, capacity planning and architecture evolutions of a CCAP virtual function.

Initially, different virtualization technologies such as virtual machines vs containers are compared, together with continuous integration and deployment as a key element for the required agility to deploy new services and network functions. It is important to mention how the evolution to a distributed computing model and particularly how Mobile Edge Computing (MEC) and network slicing will shape future networks.

Next, a network architecture design is presented focusing on the evolution to separate control and user planes and the impact on network traffic, how they align with distributed access architectures in two flavors, Remote PHY and Remote MAC PHY, and how the vCCAP function could split in two new logical functions in the near future.

Lastly, a set of conclusions is presented to help cable operators better understand the requirements, design options and tradeoffs of vCCAP and DAA deployments in general for the next two to three years and how this decision will impact on the deployment of other related VNFs such as BNG (Broadband Network Gateway) or an EPC (Evolved Packet Core).

2. Virtualization

2.1. Virtualization Introduction

Generally speaking, Virtualization is defined as running one or multiple instances of a computer system on a layer which is abstracted from the hardware. Each abstracted instance is called a virtual instance. Over the year's virtualization evolved from being just a way of running more than one operating system on a desktop computer at the same time, at the expense of noticeable performance degradation to now being a ubiquitous technology in the server world.

Virtualization offered the ability to run different operating systems, and multiple instances of each. This concept allowed a large system to be split into multiple smaller ones, and hence where now a server could be used to run multiple applications or services while allowing each of them to run completely isolated of the other like it was running in its own dedicated server (Morabito, Cozzolino, Ding, Beijar, & Ott, 2018) - Figure 1.

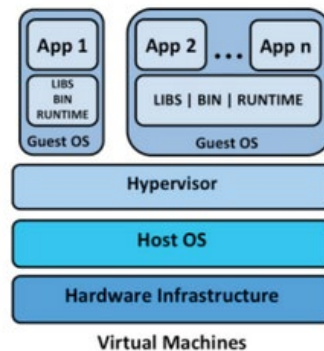


Figure 1 – Virtual Machines

2.2. The Hypervisor

As seen in Figure 1, in order to run multiple virtual machines in physical hardware, a software component is required. This software component is called a hypervisor, a program which takes care of allocating the available resources to each of the virtual machines. Any program run under the hypervisor should exhibit an effect identical with that demonstrated if the program had been run on the original machine directly (Popek & Goldberg, 1974). There are two types of hypervisors as seen in Figure 2. Type one hypervisors are operating systems themselves and run the guest virtual machines directly. Type two hypervisors need to run on a pre-installed operating system and run as application that can be stopped or started.

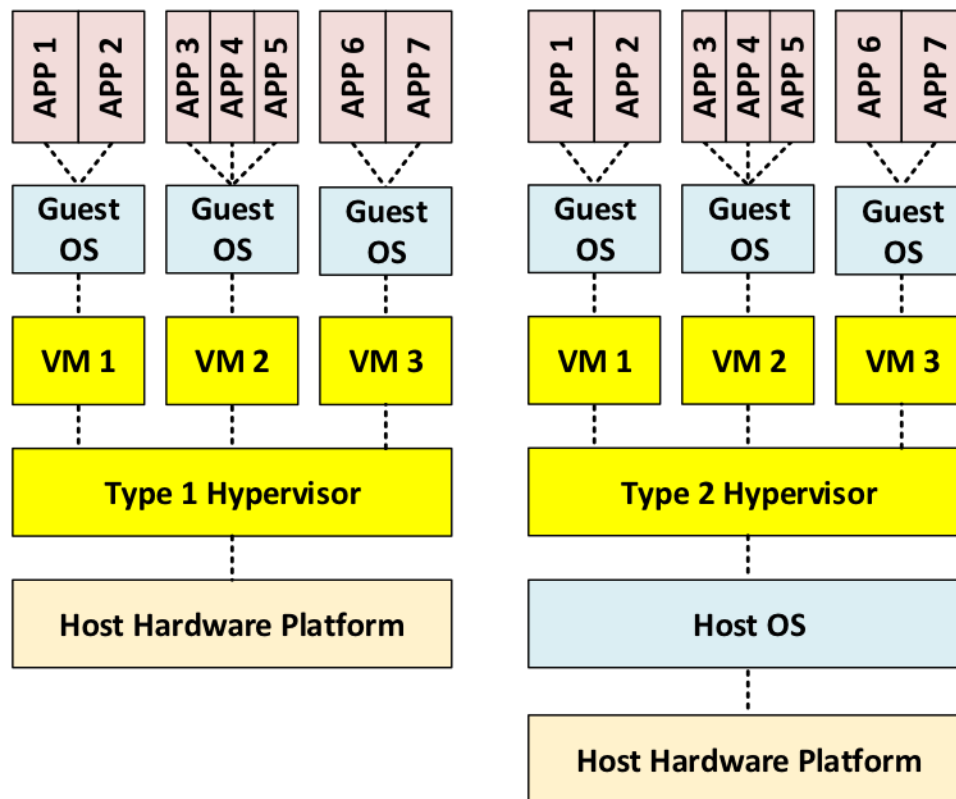


Figure 2 - Hypervisor Types

Some examples of commercial and open source hypervisors are listed in Table 1.

Table 1 - Hypervisor Examples

Type 1	Type 2
VMware ESX and ESXi	VMware Workstation
Microsoft Hyper-V	Oracle VM VirtualBox
Citrix XenServer (Xen Based)	Red Hat Enterprise Virtualization (kvm Based)
Oracle VM (Xen Based)	

2.3. Virtual Machines

A virtual machine is a term that dates back to 1974 and is defined as an efficient and isolated duplicate of a real machine, where a piece of software provides an environment which is essentially identical to the original machine and programs can run with only minor decreases in speed (Popek & Goldberg, 1974). Virtual machines or guests can have access to the resources available on the host system. The host system can provide computing power, memory, disk space and access to the network interface cards. In the early days of x86 virtualization, hypervisors needed to fully emulate the behavior of the virtual machine including the CPU instruction set, causing a big overhead of computing power and significantly affecting performance of the whole

virtualized system. In 2006, Intel and AMD introduced hardware implemented virtualization extensions (VT and SVM) which allowed the hypervisors to directly access the CPU instructions thus making x86 virtualization possible in terms of performance (Adams & Agesen, 2006). Depending on the expected quality of service, sometimes those resources could or could not be oversubscribed.

2.4. Network Function Virtualization

In 2013 ETSI coined the term Network Function Virtualization (NFV). At that time telecommunications operators' networks were populated with a large and increasing variety of proprietary hardware appliances. Launching a new service required even more hardware and the space and power requirements to integrate those boxes was becoming a real challenge. At the same time energy costs were increasing and also hardware obsolescence cycles were becoming shorter, increasing CAPEX costs (European Telecommunications Standards Institute, 2012). At the same time off-the-shelf server computing power was increasing with the costs of those servers being reduced.

