A 5G Americas White Paper

# 5G AND THE CLOUD

Dec. 2019

5G americas

# TABLE OF CONTENTS

# 1. INTRODUCTION

5G is being developed for an extremely mobile and hyper-connected society and to accommodate the need for emerging use-cases and business models. The evolution from Fourth Generation (4G) to Fifth Generation (5G) is significant and unlike previous evolution journeys for Communication Service Providers (CSP). 5G is not only affecting the consumer, enterprise and industrial space with new use cases and business cases allowed by enhanced capabilities, it's also encouraging operators and vendors to re-think and re-architect how the network will be built and managed.

To accommodate the myriad of divergent use-cases proposed in 5G, it is imperative that the network should be architected for inherent flexibility and efficiency. This might be accomplished by making network functions modular so that they can be quickly deployed and scaled on demand to provide service agility and cost-effectiveness. Cost-effectiveness can be achieved in cloud native paradign by the efficient bin-packing in the containers, more fault tolerance and deployment flexibilities.Efforts in this direction were started by adoption of Network Function Virtualization (NFV) and Software Defined Networking (SDN) related technologies to make Mobile Core (MC) applications cloud ready. Along the way, valuable lessons have been learned. In many cases, the true potential of NFV was perhaps wasted a bit; instead of bringing virtual network functions to a common NFV Cloud, the mobile industry built Cloud silos dedicated to specific Virtual Network Functions (VNF). In some instances, physical Network Functions were ported as VNFs without necessarily addressing the required changes to the underlying software architecture; this resulted in deployment and operations complexities.

The next phase of innovation in 5G Next Generation Core (NGC or 5GC) is focusing on adopting Cloud Native technologies and architecture. Ultimately, the objective of adopting Cloud Native principles and technologies by Service Providers is to achieve web scale and attain the economies of scale. The biggest motivation towards the push to Cloud Native comes from the 3GPP-proposed Service Based Architecture (SBA) in Release-15 (Rel-15) and enhancement to Service Based Architecture (eSBA) in Rel-16 (which extends the service concept from the 5GC control plane to the user plane function). Taking the cue from 3GPP, European Telecommunications Standards Institute (ETSI) has published augmented NFV referenced architecture to accommodate for the Cloud Native Network Function (CNF) and enhancement to NFV framework to include Zero-Touch, Containers, Load Balancers and more as part of the reference architecture.

With the decomposition of Network Functions and Network Services being exposed through Application Protocol Interfaces (API), the push towards Cloud Native in 5G Core is getting more realistic. Also, various use cases proposed and envisioned for 5G demand a more scalable, reliable, distributed and flexible Core Network making the case for adoption of Cloud Native for Service Providers stronger. There is also a lot of hype and confusion surrounding Cloud Native in general and the adoption of Cloud Native architectures and technologies within 5G in particular. Leveraging a service-based architecture and Cloud Native Network functions in the 5G core has proved challenging during various proof of concept efforts. While providing implementation flexibility by breaking up existing network functions into microservices, Cloud Native Functions struggle to maintain the packet processing and latency requirements of 5G.

While control and management plane applications are the immediate candidates for the Cloud Native transformation, operators and vendors are also exploring the migration of Radio Access Network (RAN) functions towards the cloud and container centric environment. This is targeted to help the operator develop a common orchestration and management model for both Core and RAN Network functions. RAN functionality is extremely sensitive to latency and throughput, typically addressed with custom hardware accelerators for performing Layer1 and Layer2 radio layer functions. Even when most parts of the RAN

functions will be migrated to Cloud Native platforms, some of the highly computation-oriented and latency sensitive components are expected to stay outside the scope of Cloud Native for some time.

This 5G Americas whitepaper examines the road to a Cloud Native 5G core by introducing and reviewing the Cloud Native landscape as a technology (Sections 1 and 2). This landscape has a rich open source-supporting ecosystem and has enabled the Cloud Native approach to build and run applications that fully exploit the benefits of the cloud computing model. This model includes services architectures, infrastructure-as-code, automation, continuous integration/delivery pipelines, monitoring tools, and more.

The requirements of 5G are then reviewed, and the Cloud Native model is applied to them (Section 3). Regardless of the fact that the Cloud Native model is an attractive direction for many good reasons, it does not map completely to the Communication Service Provider's (CSP's) mobile network operational needs. That said, the standardization community for 5G has adopted a modular approach to the way the 5G core is architected, encouraging a Cloud Native approach to its implementation. However, there are gaps in the architecture, indicating that the telecommunications industry is just at the beginning of the Cloud Native journey.

How Cloud Native is being implemented for 5G (Section 4) and what migration strategies will be employed are the next areas explored in the whitepaper. Best practices and some predictable issues around the reality of deployment are developed in the report with the conclusion that there are application dependencies on several layers of the Cloud Native stack, particularly workflow management, container engine and minimal Operating System (OS) layers. If Cloud Native is to fulfil the expectations placed on it by 5G implementation, these dependencies need to be overcome. Because the latest focus of the 3GPP Release 17 (Rel-17) standard includes the application of 5G to enterprise and verticals, and analysts are offering market evidence on the adoption and timing of 5G, there is a window of three to four years during which the Cloud Native issues relevant to 5G need to be resolved if the required network flexibility is to truly unlock the (elusive) new 5G business models. Of course, the shift to the cloud-native mind-set, especially for the telecom sector, will not be easy and will take a phased approach to reap full benefits.

The objective of this whitepaper is to look beyond the hype and provide deep insight into 5G Cloud Native design. Best practices and deployment models for Cloud Native Network Functions, to keep processing latency in line with 5G requirements while providing the required robust implementation essential for service provider deployment, is outlined and explored. 5G and the Cloud whitepaper studies deployment models for Cloud Native Network Functions (CNFs) and provides reference architectures for CNFs along with recommendations for a solution blueprint.

## 2. INTRODUCTION TO CLOUD NATIVE

Cloud Native is a relatively new concept being introduced into the telecommunications domain. This section of the whitepaper provides a baseline for the Cloud Native ecosystem and the path to its current status.

The Internet Technology (IT) and enterprise worlds have invested in technologies and methodologies for agility and efficiency with the result of faster and less expensive services development and deployment. The journey to Cloud Native started with the introduction and adoption of virtualization-related technologies (Xen Hypervisor, Intel VT-x, KVM hypervisor, Linux Containers, and etcetera) which led to more efficient usage of the system resources and an agile way to package and deliver applications (see Figure 2.1). The wide-spread adoption of virtualization techniques in the data center was pre-dominantly responsible for the genesis of Cloud as it's known today.

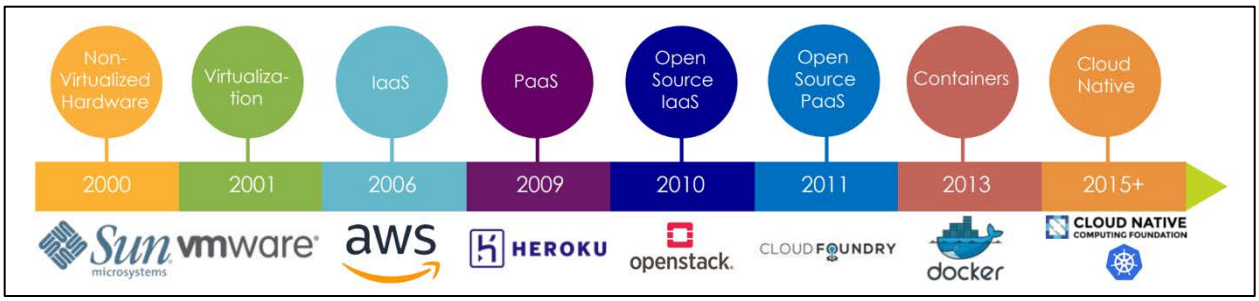


**Figure 2.1. Evolution of Cloud Native Stack.**[1]

Adoption of Cloud, even in the IT and Enterprise world, has been a major endeavor to move to an "as-a-service" paradigm while bringing dynamicity, flexibility and scalability as system hosting applications became more transient in nature and services endpoints became more predominant. Moving to the cloud led to the emergence of Cloud Native as an innovative approach of building and running applications in such a way to reap the true benefit of cloud computing models. It is how, not where, applications are created and deployed; thus deploying in a Public, Private or Hybrid Cloud environment has no bearing on an application being Cloud Native. The key is deploying in the "cloud-way" – even if it means a re-architecture.

About the same timeframe as the IT industry was undertaking cloud adoption, the telecommunications industry was also embarking on their own modernization journey. It was fueled by the publication in 2012 of the *Network Functions Virtualization – Introductory White Paper* at the SDN and OpenFlow World Congress, Darmstadt-Germany,[2] which proposed a new reference architecture based on virtualization for network functions moving away from proprietary hardware appliances allowing for accelerated service innovation.[1] Network Function Virtualization (NFV) along with complementing Software Defined Networking (SDN) technology and adoption of an All-Internet Protocol (IP) transport network, positioned the telecommunications industry to be well-equipped to adopt cloud technologies to transform the network and build a better ecosystem to promote open innovation. Adopting Cloud Native methodologies is the next logical step in the evolution, with 5G demanding the future network to be more flexible, agile and robust to satisfy the service delivery requirements for a variety of different and new use cases.

## 2.1 CLOUD NATIVE: DEFINITION AND CORE TENETS

Defining Cloud Native can be tricky. Cloud Native is about the approach to how applications can be built and operated in the Cloud Computing construct. It has become prominent as more and more application deployments are being done in cloud. Concepts of being Cloud Native can be applications to the technologies, architecture and tooling.

Cloud Native Computing Foundation (CNCF), which is an open source software foundation under the umbrella of Linux Foundation (LF), defines Cloud Native as: Cloud Native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public,

[1] *Evolving Cloud Native Landscape*, presentation by Chris Anisczyk, Cloud Native Computing Foundation. Container Days, Japan. December 2018.
[2] *Network Functions Virtualization – Introductory White Paper*. SDN and OpenFlow World Congress, Darmstadt-Germany. 22 October 2012.

private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative Application Programming Interfaces (API) exemplify this approach.[3]

CNCF provides structure and constraints for being Cloud Native, requiring that the applications being developed and architected use microservices methodologies to be deployed as Containers. Traditional applications when broken down into small, reusable components are referred as "microservices." To be properly classified as Cloud Native, microservices need to be "stateless" meaning there has to be separation of processing from the associated data and the storage of the data in Cloud. Dynamic Orchestration is another critical pillar of being Cloud Native. That's where the CNCF-hosted Open Source container orchestrator "Kubernetes"[4] plays a crucial role in actively scheduling and optimizing resource utilization while providing observability, resiliency and an immutable infrastructure.

In the next subsections, the supporting characteristics and infrastructure for Cloud Native needs are reviewed including: the decomposition and modularization of functionality; the need for light-weight virtualization; the service mesh; the reliance of Development and Operations (DevOps) and Continuous Integration/Continuous Delivery (CI/CD), and consequently the need for an appropriate tool chain.

### 2.1.1 MICROSERVICES FOR ARCHITECTURE

Microservice architecture is a design philosophy based on a general consensus, however there are no strict definitions or standards bodies prescribing for the decomposition of Business requirements into Domains. Further decomposition breaks each domain into Services (applications) that fulfill the Business requirements; Business requirements are built with the objective to prevent failure. Inter-Domain communications are clearly and concisely defined in a bounded context. This has emerged from a Domain-Driven-Design (DDD) architectural pattern. The term "Micro" in Microservices refers to the scope of functionality rather than the scale required to meet the availability and reliability required in an operational environment.