Network Functions Virtualization's goal is to address these problems by providing standardization to the virtualization technology in order to consolidate different network equipment types into industry standard off the shelves servers, switches and storage (European Telecommunications Standards Institute, 2012) as seen in Figure 3.

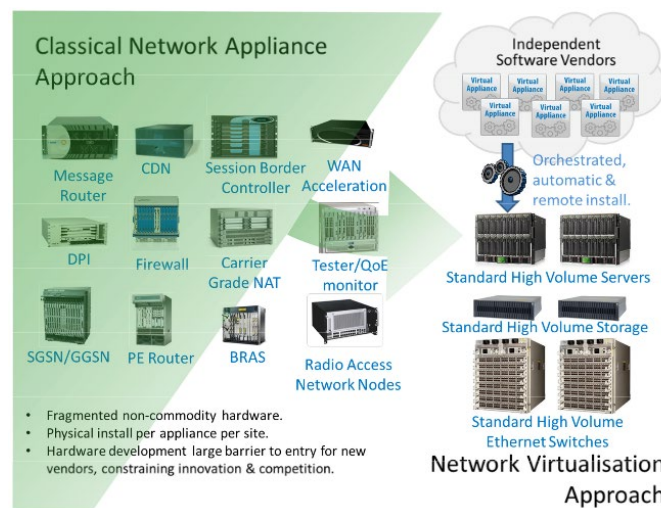


Figure 3 - Network Function Virtualization examples

2.5. Management and Orchestration

Network function virtualization (NFV) changes how networks are managed. Hundreds of network functions running on a server farm can easily make operations very complex. For that reason the European Telecommunications Standards Institute (ETSI) started in 2013, an effort to define the framework shown in Figure 4 for NFV management from the initial set-up, to day-to-day operations, which is called NFV MANO (Management and Network Orchestration). This is the framework for the management and orchestration of all resources in a data center for virtualized functions, those resources include: compute, networking, storage, and virtual

machines (VM) as seen in Figure 5. The main goal of MANO is to allow flexible on-boarding. (European Telecommunications Standards Institute, 2012)

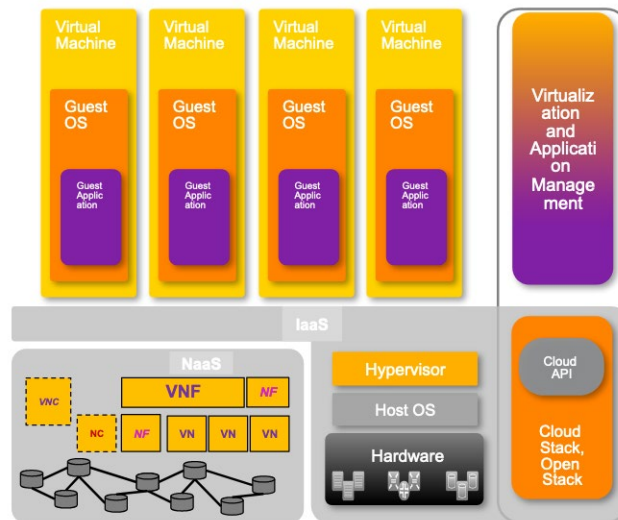


Figure 4 - NFV Paradigm

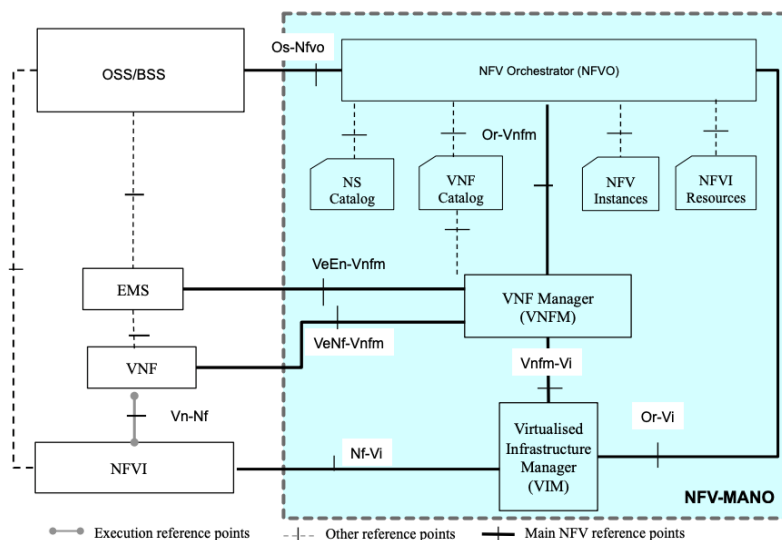


Figure 5 - NFV MANO Framework

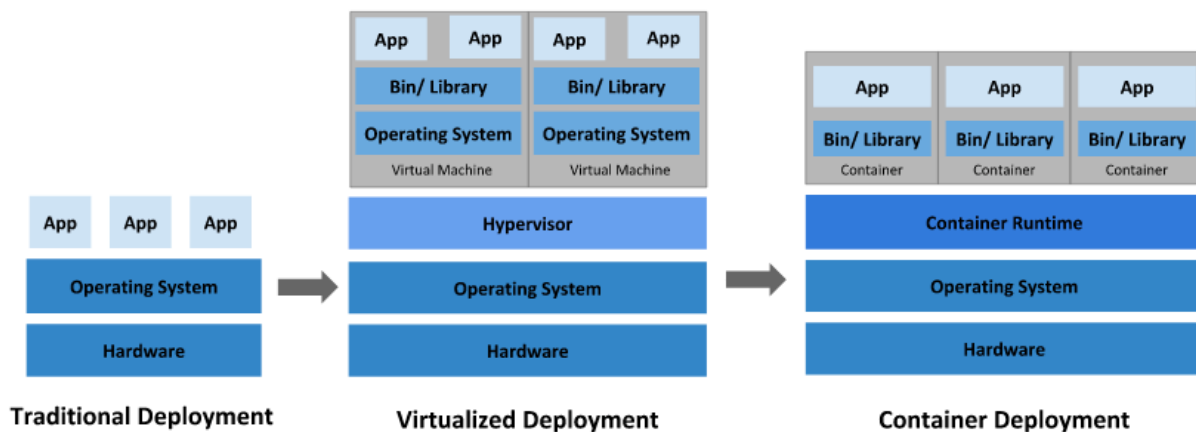
3. Containers

As VMs were deployed, organizations started to run applications on separate virtual machines in order to dedicate their own resources (CPU, memory, storage). However, each application had to have a separate operating system running in each virtual machine.

At scale virtual machines may take up a lot of system resources, and a base operating system installation may take several gigabytes of storage. For example, running a single webserver on a virtual machine may require several gigabytes of operating system and libraries, while the application itself only takes a few megabytes of storage and memory. Multiply that by a big number of webserver and other applications, and the overhead of virtualization becomes very noticeable.

At the same time, with the wide adoption of Linux as an operating system, and a big share of network applications using it as its underlying platform, an interesting idea was born: Why can't applications share the same operating system and be virtualized at the operating system level instead of at the hardware level?

A container is an isolated environment from the operating system host that runs an application by virtualizing it. This allows it to create multiple application workloads on a single OS instance. The kernel of the host operating system provides the required components for running the different functions of an application, separated into containers as shown in Figure 6. Containers run isolated tasks from each other, and an application cannot harm the host machine nor come in conflict with other apps running in other containers. (Simic, 2019)



Retrieved from: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

Figure 6 - Containers Architecture

Docker is the most common container engine now in the market, however there are several other options for container engines shown in Table 2.

Table 2 - Container Engines

Docker
Mesos
LXC
OpenVZ
Java Container
Windows Server Containers

3.1 – Containers and Microservices

Applications are easy to develop using a monolithic approach, but as the size of the application and its user base grows it becomes very complex to scale up. Monolithic applications can have up to hundreds of different services tightly coupled, which makes it very complex for different development teams to handle their coordination. This is the reason why most of the software industry is moving to a new paradigm, known as microservice architecture (Yu, Silveira, & Sundaram, 2016).

In a microservice approach, an application consists of several services, each independent of the other. Each service does a specific function which is developed and deployed independently from the others. Services transfer data or information to other services using a standardized communication protocol as seen in Figure 7.

Some of the benefits of microservices are easier automated testing, flexible deployment models and increased overall resiliency; however all this comes at the expense of careful planning of the architecture and increased R&D investment, mainly given to the following factors:

- 1) Since everything is an independent service, careful handling of requests traveling between the modules is required. This generates the need of designing in advance the APIs that the microservices will use to communicate between them its maintenance across version changes.
- 2) Testing a microservices based application can be complex. In a monolithic approach, the application just needs to be launched and validate its connectivity with the underlying services such as database or webserver. With microservices, each service needs to be confirmed to be working properly in advance before integration testing of all the microservices as an application can occur.

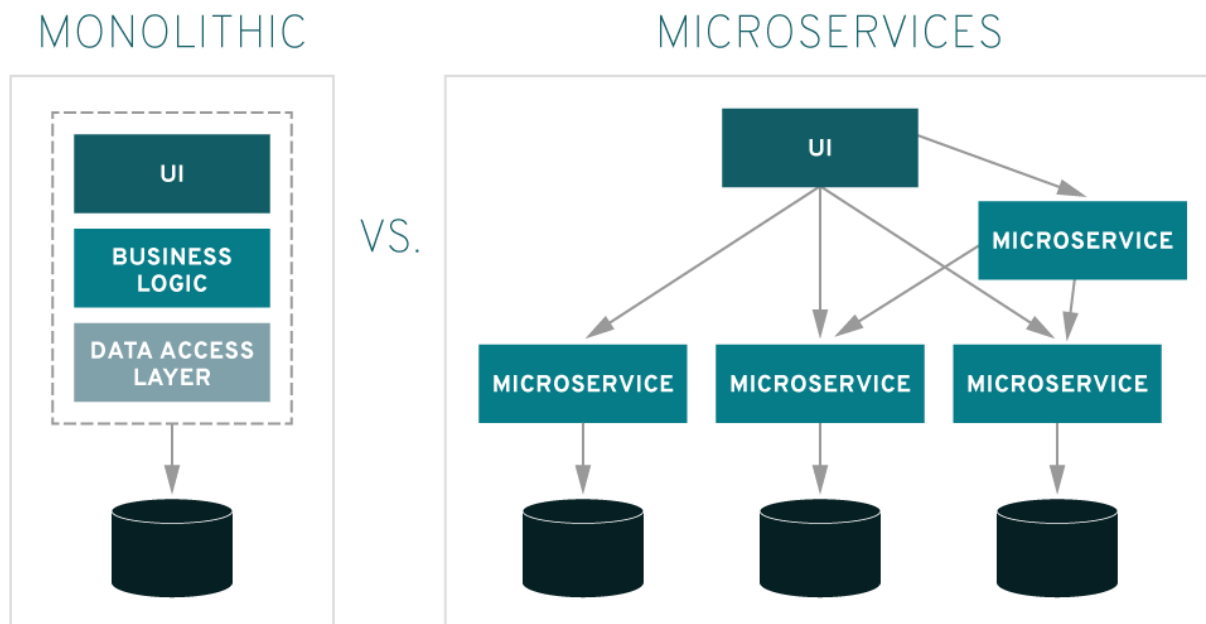


Figure 7 - Monolithic vs Microservices Architectures

As has been shown before, a container is just a way of deploying and running a program or process isolated from others sharing a common operating system. One could have one big monolithic application running as a container and in the other side there could be a big number of microservices running on a bare operating system. However, in the real world these independent approaches have been very complementary, as containers are the portable code envelope and big applications could be decomposed into many microservices that can be independently deployed.

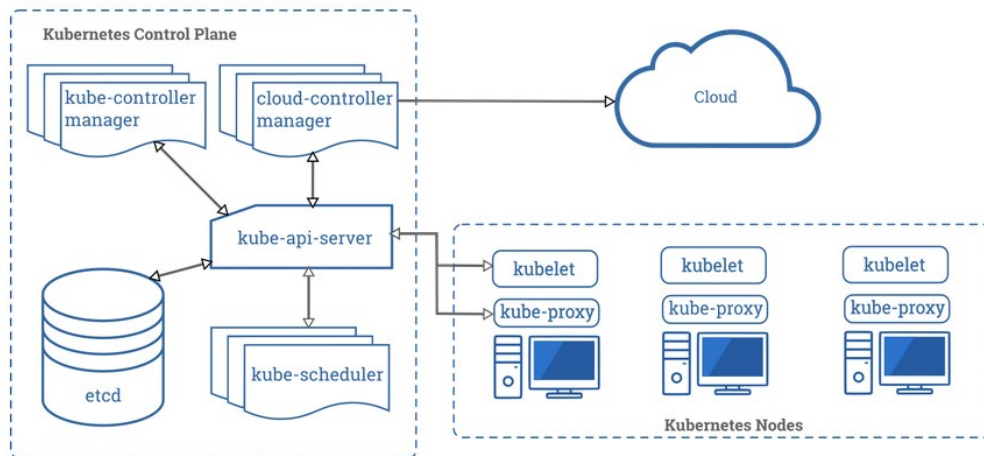
DevOps processes are the pillar of modern applications, and support running multiple parts of applications independently in different microservices, with much greater control over their life cycles.

3.2 – Containers Orchestration

Container orchestration is the process of automating the deployment, management and scaling of containers. As mentioned before, service providers will need to deploy hundreds or thousands of containers and for that reason an automation process for container orchestration is required. Container orchestration can help to deploy the same application across multiple environments and microservices in those containers and make it easier to orchestrate different types of services such as storage, networking, and security. (Redhat, n.d.)

There are several orchestration tools for managing containers at scale together with its lifecycle management. The most popular is called Kubernetes, however Docker Swarm, and Apache Mesos are also well-known orchestration tools.