Ideally, each Microservice is a single loosely coupled service (Figure 2.2) where a monolithic gateway network function includes a Firewall (F/W), Routing, Deep Packet Inspection (DPI), and Charging and Datastore. When re-architected using microservices principles, all the functions are deployed individually and with clear interfaces and dependencies. As architected, the individual functions, such as Charging, can be scaled and managed in isolation. Each microservice, by design, needs to provide well-defined, Representational State Transfer (REST) software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, are called RESTful. The appropriate Restful API, to its peer, will provide a constant endpoint for the service being delivered by that particular microservice. This results in an easier upgrade of the specific function represented by each microservice compared to upgrading the complete network function allowing for a faster delivery model.

[3] "CNCF Cloud Native Definition v1.0", Github. 19 September 2019.
[4] https://kubernetes.io/, 19 September 2019.

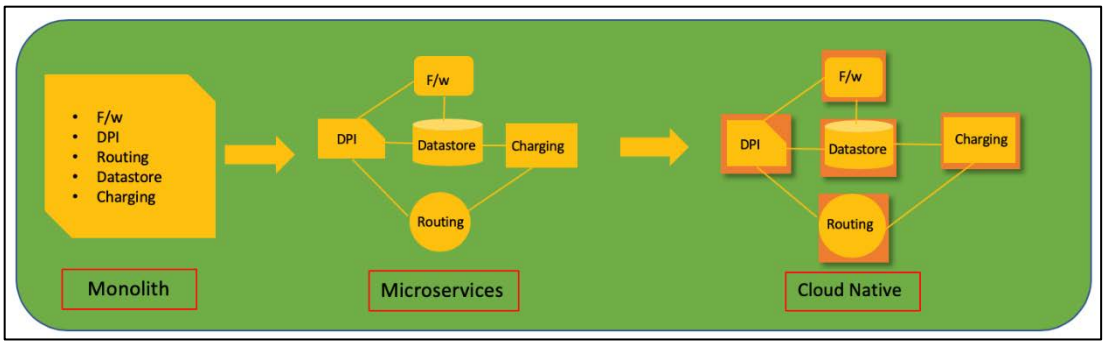


**Figure 2.2. Evolution to Cloud Native Network Function.**

Breaking the monolithic applications into microservices can also lead to challenges in developing, deploying and operating many more entities. As a result, DevOps along with automation tooling is needed for managing the increased complexity. DevOps is an important tenet of 'being Cloud Native' and is explained in section 2.1.4. Another important point, is that Microservices Architecture does not prescribe that it has be a container-based virtualization to deploy the services, but Cloud Native architecture does have that requirement.

### 2.1.2 CONTAINERS AND CONTAINER ORCHESTRATION

Breaking down the application software monoliths into a set of smaller self-contained components, as shown in Figure 2.3, and as required in microservices architecture, can have drawbacks if there is no change to deployment and the application packaging model. For example, if each self-contained monolithic application, which often has a large inherited software footprint similar to network functions, is decomposed and deployed as a set of virtual machines as per the microservices architecture, the new deployment may not be efficient. In effect, an efficient monolithic self-contained network function with low footprint and low power consumption may have been traded for a sprawling distributed network function of sub-monoliths with many virtual machines, a larger footprint and increased power consumption. This may be viewed as counterproductive.

To counter this, the idea of moving to operating systems virtualization techniques with much lower overhead seems like a natural path to take. This is where containers make an entry. As represented in Figure 2.2, it is evident that bringing in containers instead of Virtual Machines (VMs) reduces the overhead as the resource heavy hypervisor layer and the guest operating system are replaced by a container runtime to provide similar isolation but in a more efficient manner. Container runtime provides core primitives for:

- Management of containers on the host kernel
- Container execution
- Supervision of container deployment state, which includes managing network interfaces, image management and more

Examples of popular container runtime are Docker,[5] CRI-O,[6] LXC,[7] & RKT.[8]

Deploying the microservices in distinct containers allows the network functions to be scaled at a much higher granularity than would be possible when using Virtual Machines (VM). This means the service can be tuned optimally to deliver the service as needed with the most efficient resource usage. The containers offer portability; the container image is built once and can be potentially deployed numerous times into any supported runtime environment. This allows for a rapid deployment and recovery due to its being lightweight compared to VMs. The concept and technologies (namespaces, cgroups, solaris zones, and etcetera) which support containers have existed since the early 2000s and became popular as industry realized that there is huge gain in performance, resource utilization, and overall efficiency if containers are used compared to VMs. There are two types of containers:

- **system containers –** which, more or less, behave like virtual machines by sharing the kernel of the host operating system with user space isolation, where multiple services can be deployed

- **applications containers** - designed to be packaged and run single service, which is suited for the microservices architecture

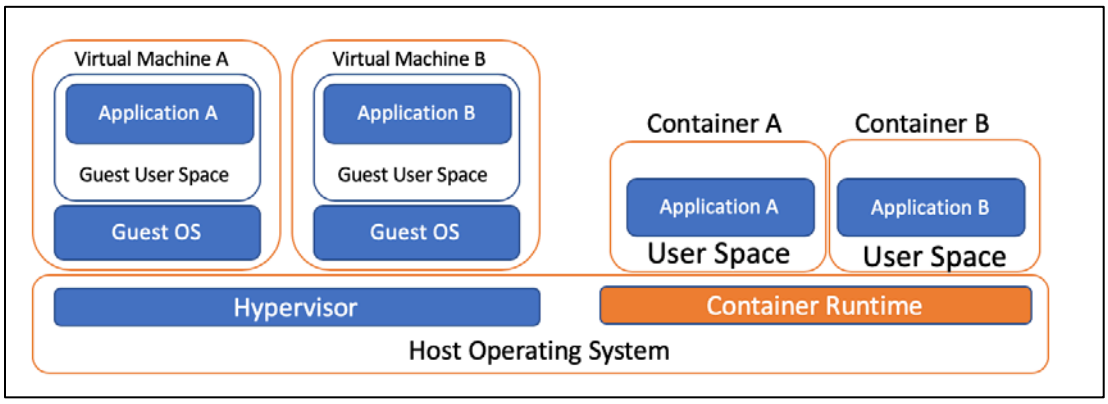


**Figure 2.3. Virtual Machines versus Containers.**

Using containers for deploying applications and services allows for an immutable infrastructure. Therefore, when applications, or the service, are deployed as containers and need to be updated, new container instances are spun up utilizing the updated image. The paradigm of immutable infrastructure is important for supporting complex network functions as it simplifies the deployment by avoiding "configuration drift" and simpler recovery mechanism as it only entails regressing to an older container image.

Due to the scalable, portable and dynamic nature of the containers, managing the lifecycle becomes a daunting task. That's just the law of entropy catching up. Container orchestration framework is the critical piece in the puzzle for solving the lifecycle management problem in the Cloud Native ecosystem. The scope of this framework entails:

- Container provisioning and deployment

[5] https://www.docker.com.
[6] https://cri-o.io.
[7] https://linuxcontainers.org.
[8] https://coreos.com/rkt/.

- Resource allocation and placement
- Redundancy and availability of containers
- Horizontal scaling
- Service discovery and load balancing
- Health monitoring
- Application configuration

There are several container orchestration solutions, however, Kubernetes has the biggest momentum and industry support. Almost all the vendors and operators in the world consider Kubernetes as the de-facto container orchestrator for managing 5G workloads. Kubernetes is the Open Source container orchestration engine originally developed by Google. It is a cluster-based container orchestration tool consisting of a master (multiple masters in High Availabilities scenario) which orchestrates lifecycle of service – including container scaling and recovery of failed containers and workers which hosts multiple pods. Pod is the basic unit of deployment in Kubernetes which consists of one or more containers sharing the same IP address and port space. Kubernetes Service represents a logical set of Pods and enables external traffic exposure, load balancing and service discovery for those pods. Details of container orchestration with relations to CNFs are reviewed in later sections.

### 2.1.3 SERVICE MESH FOR NETWORKING MODEL

No introduction to Cloud Native can be complete without discussing service mesh. Adopting microservices architecture results in complex service and application topologies as the number of components increase (Figure 2.4). This leads to new set of problems in Cloud Native deployments: service discovery; traffic management; security and policy management; and observability. The problems can be solved by either making the problem a part of the application itself, where additional logic is coded into the business logic of the application, or, bringing in a separate infrastructure solution called service mesh.

"A service mesh is a dedicated infrastructure layer for handling service-to-service communication."[9] It's responsible for ensuring reliability, security, intelligent routing and traceability of communication among the services that comprise a Cloud Native application. The service mesh manages the "east/west" traffic running between services by injecting a service proxy next to each application workload. A service proxy is the dedicated entity to the particular workload. A collection of service proxies constitutes the "data plane" of the service mesh. All intra-microservices communication messages traverse its service proxy. The proxies become the source of metrics for all the requests and can be utilized to transform inter-service communication, providing powerful behavior, such as: retries, timeouts, circuit breaking, service discovery, load balancing, secure tunneling, and much more. There is a control plane consisting of entities that live outside of the inter service communication path for supporting the data plane. The control plane collects metrics, establishes policy, and affects configuration changes to the data plane. It also provides API access to configure and manage the service mesh.

[9] "What's a Service Mesh? Why do I need one?" William Morgan, Buoyant.io, Cloud Native Computing Foundation, 26 April 2017.

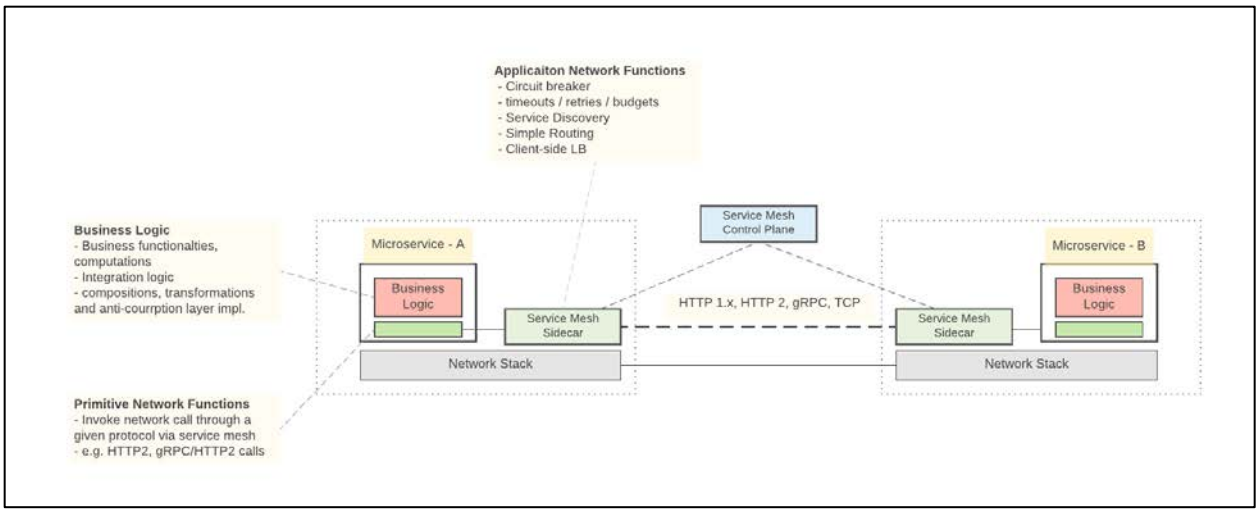


**Figure 2.4. Service to Service Communication with Service Mesh.**[10]

The most popular way of implementing service proxies is by using a sidecar design pattern.[11] All the inter-service communication logic is abstracted in sidecar and runs alongside the application container, as shown in Figure 2.4. As now evident, service mesh implementations are based on a control plane/data plane architecture. Service mesh creates an abstraction layer above the transport layer of the Transmission Control Protocol/Internet Protocol (TCP/IP) stack, decoupling the instrumentation and operational concerns of the service from the application business logic and leading to increased service velocity. The development of solutions for service mesh are still in their infancy stage and vying for market share are multiple Open Source service mesh projects, such as: Istio,[12] Linkerd[13] and Consul.[14] Current shortcomings in the existing service mesh implementations include: added complexity due to introduction of infrastructure layer; and performance impact on application and unsuitable for heterogeneous and multi-cluster topologies. However, there is momentum behind service mesh, and it will realize its potential when it achieves complete transparent integration with the container orchestrator. There are architectural implications of service mesh in 5G Core architecture (discussed in later sections) which are evolving as "SDN for Code."[15]