Kubernetes is an open source orchestration tool that was developed by Google. Kubernetes orchestration allows the deployment of applications that are supported by multiple containers, running those containers in clusters of servers and verifying their health and scale-in or scale-out as required by capacity demands. In general, Kubernetes eliminates most of the manual processes associated with deploying complex containerized applications. The Kubernetes architecture is presented in Figure 8.



Retrieved from: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

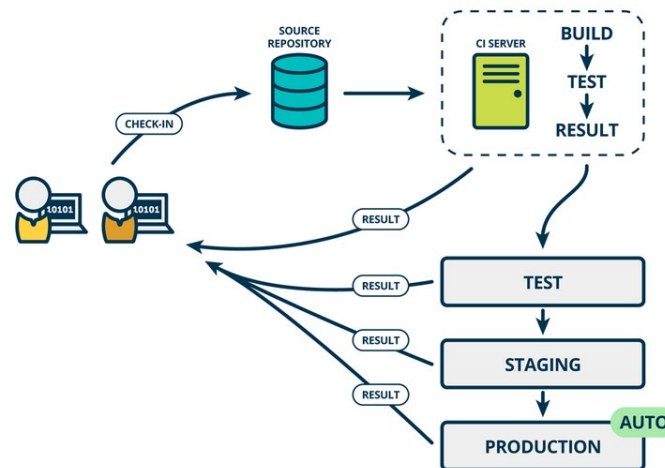
Figure 8 - Kubernetes Architecture

3.3 – Continuous Integration/Continuous Deployment

Continuous software engineering is an area which refers to the development, deployment and feedback from software and customers in a very rapid cycle (Fitzgerald & Stol, 2017). The continuous software engineering process can be split into two main areas

Continuous Integration (CI) is a widely established development practice in the software industry where teams integrate and merge development code very frequently, for example multiple times per day. CI allows software developers to have shorter release cycles and improve software quality (Fitzgerald & Stol, 2017). Many of the processes involved in CI are automated.

Continuous Deployment (CD) is a practice that goes a step further and automatically and continuously deploys the application to a production environment as seen in Figure 9. That environment can be within the same company or with an external customer. In the latter case, Continuous Delivery (CDE) may apply, where the application is only delivered but not automatically turned into production and requires some manual intervention before being deployed.



Retrieved from: <https://www.mindtheproduct.com/what-the-hell-are-ci-cd-and-devops-a-cheatsheet-for-the-rest-of-us/>

Figure 9 - CI/CD Process

Two common software tools for this process are GIT as the source code repository, and Jenkins as the CI Server. Jenkins builds the application from the code repository, automatically tests, and delivers the containers to a container repository.

4. The Clouds

4.1. Private Cloud

According to the National Institute of Standards and Technology (NIST), a private cloud is defined as infrastructure which is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises (National Institute of Standards and Technology , 2011).

The main advantage of a private cloud is that resources are not shared. A private cloud is best for businesses with dynamic needs that require direct control over their computing resources, in general to meet security or regulatory requirements.

Private clouds also have a few disadvantages as increased automation and user self-service can bring added complexity. These technologies require teams to rearchitect data centers and use extra management tools and can result in increased staff and capital expenditures to acquire the infrastructure to support it.

When a private cloud is properly architected and implemented, the organization can benefit from self-service and scalability, and the ability to launch or optimize computing resources on demand. In Table 3 there is a list of the main private cloud platforms in the market. It is important to mention that Openstack-based platforms are provided by several mainstream vendors.

Table 3 - Private Cloud Platforms

Openstack

VMware vCloud

4.2. Public Cloud

According to the National Institute of Standards and Technology (NIST), a public cloud is the cloud infrastructure that is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. (National Institute of Standards and Technology , 2011)

From a service provider standpoint, a public cloud is a platform in which a third-party service provider makes available computing resources which can be software applications, virtual machines (VMs) and complete enterprise-grade infrastructures over the public Internet. This public cloud service provider owns the data centers where customers' services run. Service providers take care of all the infrastructure maintenance and provide connectivity access to applications and data. (IBM, 2020)

There are four main models of cloud services as shown in Table 4 together with examples of cloud providers for each. Each cloud service model has a different management complexity compared to the other, as shown in Figure 10.

Table 4 - Cloud Service Models

Software as a Service (SaaS)	Google Apps, Dropbox, Salesforce, Cisco, Concur, GoToMeeting, Slack
Platform as a Service (PaaS)	Google App Engine, Force.com, Redhat Openshift
Infrastructure as a Service (IaaS)	Amazon Web Services (AWS), Microsoft Azure, Google Compute Engine, Rackspace
Metal as a Service (MaaS)	Amazon Web Services Bare Metal, Microsoft Azure Bare Metal Servers, Google Bare Metal

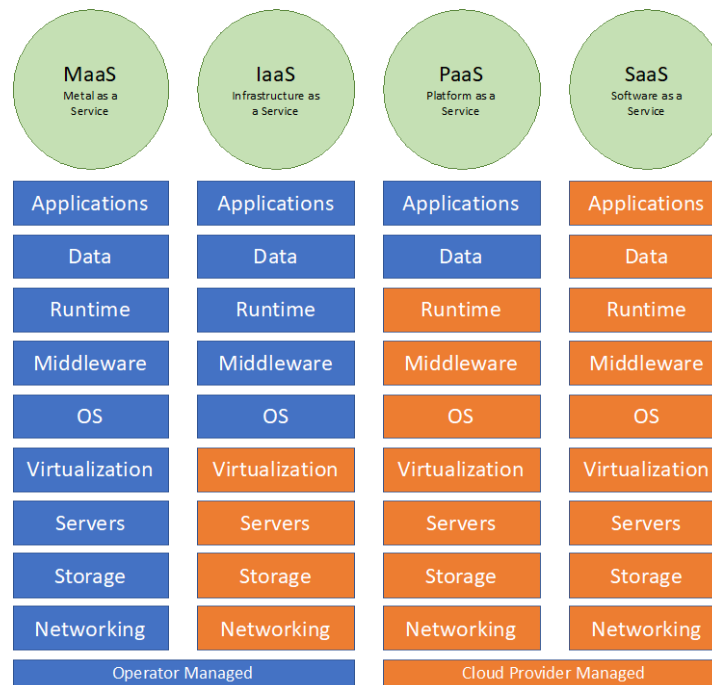


Figure 10 - Cloud Services Models

4.3. Hybrid Cloud

According to the National Institute of Standards and Technology (NIST), a hybrid cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds) (National Institute of Standards and Technology , 2011).

4.4. Platform as a Services (PaaS)

There are several common core microservices which could be shared in either a private or public cloud, without having to replicate them in each application. It may be desirable that service providers have them as a shared PaaS services. Some of them are listed below.