### 2.1.4 DEVELOPMENT AND OPERATIONS FOR PROCESS WORKFLOW

Development and Operations (DevOps) is a philosophy and practice focused on communication, collaboration, automation, and agility within operations and development teams. The whole objective of DevOps is to attain a more aligned and synchronous development and operations team. In particular, it makes organizations more adaptable for technological changes to avoid technology lock-in, reduces time-to-market and provides better team structure to support services. Although, DevOps can be equally applications to both monolithic and microservices architecture, DevOps practice becomes more critical for the success of the Cloud Native implementation, as the modularization of Cloud Native application increases. Organizations adopting DevOps to support Cloud Native architecture will be at a strong

[10] "Service Mesh for Microservices", Kasun Indrasiri, Medium.com, 15 September 2017.
[11] https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar.
[12] https://istio.io/.
[13] https://linkerd.io/.
[14] https://www.consul.io/intro/index.html.
[15] https://glasnostic.com/blog/should-i-use-a-service-mesh.

competitive advantage as their organization will be well equipped and aligned to handle the complexities of the sophisticated and at-scale applications which require new tools, new processes and a product-focused team. This leads to the realization that Cloud Native adoption cannot succeed without implementing a mature DevOps practice.

DevOps, as a philosoph,y has been evolving since its introduction a decade ago. When introduced, the major focus was to remove the barrier existing between the developers and operations. However, with the adoption of more automation, the move towards Cloud Native infrastructure and the rise of Infrastructure as Code (IaC), the scope of DevOps has broadened to encompass not only technical teams pertaining to networking and security but also non-technical teams, like marketing, business and design, focusing on creating workflow to maximize organizations' core business goal. The increased operational complexity in Cloud Native architecture helps to bring networking and security under the purview of DevOps and is leading to the emergence of Development, Security and Operations (DevSecOps) and Network, Development and Operations (NetDevOps) practices.

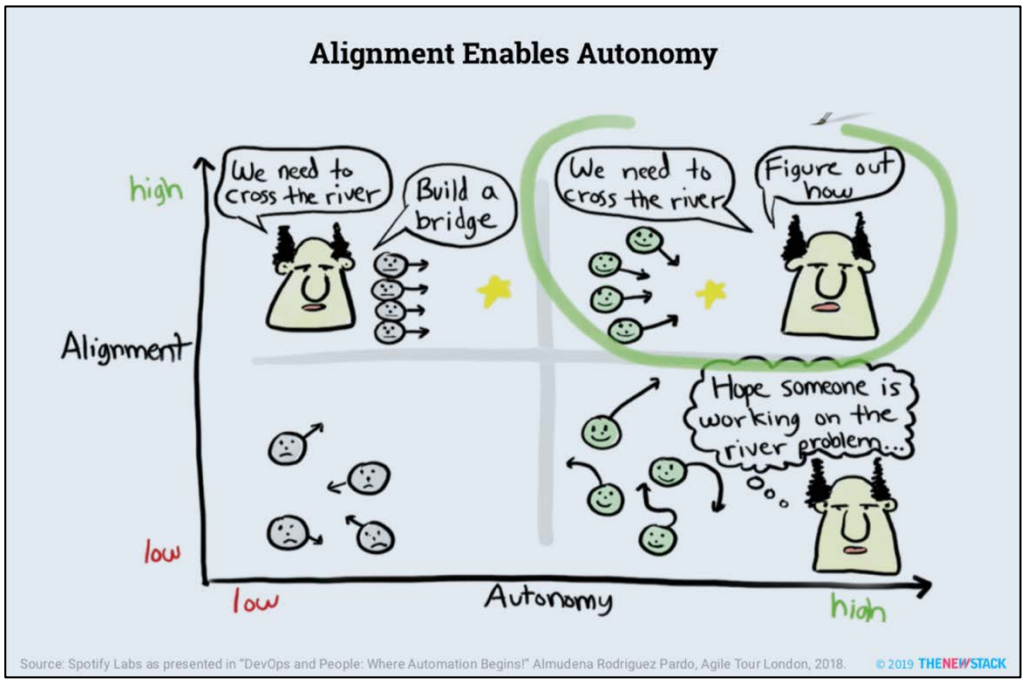


**Figure 2.5. Alignment vs. Autonomy in Cloud Native Architectures.[16]**

"DevOps isn't just about breaking down silos, but about developing skills."[17] It is about an organization's appetite for adapting, re-skilling and readiness to learn from mistakes. Organizations, while adopting Cloud Native architecture, need to have high alignment towards the team's objective based on the defined scope and have the autonomy to operate freely (Figure 2.5). A cookie cutter approach to DevOps cannot be applied as each organization and their requirements from DevOps is unique. They have to develop practices resulting in the creation of a pipeline, and tooling for automation chains that not only exclusively fits those practices, but also will grow and evolve along the way. The responsibility of the building of the

[16] https://ucaat.etsi.org/images/Presentations/2018/16---14h20---Almudena-pardo---UCAAT-2018_Almudena_Rodriguez_Pardo.pdf
[17] https://thenewstack.io/ebooks/devops/cloud-native-devops-2019/

pipeline with tooling to bring in automation (as prescribed by DevOps) falls on Continuous Integration and Continuous Delivery and/or Continuous Deployment (CI/CD).

### 2.1.5 CONTINUOUS "EVERYTHING" FOR TOOLCHAIN

Moving to Cloud Native leads to increased agility and complexity where applications are delivered as vastly sophisticated microservices. Cloud Native's effectiveness is increased when CI/CD tools and practices are implemented. Applications that are natively developed for the cloud platform benefit from CI/CD, since the software development of complex systems gets broken down into a set of smaller, more manageable loosely coupled microservice components, which are integrated into the application with their own delivery pipelines. The greater the number of components, the greater the need for automated delivery.

Continuous Integration (CI) is the automated process of integration of newly developed code to the main source code and with unit testing, this is the first step towards Continuous Delivery (Figure 2.6). It permits the software vendor to integrate newer component releases with existing components, thereby allowing a continuous incremental evolution of a software system leading to the faster introduction of new features. Container is the unit of the deployment in the Cloud Native construct and using it helps in consistency between the development and production environment. Application code wrapped with configuration in containers is shifting the focus to continuous delivery. Containerization of the CI/CD infrastructure allows for isolated builds and makes it more elastic as each build pipeline gets its own container agent.

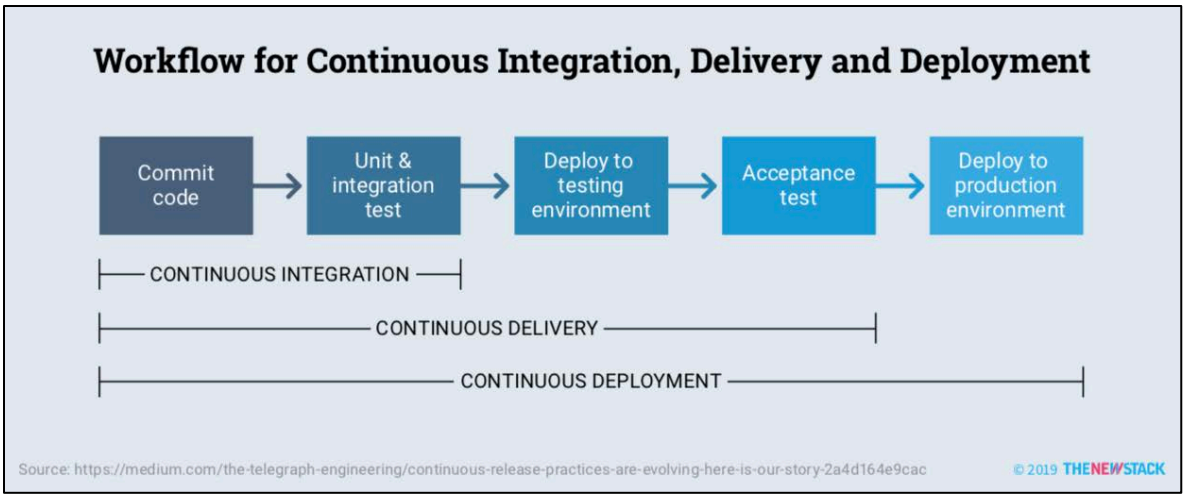


**Figure 2.1. Workflow for Continuous Integration, Delivery and Deployment.**[18]

Available tooling in Kubernetes, the container orchestrator, along with dynamic, composable and immutable infrastructure helps in reducing the complexity of implementing continuous delivery in a Cloud Native framework. Product teams are empowered with delivery ownership using isolated deployment pipelines. Cloud Native CI/CD allows for independent changes to services which can be monitored to ensure applications changes are working as expected in production. Deployment strategies in continuous delivery for Cloud Native architecture, like canary releases (phased rollout of changes to smaller subset of users) and blue-green deployments (previous version of live environment available for quick rollback) can help

[18] https://medium.com/the-telegraph-engineering/continuous-release-practices-are-evolving-here-is-our-story-2a4d164e9cac

test new features with velocity to improve the customer experience, minimize the risk of failure and offer much quicker rollback to stable release in the event of failure.

Properly implemented, a multi-stage continuous delivery pipeline should also include the test automation framework which helps in promoting code and artifacts from one stage to another. Test automation becomes critical in creating the much-needed feedback loop. It is important that creating continuous testing cannot be possible in all the scenarios, and thus procedures for manual intervention in the CI/CD pipelines must be built in. Due to the distributed design of Cloud Native architecture and the continuous nature of software delivery, continuous monitoring and security scanning becomes paramount. Continuous monitoring brings in mechanisms that can indicate the performance of the delivered software so that flaws can be identified promptly, and updates/patches are released continually with minimal intervention. Providing built-in monitoring capabilities and security scans in the CI/CD pipeline helps in enforcing security policies and compliance at scale.

## 2.2 CLOUD NATIVE LANDSCAPE AND OPEN SOURCE FOR 5G

The Cloud Native ecosystem is evolving and growing at a very rapid pace. Figure 2.7 shows the large number of critical projects and tools in Cloud Native being tracked by CNCF; their objective is to drive adoption of a Cloud Native paradigm by fostering and sustaining an ecosystem of Open Source, vendor-neutral projects.[19] The intent of the CNCF landscape is to categorize all the solutions in the Cloud Native ecosystem for: the ease of adoption, the tracking of community engagement, and tracking the code maturity of projects.

CNCF provides sandbox environments for germinating new and exciting projects around Cloud Native technologies. Network Service Mesh, which is the concept of bringing Layer2/Layer3 networking to containers, is an example of a sandbox project in CNCF which is very relevant for telecommunications operators and vendors (and will be discussed in section 4.2.5).

[19] https://github.com/cncf/toc/blob/master/DEFINITION.md.

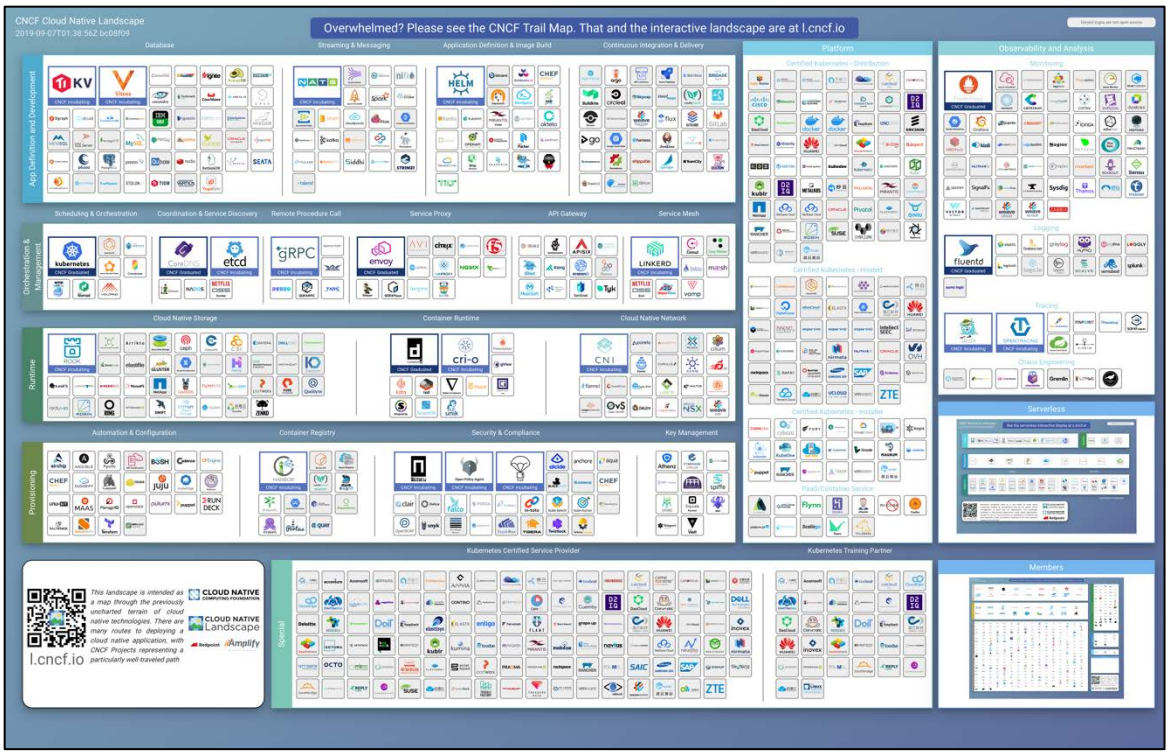


**Figure 2.2. Cloud Native Landscape.**[20]

Kubernetes is the container orchestration project, notable for being one of the most rapidly developed Open Source projects of all time due to the community's contribution and is often referred to as the "Linux of the cloud."[21] CNCF has been a hotbed for hosting Open Source Cloud Native projects contributed by web-scale companies such as: Kubernetes from Google,[22] and Envoy from Lyft.[23]

With 5G development and adoption progressing at full force, Cloud Native is central to the 5GC architecture. This has resulted in vendors and operators becoming active members of the CNCF community. They are bringing telecommunication (telco) requirements and use cases to the community and looking for projects in CNCF to solve their challenges in adopting Cloud Native technologies for 5G. Even the CNCF has taken notice of the trend and has created a "Telecom User Group" (TUG) for telco operators and vendors to identify CNCF projects, or projects' specific features that are critical for Cloud Native adoption. The TUG tracks upstream projects development and provides recommendations on projects for feature development based on operators' use cases.[24]

The objective of CNCF's TUG group is to dispel Fear, Uncertainty and Doubt (FUD) over the adoption of containers and Cloud Native technologies for the telco workloads. The "noisy neighbor" problem is one such area of FUD. The traditional approach to tackling this problem is to deploy containers within VMs which wastes most of the benefits of containerization. MicroVMs technologies, such as Kata, gVisor, Firecracker, and etcetera are upcoming options for solving the noisy neighbor problem. They can securely

[20] https://landscape.cncf.io/.
[21] https://opensource.com/article/18/4/what-is-cloud-native-computing-foundation-cncf.
[22] Kubernetes is based on Google's internal project called Borg: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/44843.pdf.
[23] Envoy is the de-facto service proxy originally developed at Lyft: https://blog.envoyproxy.io/envoy-graduates-a6f71879852e
[24] https://github.com/cncf/telecom-user-group.

run untrusted pieces of code on a cloud platform. However, these approaches can be an overkill for telco use cases, since most of telco applications are 1st party (their own) or 2nd party (vendors) solutions. These applications can (usually) be deployed as containers using native container orchestration (Kubernetes) features such as: resource limits, and QoS to constrain both compute and networking resources for the deployed containerized workload.

Providing performant network connectivity for CNFs is another problem which is part of CNCF TUG's work items. MULTUS,[25] DANM[26] and NSM[27] projects hosted by CNCF are specifically focused on solving the CNF networking challenges. In section 2.3, possible solutions are explained in detail to understand the current state of Cloud Native networking technologies and provide a gap analysis.

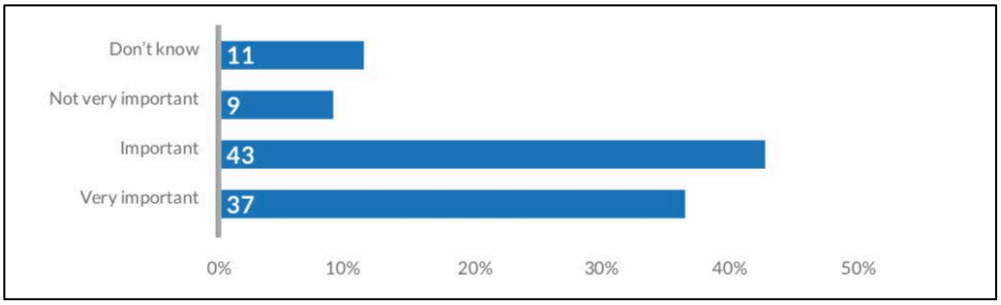


**Figure 2.3. Role of Open Source in the Transformation of Infrastructure to Support 5G.[28]**

The work which has started at CNCF TUG can be critical for laying the path of migration from VNF to CNF and outlining the Cloud Native best practices for CSPs. The journey has just started, and it will take the industry a few iterations of architecture evolution and infrastructure modernization to run true CNF in a service provider's network. It is evident that the Open Source community, in general, and CNCF, in particular, are playing a very critical role this transition as shown in Figure 2.9. Adoption of Open Source has provided telecommunications industries with the opportunities to alleviate some problems by: lowering the cost barrier to bring in technologies for evaluation and rapid prototyping; preventing vendor lock-in; providing agility to innovate at a quicker pace; and drive innovation through collaboration and common Open projects/tools. All of this is helping Service Providers to gain better control of their ecosystem. Open Source projects provide an ideal sounding board for development of standards and protocols. The collaborative activities across open source projects and standards organization for the networking industry provide the opportunity of a quick feedback loop. Signing of Memorandums of Understanding (MoUs) between European Telecommunication Standards Institute (ETSI) and the Linux Foundation is good first step to enable collaboration for less fragmentation, faster deployments and more streamlined innovation for 5G.[29]

[25] https://intel.github.io/multus-cni/.
[26] https://github.com/nokia/danm.
[27] https://networkservicemesh.io/.
[28] https://www.interdigital.com/white_papers/telecomtvs-2018-5g-evolved-and-defined-survey.
[29] http://www.intelligentcio.com/eu/2019/04/26/etsi-and-the-linux-foundation-sign-mou-enabling-industry-standards-and-open-source-collaboration/.

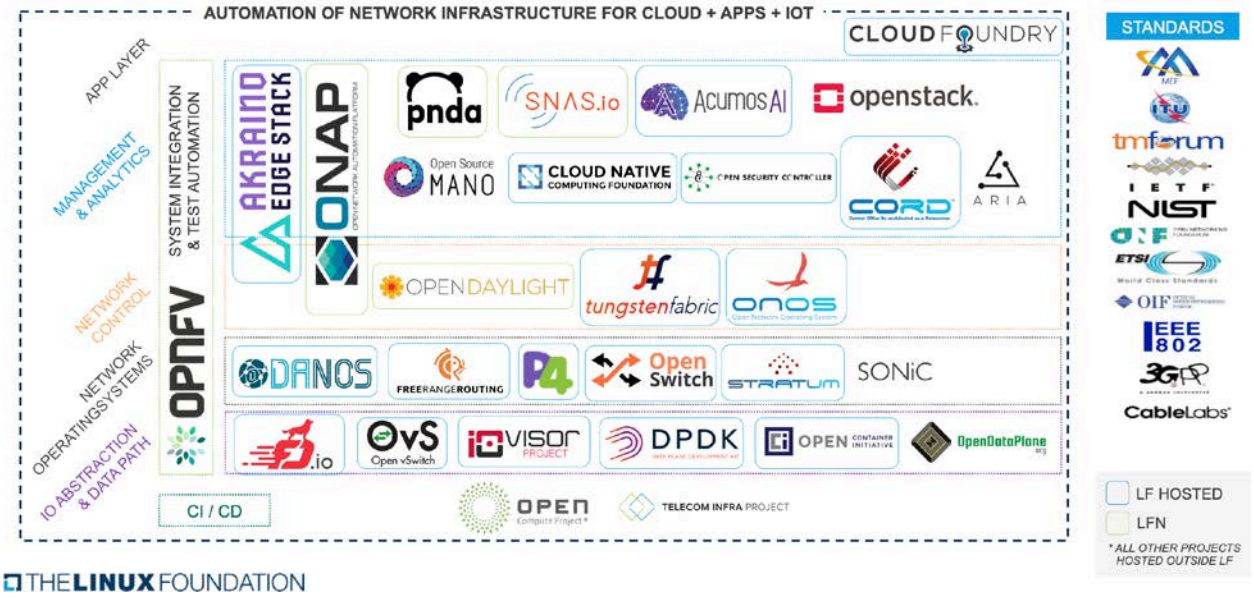


**Figure 2.9. Current Opensource and Standard Landscape for Telecommunications.**[30]

## 2.3 BENEFIT OF ADOPTING CLOUD NATIVE

Adopting Cloud Native application architecture and development methodologies can get complex. Thus, it is important to understand the value proposition of Cloud Native; adoption is strong in the enterprise and IT domain and recently the telecommunications industry is trending toward Cloud Native for 5G. Key benefits of adopting Cloud Native computing architecture and technologies include:[31]