- **Message Queue.** Responsible for delivering messages from one service instance to another. The message queue will support different messaging patterns including request/response, broadcast, pub/sub model.
- **Session Database.** Responsible for storing session state in a common place, which enables stateless processing.
- **Service Discovery.** Responsible for service registration, service health monitoring and service state notification. Note some platforms like Kubernetes provide built-in service discovery.
- **Configuration store.** Responsible for maintaining persistent configuration for the containers. The service configuration will be modeled as a tree structure. Each micro service will have its own subtree

- **Logging.** Responsible to collect logs from all micro-service instances. This service provides a single access point to view the logs.
- **Analytic and Visualization.** Responsible for providing in-depth knowledge of the platform status and presents the result in an easy-to-understand graphical form.
- **Management API.** Responsible for providing RESTAPI/CLI to configure the platform and provide service statistics or session information.

An example implementation of such architecture is shown in Figure 11 with a list of some of the most common software packages for them listed in Table 5.

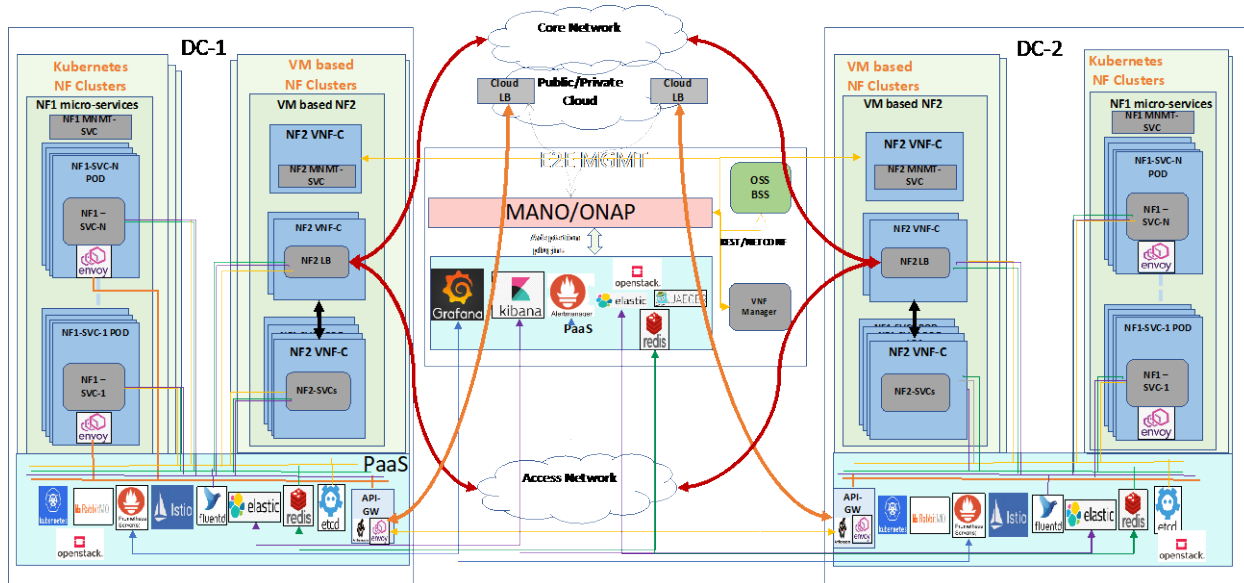













Figure 11 - Cloud Native Hybrid Architecture

Table 5 - PaaS Platforms

	Openstack	VM Orchestration
	Kubernetes	micro-service container orchestration
	Docker Registry	NF micro-service Container image
	Helm	Application packaging, deployment and upgrades
	RabbitMQ	Intra-NF micro-services message bus

	ETCD	NF micro-service specific configuration store
	Istio/Envoy	Service Mesh
	Fluentd/ELK Stack/Jaeger	Logging/tracing
	Prometheus/Grafana	Monitoring and Alerting
	Redis-DB	NF micro-service specific state and stats store
	Calico/Multus	Container Networking

5. Virtualizing the CCAP

5.1. Control and User Plane Separation (CUPS)

Cable Operators have historically used CMTS as Edge equipment which has several functions like per-subscriber session, policy enforcement and data forwarding in the same box. With the advent of Control Plane User Plane Separation (CUPS), CMTS functions could be decomposed and disaggregated such that the User Plane Function (UPF) could be deployed in a distributed manner, and the Control Plane Function (CPF) could be deployed in a centralized manner depending on the level of scale and aggregation needed. (Asati & Bernstein, 2019)

The deployment of new services, such as 4K video, IoT, etc., and increasing numbers of home broadband service users present some new challenges for broadband routers such as:

- Low resource utilization: The traditional CCAP acts as both a gateway for user access authentication and an IP network's Layer 3 edge. The nature of the tightly coupled control plane and forwarding plane makes it difficult to achieve the optimum performance of either of the planes.
- Complex management and maintenance: Due to the large numbers of traditional CCAP instances a network must have, each device must be configured one at a time when deploying global service policies. As the network expands and new services are introduced, this deployment mode will cease to be feasible as it is unable to manage services effectively and rectify faults rapidly. (Hu, et al., 2018)

To address these challenges, a cloud based BNG with CU separation conception is defined in (Broadband Forum, 2018) however the same idea is applicable to a CCAP box. The main idea of Control-Plane and User-Plane separation is to extract and centralize the user management functions of multiple CCAP devices, forming a unified and centralized control plane (CP). The traditional router's Control Plane and Forwarding Plane are both preserved on CCAP devices in the form of a user plane (UP). Note that the CU separation concept has also be introduced in the 3GPP 5G architecture. (3GPP, 2018)

5.2. Virtual CCAP Core and Remote PHY

As mentioned in the previous section, the paradigm of separating control and user planes brings significant benefits to the table. Traditional integrated CCAP boxes running on dedicated hardware were not able to benefit from the approach where control plane, user plane and physical modulation were part of the same hardware box. However, with the adoption of Distributed Access Architectures the paradigm has shifted so that the modulation function has moved to the RPD and now the CCAP core may be an IP-in/IP-out only box. This shift brought the possibility of moving CCAP software functions from dedicated hardware onto commercial off-the-shelves (COTS) servers and generated the term virtual CCAP as seen in Figure 12

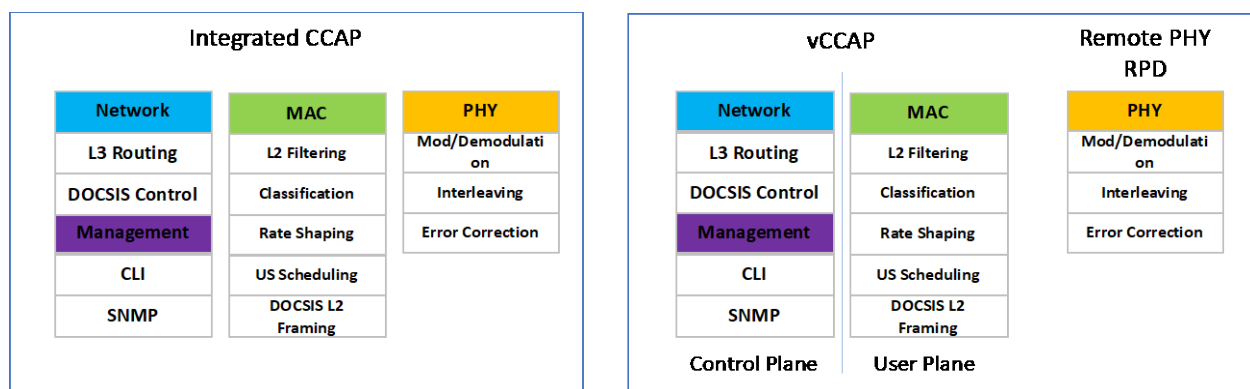


Figure 12 - Integrated CCAP vs virtual CCAP Functions

Two conclusions can also be drawn from Figure 12:

- The CCAP's different software components can be grouped to exploit the benefits of microservices which were analyzed in section 2
- At the same time, it also makes sense to separate the control and user plane functions, as there is an advantage of potentially being able to put them in separate locations.

The above points support using containers for the vCCAP user plane and control plane functions. Now one could think that a user plane container can be the equivalent of integrated CCAP MAC card which typically serves 6 to 8 HFC serving groups, but in reality, that constraint does not exist anymore. A user plane container can serve a single serving group as seen in Figure 13; and at the same time scale to manage multiple user plane functions.

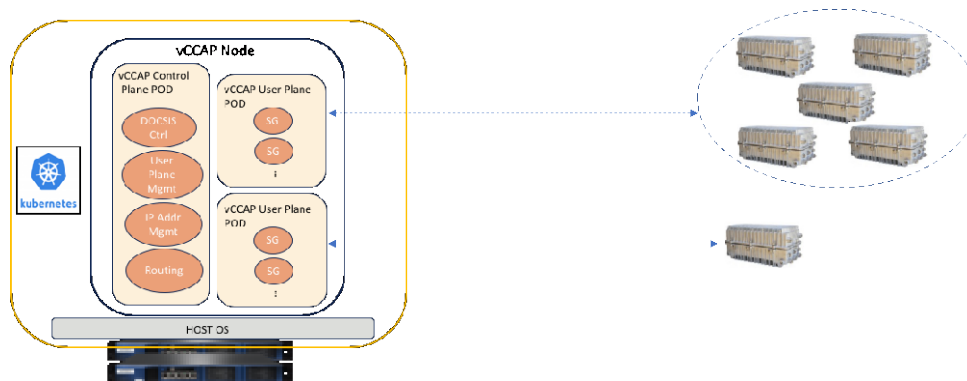


Figure 13 - vCCAP Node to Container Mapping

No matter how many serving groups per container one decides to use, it is evident that the quantity of containers will be significant, so as analyzed in section 2, a container orchestration engine would be also mandatory in order to manage the growing number of containers. However the benefits of container orchestration are not only related to the onboarding of the containers but also how they can handle dynamic scale-in and scale-out of containers based on server utilization or network traffic, or even on redundancy management in the case of software or hardware failures.

As an example, architecture, a container farm is shown in Figure 14 using Kubernetes as the container orchestrator. Kubernetes relies on control-plane and worker nodes; the control-planes take care of the onboarding, deployment and monitoring of the groups containers or PODs which are dynamically deployed in the worker nodes on multiple datacenters on the operator's network.

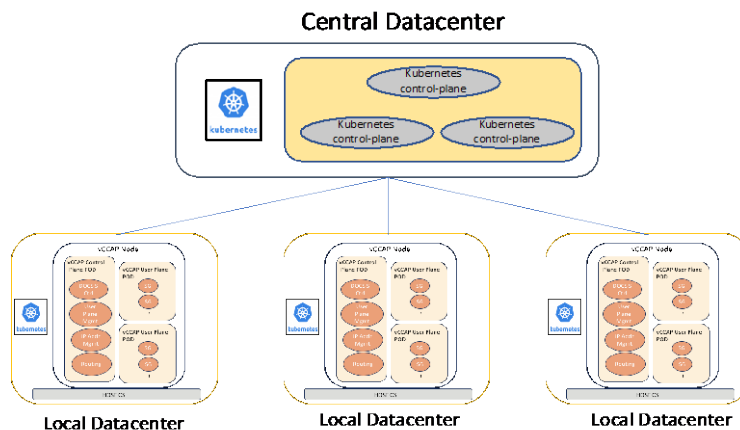


Figure 14 - vCCAP Kubernetes Managed Architecture

As mentioned before, Kubernetes can monitor health status of the running pods. In the example in Figure 15, if it detects a failure on the hardware or software of a user plane pod, it can switch its operation to a redundant POD.

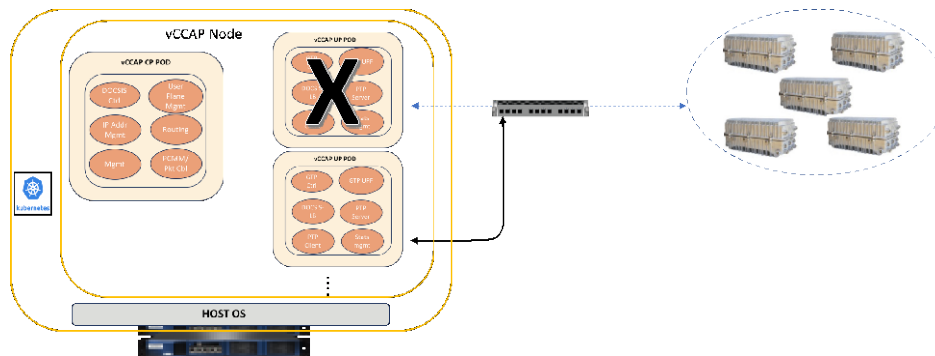


Figure 15 - vCCAP Container Based Redundancy

Perhaps one of the most important advantages of deploying virtualized CCAP is the flexibility to support different types of architectures, going from a totally distributed model where the computing power can be in the hubsites (which can be repurposed as data centers) to a totally centralized model where the computing power is a main datacenter, to any combination in-between. Figure 16.

Each model has its advantages and disadvantages depending on space availability, bandwidth consumption and server usage efficiency, however having the ability to mix them and use the most efficient model for each case is one of the key highlights of vCCAP. This is known as a Hybrid model.