- **Agility:** Microservices- and container-based applications are optimized for changes in application design. Cloud-native applications enable developers to rapidly change components of an application with minimal disruption to the overarching application structure and edifice. This ability to rapidly change an application enables organizations to reliably upgrade and modify applications to accommodate the changing needs of the business

- **Enablement of DevOps:** Cloud-native applications illustrate the realization of DevOps because they automate operational processes such as scaling, monitoring, self-healing, redundancy and high availability. The automation of these operational processes enables development and operations teams to collaborate more efficiently and subsequently ship code faster

- **Resiliency:** Cloud-native applications are architected with explicit attention to resiliency. The loosely coupled quality of microservices architectures, for example, ensures that the degradation of one microservice has minimal impact on other microservices. As a result, development teams can swiftly locate and restore loss of functionality specific to a degraded microservice. Moreover, self-healing processes specific to container orchestration frameworks automate the replacement of unhealthy containers with healthy containers

[30] https://www.fiercewireless.com/wireless/linux-foundation-zeros-harmonizing-open-source-standards
[31] https://www.accenture.com/us-en/blogs/blogs-miha-kralj-cloud-native-architecture-benefits

- **Portability:** Because cloud-native applications are container-based and abstract away dependencies on their external environment, they are more easily transported across different environments than other types of applications. The portability of cloud-native applications renders them less prone to vendor lock-in and subsequently more attractive to customers that are reluctant to deploy their applications to a unitary framework

- **Granular application scalability:** Developers can automate the scalability of container-based applications by means of a container orchestration framework such as Kubernetes. Moreover, orchestration enables developers to scale the container or containers within an application that require scaling instead of scaling the entire infrastructure on which the application runs

- **Efficient resource consumption:** The lightweight quality of containers allows them to be rapidly destroyed and created. In addition, containers can be deployed to a server with greater density than VMs. Containers deliver more efficient resource consumption than VMs and subsequently facilitate accelerated development, improved operational efficiency, and cost savings

- **Abstraction and Cloud Agnostic:** Container and container orchestration technologies provide inherent abstraction as they are infrastructure and technology agnostic. This makes it much easier to adopt hybrid cloud strategy for Cloud Native application architecture, as an application can run on any cloud platform (private or public) as same set of APIs for managing container lifecycle is exposed and can be managed any available Cloud Native tooling

- **Orchestration**: Due to the immutable nature of containers, it is much simpler to manage the lifecycle of containers than VMs. Kubernetes is the de-facto container orchestrator and has mature constructs available like helm charts and operators for the Create, Retrieve, Update, Delete (CRUD) operations for applications deployed and exposed as service endpoints. Moving to containerized microservices provides the ability to orchestrate the containers such that separate lifecycle management processes can be applications to each service, allowing for each service to be versioned and upgraded separately. If the application is implemented with the appropriate level of state separation, this process can allow for fully automated in-service upgrades and rollback of the containers that make up the application.

## 3. ETSI NETWORK FUNCTION VIRTUALIZATION, CLOUD NATIVE NETWORK FUNCTIONS AND 5G

To understand Cloud Native in 5G, perhaps it is worth understanding the context surrounding how Cloud Native arrived. One problem of cellular systems in the past is their slowness to evolve, and consequently, the prevention of the cellular network from addressing (new) markets in a nimble and responsive manner. There are numerous reasons and involve the way that cellular networks have been architected over many years. Historically, cellular networks have been architected as a collection of independent “boxes” having specific functionality with standardized (open) interfaces between the “boxes”. This approach led naturally to having monolithic software architectures for the boxes in question. This architecture has worked well over many years since the reliability and resiliency of the cellular network was well understood and led to well-engineered networks. However, over recent years, a new infrastructure paradigm has emerged, namely cloud-based infrastructure. In this paradigm, the infrastructure, is composed of a collection of general-purpose servers. The net effect of this is that the software designed for this type of (shared) infrastructure had to adopt development methodologies (CI/CD) and software architectures (for example,

microservice) more in keeping with this type of environment. A by-product of the adoption of these methodologies and architectures is that a potentially nimbler cellular network service platform results in being capable of addressing new cellular markets. How the industry, 3GPP and ETSI have arrived at the orchestration and Service Based Architecture for cellular networks is presented in this section.

## 3.1 BUSINESS DRIVERS

Arguably, there are two perspectives on the business drivers for the adoption of Cloud Native in 5G (or any wireless generation for that matter). It can be viewed from the Communications Service Provider (CSP), or, it can be viewed from the network function vendor perspective. In both cases, the need to adopt Cloud Native approaches converge for very different reasons. The actual motivations for adoption by the entities concerned are probably due to a unique set of conditions. Following are some broad generalizations.

From the CSP perspective, the arrival of Clould Native is an interesting path on cost optimization. Mobile networks are predominantly an access and edge network. The costliest portions of any network tend to occur where there are large amounts of aggregation due to numerous physical points of presence. In mobile networks, this translates to the physical numbers of base stations and their necessary aggregation. In parallel with this, there has been intense competition between CSPs in saturated markets to gain and retain subscribers. This competition led to a decline in Average Revenue Per User (ARPU) as witnessed in some markets. Because of this, over many years, there have been focused efforts to bring the cost of access down to levels sustainable by the market subscriber ARPU. This effort has been successful. However, this success has exposed other areas of cost that need to be addressed. Among these are the operational aspects of the network. This motivated many to contrast and compare CSP network operation with network operation in parallel industries (see for example [32] [33] [34] [35] ), namely the hyperscale public cloud operators, where large infrastructure has very lean operations. In this examination, several very appealing facets of a cloud operation were exposed:

- Automated operation and deployment
- Appeal of high service availability based on common (and relatively cheap) infrastructure

Both facets address the (newly) exposed cost reduction needs. Moreover, these facets when matched with new trends in software development (such as CI/CD) also opened the prospect of delivering new and competitive services more quickly to their customer base, leading the potential of time-to-market advantage. Hence, the appeal of Cloud Native architectures.

From a network function vendor perspective, the design and implementation of monolithic network functions (therefore, functions in a box) for the most part had waterfall software development methodology to which they were associated. This implied that the software development process was sequential in nature, for example, requirements gathering, and analysis had to be complete before system design could start, System design had to be complete before implementation could begin, implementation had to be complete before testing could begin, and so on. It could be argued, from a historical perspective, that the reason the waterfall model was applied to monolithic network functions was that they were delivered into a system that

[32] *From Webscale to Telco, the Cloud Native Journey*, 5G-PPP. July 2018.
[33] *View on 5G Architecture*, version 3.0, 5G-PPP. July 2019.
[34] *5G Mobile Network Architecture for diverse services, use cases, and applications in 5G and beyond*, Deliverable D2.1, 5G-Monarch. November 2017.
[35] *5G Mobile Network Architecture, for diverse services, use cases, and applications in 5G and beyond*, Deliverable D2.3 - Final overall Architecture, 5G-Monarch. April 2019.

had high reliability and availability requirements (5-9's) and needed a well thought-through frontend to the software development process similar to waterfall methodology.

However, there are several drawbacks to the waterfall methodology that promoted a move to more agile software development. Among these are:

- Development process was inflexible to requirements change
- Cadence of delivery meant that new features often arrived late or had been superseded by newer requirements or totally different features
- Long development cycles when addressing complex systems

This results in two commercial effects:

- It tied up development resources on features that, for the most part, were never really used
- Led to release cycles that were not competitive in the market

Both detractions are opportunity costs which are detrimental to the effective use of Research and Development (R&D) investment. In short, a high level of resource was being inflexibly consumed for a relatively low return on the engineering investment.

These detractions speak more to the need for an agile software development process rather than per se a Cloud Native proccess. However, when the target platform for the network function also changed, from a bespoke piece of hardware to a generalized server blade in a data center, then the agile process was adopted to include those development components (CI/CD) and independent modular architectures needed for cloud delivery.

Given both perspectives (CSP and the network function vendor), the question arises whether the move to Cloud Native has yet delivered on those business drivers. That discussion is out of scope of this whitepaper.

## 3.2 TECHNICAL DRIVERS

Much of the justification for the technical drivers for the application of Cloud Native to 5G can be attributed to studies promoted by the Next Generation Mobile Network Alliance (NGMN),[36] [37] 5G Infrastructure Public Private Partnership (5G PPP)[38] and the European project 5G-MoNArch.[39] [40] Because of the need to enable new business models, there is an underlying requirement for the network to be operationally flexible enough to permit these new business models to develop. Since these new business models are not (yet) known, then the implication is that that the underlying architecture has to be extensible enough obviating any major architecture upheaval to fulfil this requirement at some future date.

The implication of the flexibility required by 5G business and operational models, combined with the need to place VNF/CNF onto Common Off-The-Shelf (COTS) hardware means that a modular architecture is

[36] *5G White paper v1.0,* NGMN Alliance. Rachid El Hattachi ,Editor. 17th February 2019.
[37] *Service-Based Architecture in 5G v1.0,* NGMN Alliance. Dan Wang Editor. 19th January 2018.
[38] *From Webscale to Telco, the Cloud Native Journey*, 5G-PPP Software Network Working Group. July 2018.
[39] *5G Mobile Network Architecture, Deliverable D2.1*, 5G-MoNArch. November 2017.
[40] *5G Mobile Network Architecture, Deliverable D2.2*, 5G MoNArch. July 2018.

required in which new functionality can be added with relative ease. This also implies a need for a framework approach to the modular architecture, therefore, self-contained services, service registry, authentication and discovery components, and so on.

In order to satisfy this requirement, comparisons were made with existing web-scale infrastructures (their architectures and operations) and studies performed to see if key characteristics would translate for use in future mobile network architectures.[41] [42] The key characteristics that have driven web scale adoption of Cloud Native architectures are given below with additional annotations to indicate where these characteristics breakdown in CSP mobile network environments:[43]

1. **Codebase**: A single codebase properly versioned should be associated with varying application deployment strategies (recreate, rolling-update, blue/green, canary, A/B testing, shadow)

    a. CSP/Vendor Limitation - because of the concerns at 2.a below then this may also carry over to the codebase

2. **Dependencies**: Dependencies (either system or between applications) should be explicitly declared in a dependency manifest and isolated in environments during both development and production stages

    a. CSP/Vendor Limitation - different H/W due to different suppliers and resulting N/W configurations (particular to the vendor) may cause dependency problems

3. **Configuration**: The configuration that varies between deployments should be stored in the deployment's environment separately from the code

4. **Backing services**: Backing services are services the application consumes over the network (databases, messaging queues, caches and etcetera and are considered as loosely coupled resources attached to the application

5. **Build, release, run**: Code is transformed to a deployment in three separate stages: **build**, during which a codebase is transformed to an executable with its appropriate dependencies; **release**, which combines the build with the environment configuration; and the **runtime** stage which launches the application's processes. Such stages should be air- tight

    a. CSP Limitation - build, release, run is not part of any telco standards (for example, ETSI, 3GPP, and etcetera)

6. **Processes**: The application is executed as stateless, share-nothing processes. Persistence is offloaded in a stateful backing service (4). Cached data is considered volatile and not stateful. This is a factor that can break scalability of a cloud-native application

7. **Port binding**: Services are exposed to other services through specified ports. This aspect gives rise to the service discovery role of an application orchestrator mechanism

[41] 5G-PPP Software Network Working Group, From Webscale to Telco, the Cloud Native Journey, July 2018.
[42] NGMN Alliance, Service-Based Architecture in 5G v1.0, 19th Jan 2018. Dan Wang Editor.
[43] These 12 characteristics or factors come from https://12factor.net/ and the list is quoted mostly verbatim from 5G-PPP Software Network Working Group, From Webscale to Telco, the Cloud Native Journey, July 2018.

8. **Concurrency**: To scale out the application concurrency follows the process model, therefore, by adding more of the same processes to enable concurrency. This scalability enabler is complemented by the shared-nothing, stateless property of the application

    a. CSP/Vendor Limitation - the problem is that this is not the case today. Not all NFs scale the same way. Moreover, there is an aspect of statelessness that may not be adopted universally by all VNFs/CNFs. It may take more than one generation of the VNF/CNF in order to comply with this factor

9. **Disposability**: Processes should require minimum start-up time (to quickly fire up new processes and support elasticity and rapid deployment). Graceful shutdown is also a goal; a process should accommodate standing requests or computation before responding to a SIGTERM signal

    a. CSP Limitation - the start-up times need to be small, and the tear downs need to be graceful. Currently this varies widely between VNFs. This is a concern because it reflects directly on the necessary 5-9s reliability requirement of CSPs

10. **Development/production parity**: A cloud-native application's development environment should mimic the production environment as much as possible. More specifically, the time, personnel and toolset gaps between the deployment areas should be as small as possible to increase the effectiveness of Continuous Delivery (Deployment) of the application. This is facilitated by the use of containerized environments and declarative provisioning and configuration tools aiming exactly at shortening the time and toolset gaps

    a. CSP/Vendor Limitation - the problem is again due to concern 2a, and this may carry over to this factor. In short it may make it difficult to mimic the deployment environment because of the uniqueness of the CSP that has historically arisen

11. **Logs**: Logging should be treated as a stream of events available for monitoring all separate services of the application. The streams should be aggregated together to better understand the whole picture of the state of the application. However, the application is not involved with actual log analysis; this is offloaded to separate external tools

    a. CSP Limitation - Current logs tend to be particular to a given vendor, therefore, not heterogenous, and also the update rates vary from vendor to vendor

12. **Admin processes**: Run admin/management tasks as one-off processes that are preferably declaratively stated against a release, using the same codebase and configuration. These tasks are to be shipped with application codes under their environment

    a. Vendor Limitation –in some way all CSPs are unique in this regard

The conclusion of these comparisons is that although not all of the characteristics satisfied the demands of current mobile network architectures, the potential benefit of adoption of Cloud Native outweighed the limitations.

In summary, the need to satisfy the requirements on flexibility are addressed through the concept of services as atomized capabilities with the desired characteristics of high-cohesion, loose coupling and independent management from other services and lend themselves to a microservices architecture. Moreover, given the necessary loose coupling between services, the service bus naturally falls out of this requirement as does the service-based interface.

## 3.3 3GPP'S TAKE ON CLOUD NATIVE

The requirement on flexibility can be satisfied by having a modular, loosely coupled architecture. This has manifested itself in 3GPP as the Service Based Architecture (SBA).

The actual justification for SBA is a compromise between the size of a mobile service function and the suitability of a variety of other service-based of architectural styles,[44] for example, microservices (or µservice),[45] Service-Oriented Architecture (SOA),[46] and service-based hybrids types.[47] This compromise is depicted in Figure 3.1.

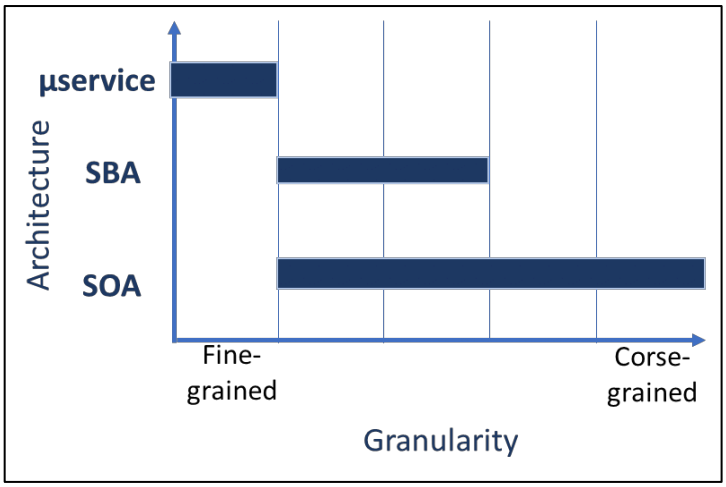


**Figure 3.1. 5G-PPP Software Network Working Group view of SBA.[48]**

Given this compromise, the requirements on the SBA can be stated as follows:

- Updating Production Network
    - Services operate with finer granularity than in legacy networks and are loosely coupled with each other allowing individual services to be upgraded with minimal impact to other services. A service, in this instance, refers to an atomized 5G network capability, for example policy control, or access management, and etcetera
    - This provides many operational benefits such as shrinking testing and integration timescales (moving towards continuous integration) which reduces the time to market for installing bug fixes and rolling out new network features and operator applications
- Extensibility:

[44] Neal Ford, Mark Richards, 'Service-Based Architectures: Structure, Engineering Practices, and Migration', O'Reilly Media, July 2015
[45] https://en.wikipedia.org/wiki/Microservices
[46] https://en.wikipedia.org/wiki/Service-oriented_architecture
[47] Neal Ford, 'Comparing Service based Architectures', http://nealford.com/downloads/Comparing_Service-based_Architectures_by_Neal_Ford.pdf
[48] 5G-PPP Software Network Working Group, From Webscale to Telco, the Cloud Native Journey, July 2018.

- Each service can interact directly with other services with light-weighted service-based interface
- Comparing to the legacy hop-by-hop model, the service-based interface can be easily extended without introducing new reference points and corresponding message flows

- Modularity & Reusability:
  - The network is composed of modularized services, which reflects the network capabilities, and provides support to key 5G features such as network slicing
  - A service can be easily invoked by other services (with appropriate authorization), enabling each service to be reused as much as possible

- Openness:
  - Together with some management and control functions (therefore, authentication, authorization, accounting), the information about a 5G network can be easily exposed to external users such as 3rd-parties (for example, enterprise) through a specific service without complicated protocol conversion

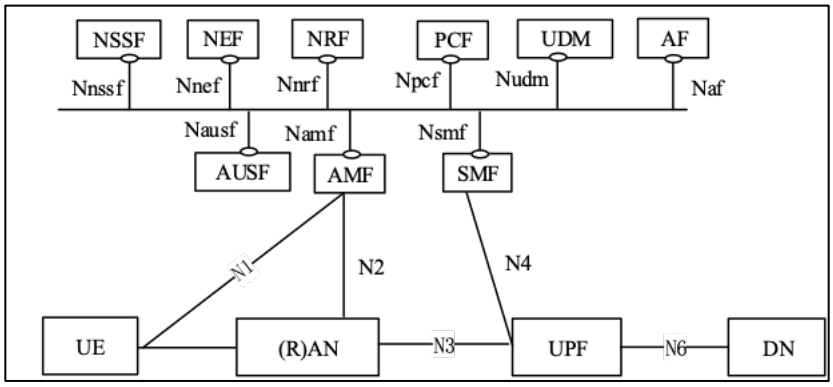


**Figure 3.2. 3GPP TS 23.501 5G System Architecture showing the SBA.**

For 5G, these requirements translate architecturally into the 3GPP diagram of Figure 3.2, in which the service modularity breaks down into the following self-contained functional components communicating via standardised lightweight (RESTful) interfaces:[49]

- AF - Application Function(s). These modules interact with the 5G Core in order to provide services to applications, such as those that influence traffic routing for an application, or access for an application to a given Network Exposure Function, or an applications' need to interact with 5G Core policy functions
- AMF - Access and Mobility Management Function. This module supports all aspects of a device's access and mobility management, among which are the management of: registration, connection, reachability, mobility, access authentication and authorization, and etcetera

[49] For those who are interested in greater detail on some or all of these functions, refer 3GPP TS 23.501 and 3GPP TS 38.413 for the protocol details.

- AUSF - Authentication Server Function. This module supports authentication for access onto the 5G network, both for 3GPP access and untrusted non-3GPP (for example, WiFi) access
- NSSF - Network Slice Selection Function. This module is responsible for selecting the 5G network slice instance serving User Equipment (UE) and which AMF, or list of AMFs, can be used by a device (UE)
- NEF - Network Exposure Function. This module supports the exposure of NF capabilities and events
- NRF - Network Resource Function. This module supports service discovery and maintains the NF profile of available NF instances and their supported services
- PCF - Policy Control Function. This module supports a unified policy framework that governs the 5G network behaviour, provides enforceable policy rules to the control pane functions, and access relevant policy subscription information from the UDR
- SMF - Session Management Function. This module supports all aspects of session management of a flows between the access node and the UPF, such as the selection and control of the UPF, the traffic steering of the flow, the collection of charging data, and so on. It is also responsible for IP functionality such as IP address allocation, DHCP, ARP, and etcetera
- UDM - Unified Data Manager. This module supports the following functionality: the generation of 3GPP authentication credentials, user identification handling, support of de-concealment of privacy-protected subscription identifier, access authorisation based on subscription data, a devices (UE) current NF registration management (therefore, its current AMF and serving SMF for a given device PDU session). It also provides support for service/session continuity, MT SMS delivery, Lawful intercept, Subscription management and SMS management
- UPF - User Plane Function. This is the only modular function that does not reside in the control plane and is not currently part of SBA. This function acts as an anchor point for mobility in the 5G core, and performs a myriad of actions on a data flow, such as packet routing and forwarding, packet inspection, policy enforcement, Quality of Service (QoS) handling, legal intercept, and etcetera

SBA is key to the 5G Core, and in effect lays out the base modularity for which Cloud Native implementations are intended.

### 3.3.1 SERVICE FRAMEWORK

The SBA relies on a service framework. At a high-level, there are three main components to the service framework:

- A registration component

  Implemented based on a service registry which is a database of available (currently active) services and their reachability. Services are registered in the registry once the service are activated, and of course, on deregistration are deactivated. A service consumer can query this database (the registry) to find available services and how to access them

- An authorisation component

  The authorisation of a service is required in order to control if a service can be called by other services. This component is really necessary when (alien) third party services are using native services of the network

- A discovery component

  A service consumer queries for a specific service in the service registry. The service registry responds to several available services and their addresses to the consumer. Load-balancing mechanisms can be used to assist the appropriate selection of available services

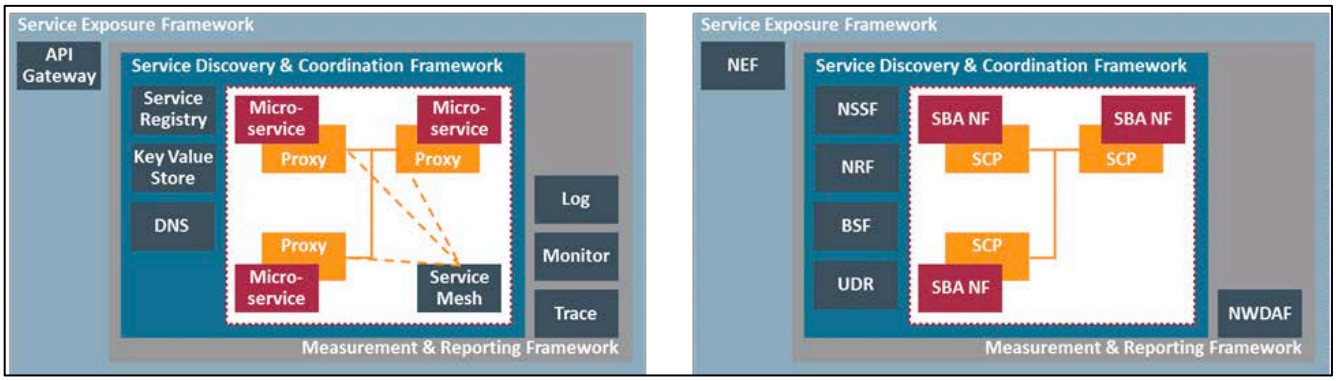


**Figure 3.3. CNCF Service (left) versus SBA Services (right).**[50] [38]

These components combine to provide a service exposure framework for the 5G SBA modules, which (as can be expected) is close to that employed by web-scale Cloud Native deployments. Figure 3.3 shows the comparative service exposure framework for web-scale and 5G Cloud Native deployments.