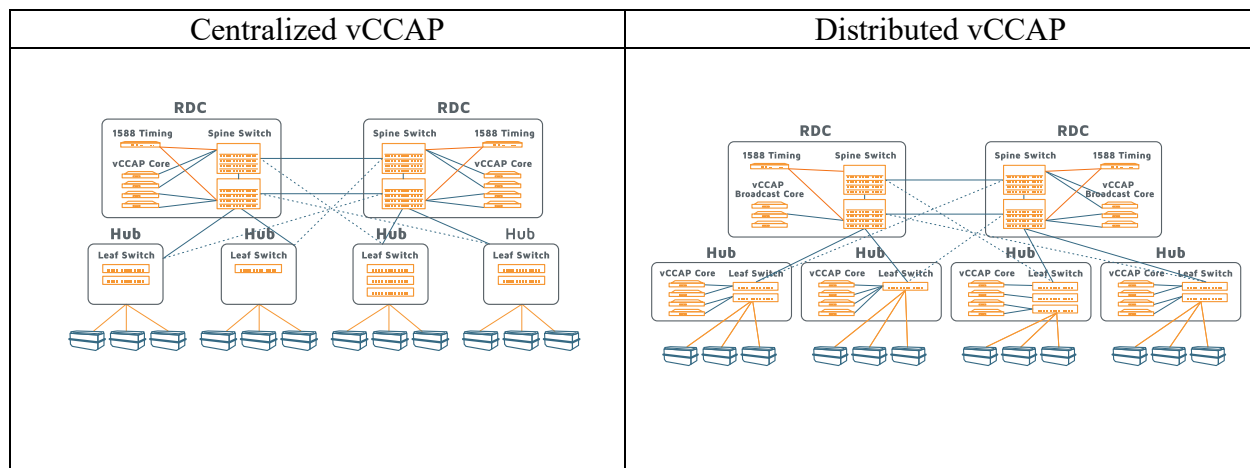


Figure 16 - vCCAP Deployments Models

5.3. Multi-access Edge Computing (MEC)

Multi-access edge computing (MEC) is a cloud environment located at the edge of the network, in close proximity to end users and coupled with the service provider's network infrastructure. Even before 5G is rolled out, current fixed and mobile networks can already enable support for these challenging use cases by using MEC technology. MEC is able to offer low latency and high bandwidth, and, in addition allows services to be deployed in different industrial premises such as road infrastructure, airports, and factories, bringing computing power where it is needed

most. MEC is a key enabler for low latency IoT technology and having this computing power on a node in the network can provide a big number of extra benefits apart of the network function virtualization (Porambage, Okwuibe, Liyanage, Taleb, & Ylianttila, 2018). ETSI Industry Specification Group MEC were the pioneers in creating a standardized computing platform for mobile networks applying network edge related use cases. (Giust, et al., 2018)

5.4. Virtual MAC Manager and Remote MAC/PHY

The Remote MAC-PHY technology moves both the DOCSIS MAC and PHY layers down to the Remote/Fiber Node. The link between the Headend and the node is essentially a Layer 2 connection using Ethernet (Cable Television Laboratories, Inc, 2015). This new device is called the Remote MAC Device (RMD), where this device can be integrated with a Remote PHY Device (RPD) as in Figure 17 or both functions can be in different devices as in Figure 18.

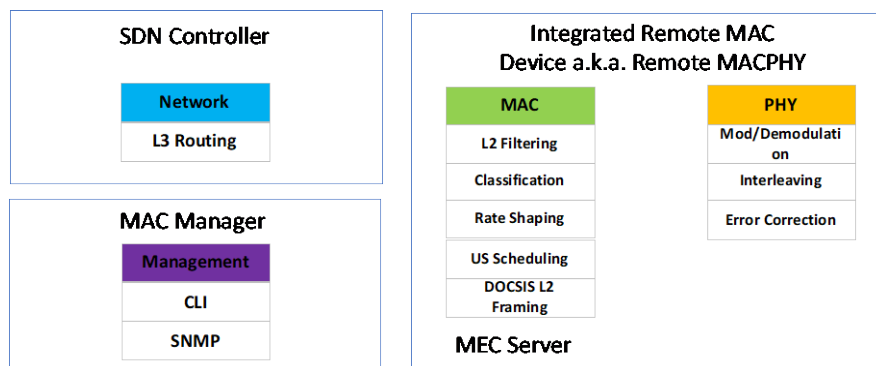


Figure 17 - Integrated Remote MAC-PHY Device

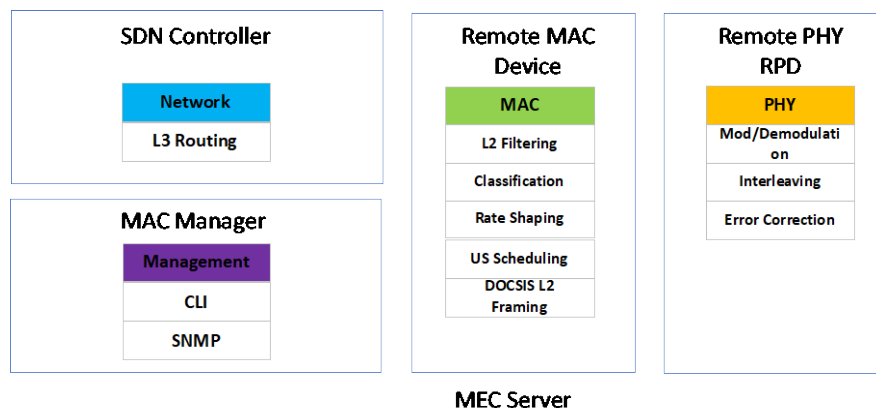


Figure 18 - Decoupled Remote MAC - Remote PHY Devices

It is particularly relevant to highlight that as mentioned in section 5.2, the MAC function is the user plane of a virtual CCAP, which could be run as a container in a cloud native platform. So, in this architecture the Remote MAC Device doesn't need to be more than a Multi-access Edge

Computing server running on a hardened node enclosure, which is automatically managed by container orchestration engine such as Kubernetes.

At this stage the only remnant of the CMTS is the MAC manager which is a centralized software function which provides management access such as command line interface and monitoring data of the CCAP network function running on the remote devices.

6. Future and Convergent Core Functions

Cable Operators can use this technology shift to embrace Fixed Mobile Convergence (FMC) by putting edge network functions at the hub sites which will become virtualization datacenters. As mentioned before, CUPS could enable a common user plane function for cable access, FTTH access and mobile access distributed, while the control plane could be located in centralized datacenters. This is aligned with the decomposed vCCAP and RPHY trend that the industry is following (Asati & Bernstein, 2019). Virtual BNGs and 5G Virtual EPCs can run in the shared cloud environment providing a high level of optimization and integration.

In Figure 19, a potential evolution path for core access networks is presented, where the virtualization of the core and densification of the access will be the first step towards a real Fixed and mobile converged network.