While all of these approaches are very cloud operational in nature, in the case of 5G they have a very specific connotation to them, meaning that it is more than likely that only other (roaming) partner's CSP networks can actually avail themselves of these services. It is unlikely that just any 3rd party can access these services in a direct manner, as it is common for any paying member of the public accessing AWS services today.

### 3.3.2 THE SERVICE-BASED INTERFACE (SBI)

Critical to the operation of SBA is communication between modules, preferably via RESTful interfaces, with the necessary characteristics that such communication interfaces should possess including to:

- Use standard protocols and data models for multi-vendor interworking
- Be light weight for efficient communication, for example, high concurrency, low latency, and etcetera
- Easily exposable internally and externally for invocation or reuse
- Be widely used with rich tools for software development

Additionally, for the 5G SBA architecture, there are some properties of the protocol which should also be considered such as:

- **Extensibility:** protocol needs to be easy to extend, not only in the 3GPP context, but also outside the standards. The protocol should support the service to be deployed and instantiated with minimal impact on the system

[50] https://www.oracle.com/a/ocom/docs/industries/communications/cloud-native-journey-telecomm-wp.pdf

- **High efficiency:** support to reduce resource consumption for the protocol analysis and adopt efficient serialization methods for the protocol
- **Reliability:** support of reliable communication between services. The protocol should support secure communication, in particular, service authentication, authorization, and possibly encryption, in particular for inter-operator communication
- **Simplicity:** the number of protocols to be supported in a network should be minimized to simplify the implementation of the system. The selected protocol should be able to support intra- and inter-operator interfaces
- **Functionality requirement:** The protocol should enable stateless operation. While for particular services, the User Equipment (UE) session context should be considered for the service selection

For Rel-15 (and onward), 3GPP considered the necessary characteristics and properties for its service-based interfaces and concluded on the need for: HTTP/2 for application layer, TCP for transport, JSON for serialization, and apply the RESTful framework for the API design style whenever possible, and use OpenAPI[51] for Interface Description Language (IDL).

## 3.4 3GPP RELEASES

Figures 3.4 and 3.5 show the 3GPP timelines for 3GPP Releases 15 and 16. The initial deployments of 5G with SBA and Cloud Native approaches are already being considered for commercial deployment. Moreover, as Rel-17 focuses on enterprise and vertical use cases, the architecture to support new business models and service innovation needs to be mature. A pragmatic window of between three to four years is open in which to develop the required Cloud Native maturity as applications to the 5G core network.

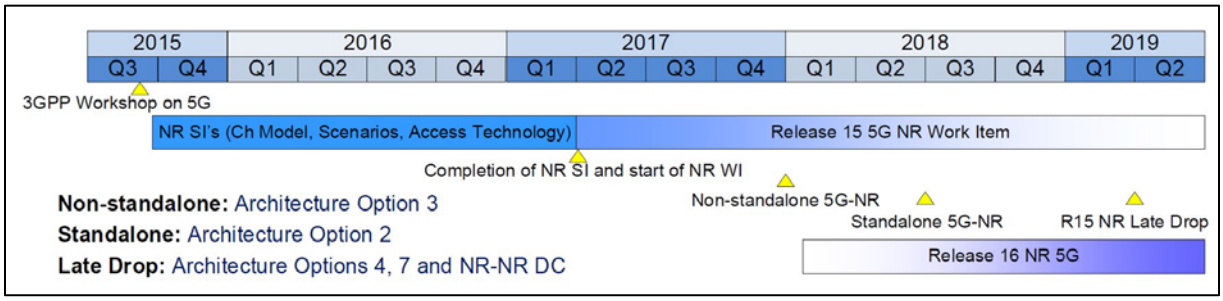


**Figure 3.4. 3GPP Release 15 Timetable.**

[51] https://github.com/OAI/OpenAPI-Specification.

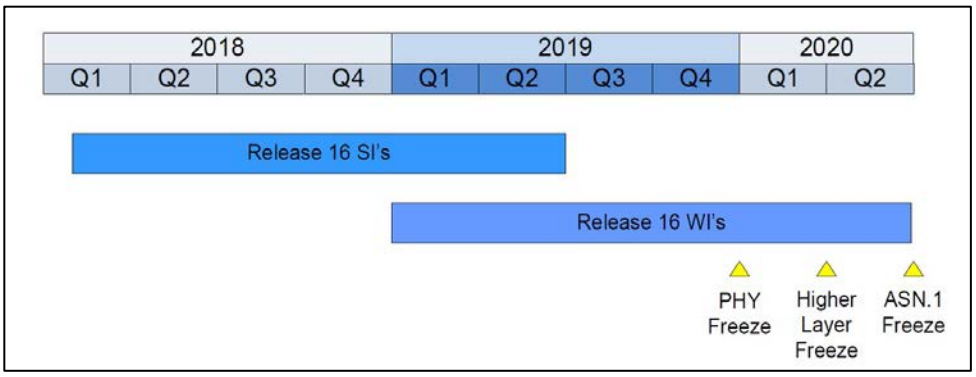


**Figure 3.5. 3GPP Release 16 Timetable.**

## 3.5 PROPOSED ETSI NFV REFERENCE ARCHITECTURE AUGMENTATION FOR CLOUD NATIVE

As a standard specification, ETSI focuses on high-level architecture, development guidelines, and interoperability enabled by open interfaces. Most of the specifications in ETSI Management and Network Orchestration Group Specification (MANO GS) continue serving the purpose when applying to the Cloud Native NFV. Nevertheless, augmentation is needed in some areas because of the differences between the VM based and Cloud Native solutions.[52]

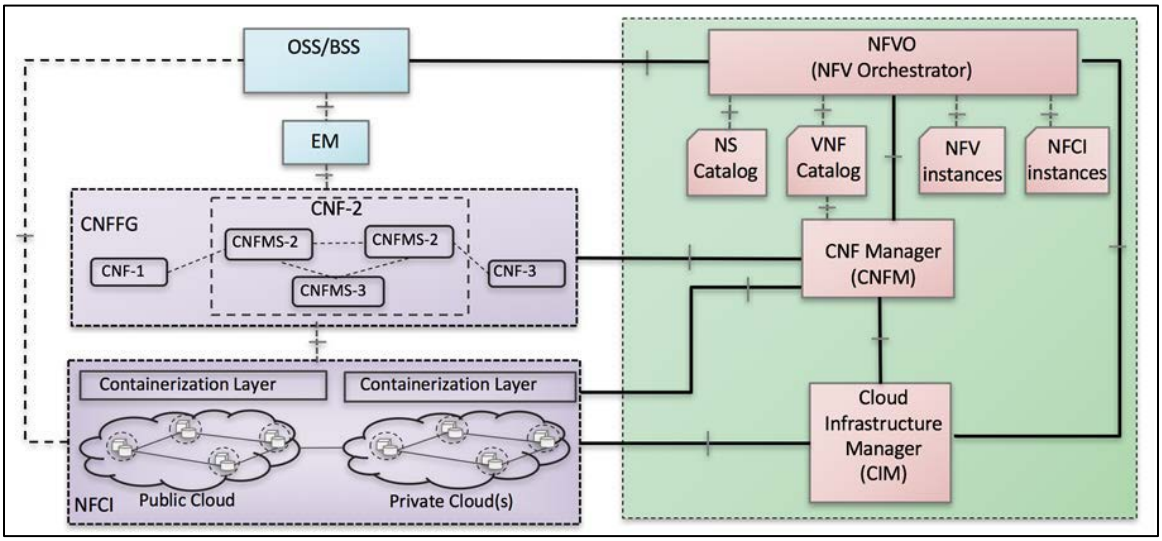


**Figure 3.6. ETSI MANO Cloud Network Function Orchestration Architecture.[53]**

Cloud Network Function MicroService (CNFMS)

- CNFMSs are the microservice containers from which a Cloud Network Function is composed

[52] © 2017 SCTE-ISBE and NCTA.
[53] © 2017 SCTE-ISBE and NCTA.

Cloud Network Function (CNF)

- CNFs are the network functions deployed in the cloud as microservices, usually in container format

Network Function Cloud Infrastructure (NFCI)

- NFCI provides the underline physical infrastructure for the network functions. This includes the hardware equipment for the computer, networking, storage, as well as the containerization layer on top of the hardware platform. CNFs are deployed on top of the NFCI

Cloud Infrastructure Manager (CIM)

- CIM is responsible for controlling and managing the NFCI compute, storage and network resources, as well as scheduling the microservice containers in the cloud. It manages the lifecycle of the containers in the cloud
- CIM also collects performance measurements in the infrastructure including container level and makes the data available for other functional blocks for monitoring purposes
- Other responsibilities of CIM include virtual networking control and management, as well as the southbound integration with various network controllers to achieve the physical network control and management capabilities
- Examples of the CIMs available in the market are Kubernetes, AWS ECS, and Docker Swarm

Cloud Network Function Manager (CNFM)

- CNFM focuses on the life cycle management of individual CNF instances. In the Cloud Native architecture, a CNF is usually composed of a set of containers that implement a network function. A CNF manager takes the responsibility of the management of the multiple container instances of the same network function. To control the lifecycle of the CNFs, CNFM works closely with CIM, which manages the lifecycle of the individual container of the CNF
- CNFM also serves as an overall coordination and adaptation role for configuration and event reporting between the CIM and the EM systems of traditional operator architectures

NFV Orchestrator (NFVO)

- The NFVO continues serving the responsibility of on-boarding a new Network Service (NS) composed of multiple CNFs, CNF forwarding graph, Virtual Links, and, as an option, Physical Network Functions (PNFs). The orchestrator also controls the life cycle of the Network Service including instantiation, scale-in/out or up/down, performance measurements, event correlation and termination. Further key operational functions are global resource management, validation and authorization of NFCI resource request, as well as policy management of Network Service instances

Cloud Network Function Forwarding Graph (CNFFG)

- The CNFFG contains a list of CNFs and the virtual links among the CNFs and the physical endpoints