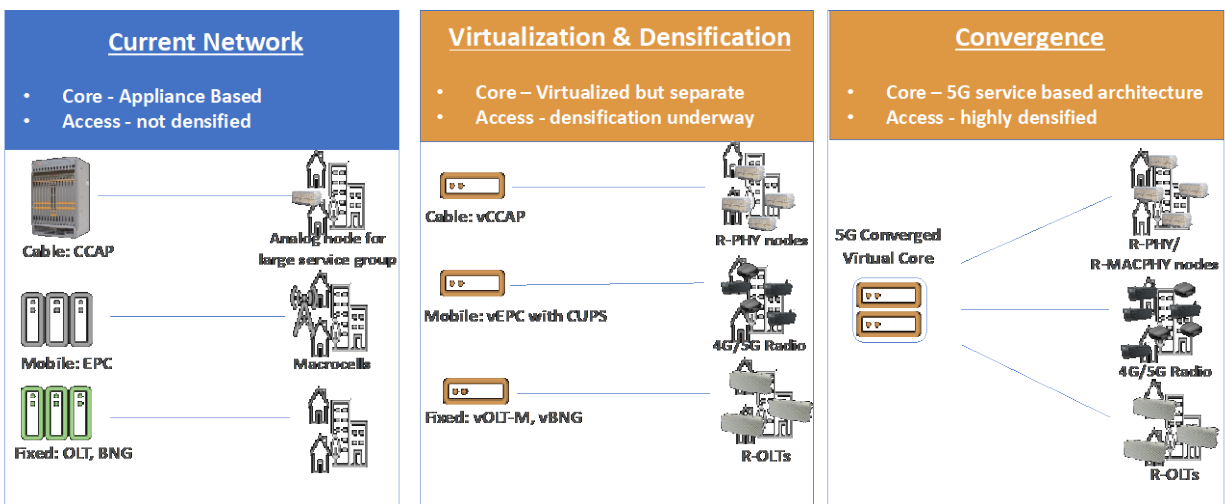


Figure 19 - Fixed Mobile Convergence Evolution

7. Conclusion

This paper described the state of current virtualization and containerization technologies, discussing benefits and disadvantages applied to cloud technologies. A review of virtual and private cloud was done with a focus on the implementation of virtual network functions and how a continuous integration and deployment approach will help in making networks more agile and flexible.

Next, an architecture proposal was presented for deploying a virtual CCAP network function leveraging the benefits of cloud native platforms together with the benefits of control and user

plane separation. A brief discussion of Multi Access edge computing aligned with the transition path to Flexible MAC Architectures and remote MAC-PHY was shown.

Lastly, a vision for future Fixed Mobile Convergence is presented with the evolution through virtualization of the network functions and densification of the access network.

As a final conclusion: HFC networks need to evolve into adopting the benefits provided by virtualization, containerization and cloud technologies. As reviewed during the paper, those technologies provide not only big improvements in efficient usage of the infrastructure but more importantly bring the ability to deploy new services in a quick and agile manner, something that will be a key factor for cable operators to continue its success in the new world of Fixed Mobile Converged networks.

Abbreviations

BNG	Broadband Network Gateway
CAPEX	Capital Expenditures
CCAP	Cable Converged Access Platform
CD	Continuous Deployment
CI	Continuous Integration
CMTS	Cable Modem Termination System
CPF	Control Plane Function
CPU	Central Processing Unit
CUPS	Control and User Plane Separation
DevOps	Software Development / IT Operations
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
FTTH	Fiber to the Home
IaaS	Infrastructure as a Service
IoT	Internet of Things
IP	Internet Protocol
ISBE	International Society of Broadband Experts
MaaS	Metal as a Service
MAC	Media Access Control
MANO	Management and Network Orchestration
MEC	Multi-Access Edge Computing
NFV	Network Function Virtualization
OS	Operating System
PaaS	Platform as a Service
HFC	hybrid fiber-coax
PHY	Physical
R&D	Research and Development
RMD	Remote MAC Device
RPD	Remote PHY Device
SaaS	Software as a Service
SCTE	Society of Cable Telecommunications Engineers

UPF	User Plane Function
vCCAP	Virtual Cable Converged Access Platform
VNF	Virtual Network Function
VT	Virtualization Technology

Bibliography & References

- 3GPP. (2018). *System Architecture for the 5G System 3GPP GPP TS 23.501 15.0.0*. 3GPP.
- Adams, K., & Agesen, O. (2006). A comparison of software and hardware techniques for x86 virtualization. *ACM SIGOPS Operating Systems Review*, 40(5).
- Asati, R., & Bernstein, A. (2019). Cable Edge Compute: Transforming Cable Hubs into Application-Centric Clouds. *SCTE Cable-TEC 2019*. New Orleans, LA.
- Broadband Forum. (2018). *Cloud Central Office Reference Architectural Framework TR-384*. Broadband Forum.
- Cable Television Laboratories, Inc. (2015). *Distributed CCAP Architectures Overview Technical Report*. Cable Television Laboratories, Inc.
- European Telecommunications Standards Institute. (2012). Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. *SDN and OpenFlow World Congress*. Darmstadt, Germany.
- European Telecommunications Standards Institute. (2013). ETSI NFV Management and Orchestration - An Overview. *IETF #88*. Vancouver, CA.
- Fitzgerald, B., & Stol, K. (2017). Continuous software engineering: A roadmap and agenda. *The Journal of Systems and Software*, 123, 176–189.
- Giust, F., Sciancalepore, V., Sabella, D., Filippou, M. C., Mangiante, S., Featherstone, W., & Munaretto, D. (2018). Multi-Access Edge Computing: The Driver Behind the Wheel of 5G-Connected Cars. *IEEE Communications Standards Magazine* (September), 66-73.
- Hu, S., Qin, F., Li, X., Chua, T., Eastlake, D., Wang, Z., & Song, J. (2018, October 22). *Architecture for Control Plane and User Plane Separated BNG*. Retrieved July 31, 2020, from <https://tools.ietf.org/id/draft-cuspd-tgtwg-cu-separation-bng-architecture-02.html>
- IBM. (2020, March 3). *Public Cloud Cloud*. Retrieved July 31, 2020, from <https://www.ibm.com/cloud/learn/public-cloud>
- Morabito, R., Cozzolino, V., Ding, A. Y., Beijar, N., & Ott, J. (2018). Consolidate IoT Edge Computing with Lightweight Virtualization. *IEEE Network*.
- Motjaba, S., Babar, M. A., & Zhu, A. L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, 5(2017), 3909-3943.
- National Institute of Standards and Technology . (2011, September). *The NIST Definition of Cloud Computing*. Retrieved July 31, 2020, from <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>
- Popek, G. J., & Goldberg, R. P. (1974). Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM*, 17(7), 412-421.
- Porambage, P., Okwuibe, J., Liyanage, M., Taleb, T., & Ylianttila, M. (2018). Survey on Multi-Access Edge Computing for Internet of Things Realization. *IEEE Communications Surveys & Tutorials*.

- Redhat. (n.d.). *What is container orchestration?* Retrieved July 31, 2020, from <https://www.redhat.com/en/topics/containers/what-is-container-orchestration>
- Simic, S. (2019, April 19). *Containers vs Virtual Machines (VMs): What's the Difference?* Retrieved from <https://phoenixnap.com/kb/containers-vs-vms>
- The Linux Foundation. (n.d.). *What is Kubernetes?* Retrieved July 30, 2020, from <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>
- Yu, Y., Silveira, H., & Sundaram, M. (2016). A microservice based reference architecture model in the context of enterprise architecture. *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference*. Xi'an, China.