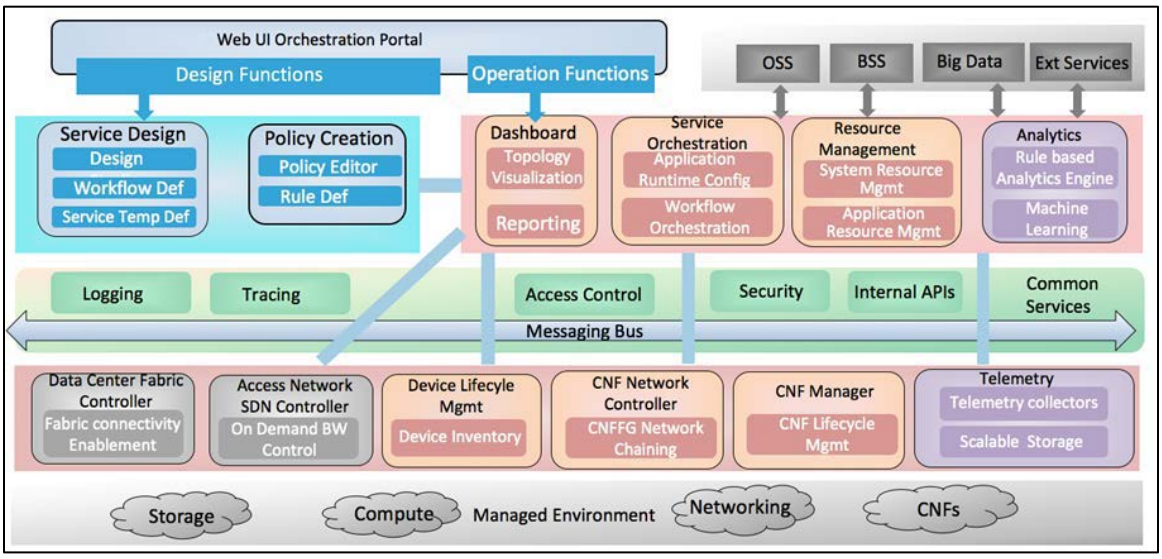


**Figure 3.7. A pragmatic NFV Software Architecture in the Cloud Native Environment.**[54]

ETSI NFV architecture for Cloud Native environment contains the microservices for the NFV Management and Orchestration in both the design phase and the Runtime execution phase. The design phase generates the network services model to be deployed onto the target private or public cloud(s). The Runtime execution phase contains the microservices for the network service deployment, orchestration, lifecycle management, network control, monitoring, and analytics. Besides the functional services, there are a set of common infrastructure services for all the microservices containers in the Cloud Native architecture environment.

## 4. IMPLEMENTATION AND MIGRATION STRATEGIES FOR 5G CORE

It is critical for operators and vendors alike to devise effective strategies for migration from 4G Core to 5GC and to understand how Cloud Native architecture is of influence. The crux of the new architecture is the separation of the control plane and user plane. This is based on SDN principle of centralized control and distributed processing and is part of what makes the 5G core a "cloud-native" design.[55] A necessary step in the migration path to 5GC is the implementation of Control- and User-Plane Separation (CUPS) introduced in Rel-14. CUPS implementation of the EPC core can provide the scale to meet increasing traffic demands and flexibility of edge deployment to serve low latency applications. Figure 4.1 represents the broad conceptual similarity in the architecture of 4G and 5G core.

[54] © 2017 SCTE-ISBE and NCTA
[55] https://img.lightreading.com/downloads/Service-Based-Architecture-for-5G-Core-Networks.pdf

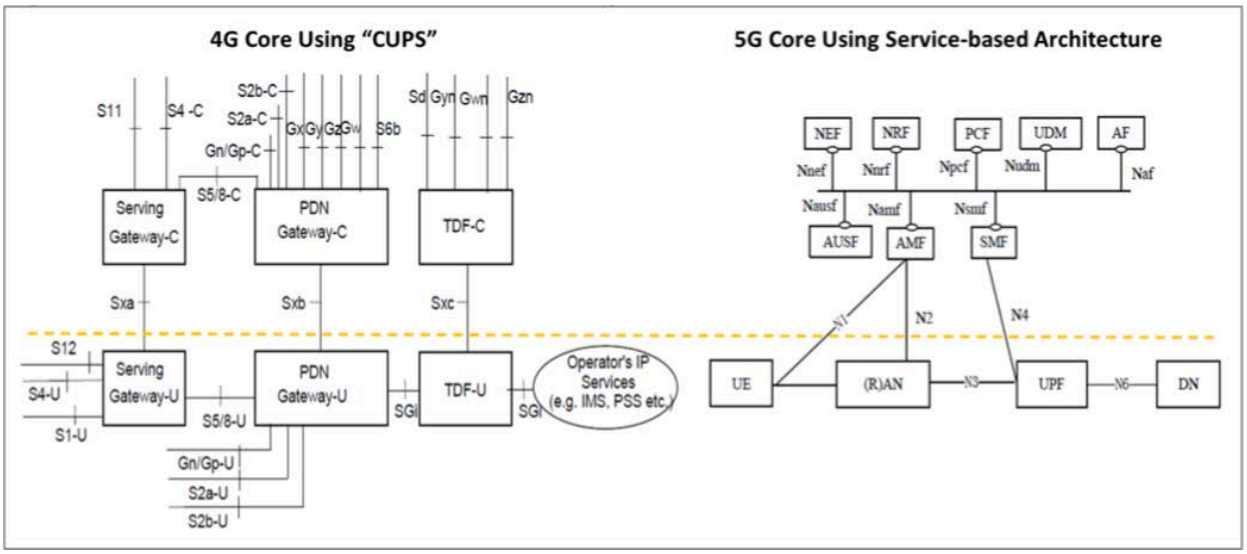


**Figure 4.1. 4G and 5G Core Architecture Comparison.** [56]

5G is unique compared to other 3GPP evolutions as it allows for integration of different elements (access and core) from different generations. This results in the option of having two different types of configuration in the network, namely:

- Standalone configuration (SA) – Only one radio access technology to be used with three 3GPP defined variations: Option 1, Option 2 and Option 5
- Non-standalone configuration (NSA) – Allows for the combination of multiple radio access technologies (LTE and 5G NR) with three other 3GPP defined variations: Option 3, Option 4 and Option 7

Figure 4.1 gives the comparison between the most popular 3GPP deployment options. The NSA options allow legacy operators to make the most of the current LTE core investment, with the provisions of making CUPS based enhancement of EPC to accommodate large NR capacity, and the adoption of 5GC core at a later date. Both Option 7 (NSA) and Option 2 (SA) require a 5GC in the deployment configuration. Moreover, to service all the important 5G use cases, requires 5GC capabilities such as: flow based QoS frameworks, flexible network slicing features, and access agnosticism.

[56] https://img.lightreading.com/downloads/Service-Based-Architecture-for-5G-Core-Networks.pdf

| | NSA | | SA (Op. 2) |
|---|---|---|---|
| | Op. 3 | Op. 7 | |
| **3GPP 5G specification** | Rel-15 ('17.12) – 1st prioritised, | Rel-15 ('18.6) | Rel-15 ('18.6) |
| **5G spectrum** | Sub-6GHz and mmWave bands are feasible | Sub-6GHz and mmWave bands are feasible | Sub-6GHz band is desirable |
| **CN** | EPC, | 5GC | 5GC |
| **CN interworking** | Not required | Not required | Required (with or without N26 interface between 5GC and EPC) |
| **Network slicing and 5G QoS** | Not supported | Supported | Supported |
| **UE impact (for 5G/LTE dual-mode)** | EPC-NAS | 5GC/EPC-NAS | 5GC/EPC-NAS |
| **Leverage of LTE** | Full | Full | Partial (Reattach) |
| **LTE upgrade** | Required (eNB and EPC) | Required (ng-eNB and 5GC) | None or minor |
| **RAN interworking** | EN-DC | NGEN-DC | NR-DC (Intra-RAT) |
| **Inter-RAT data session continuity** | MR-DC and intra-system handover | MR-DC and intra-system handover | Inter-system handover (N26) |
| **Forward compatibility with SA or Release-16 onwards** | Low | Mid | High |

**Figure 4.2. Technical Comparison between 5G NSA and SA Options.**[57]

It is evident that the development of 5GC has to be "Cloud Native" in order to adhere to the 3GPP defined standards from Rel-16 onwards (Open API, Control-User plane separation, SBI, Harmonized protocol based on HTTP2/QUIC, and etcetera). Furthermore, it has to be a Cloud Native-only solution as the deployment infrastructure for 5GC will be in its virtualized form only.

The Cloud-native Network Functions must be designed as loosely coupled components based on microservices, deployed on cloud infrastructure as workloads that can be scheduled on demand by an orchestrator, and that can scale in/out on demand with much greater granularity and velocity.

## 4.1 MOVING FROM VIRTUAL NETWORK FUNCTION TO CLOUD NATIVE NETWORK FUNCTION

In the initial phase of NFV adoption, the network function migration to a virtual platform used a "lift and shift" approach, resulting in disappointment for operators deploying NFV 1.0. The touted benefits of moving to the virtualized cloud platform (multi-tenancy, scalability and flexibility) could not be realized. Furthermore, this initial approach was not as cost-effective when considering additional tooling, integration and operation effort required to make it operational. Moreover, telco "way of working" led to monolithic VNF designs yielding in static and slow-to-respond systems. However, the exercise of adoption for NFV has resulted in some important lessons learned: network functions should be "abstracted" to be technology-agnostic; the network should be decoupled from the application as much as possible; and applications need to be "stateless" rather than "stateful. All this makes the case for building a "true Cloud Native" network function deployable on network cloud infrastructure.

[57] ..https://www.gsma.com/futurenetworks/wp-content/uploads/2018/04/Road-to-5G-Introduction-and-Migration_FINAL.pdf

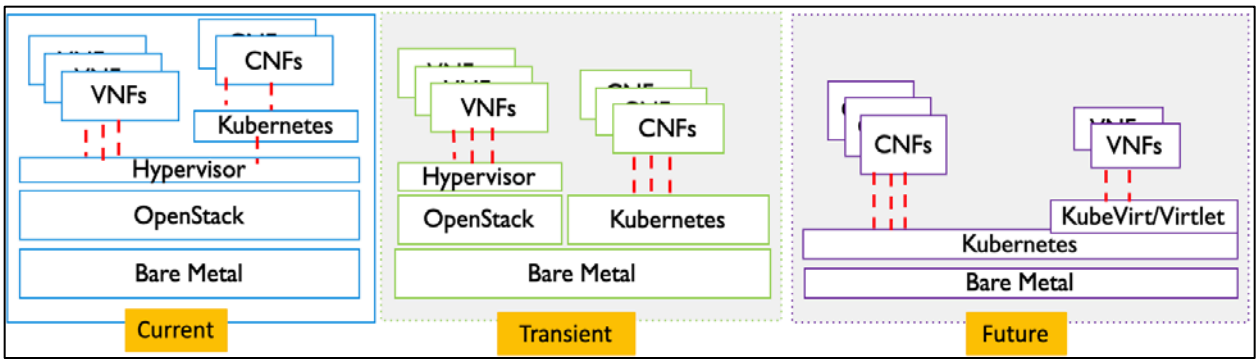


**Figure 4.3. Evolution to Cloud Native Ecosystem.**

In NFV 1.0, the network functions were deployed as VMs in conjunction with OpenStack. Operators who already started their NFV journey have deployed VNFs running on an OpenStack platform and probably made significant investment in building tooling capabilities for managing and operating the NFV platform. Such operators can be tempted to run Kubernetes in VMs as a quick extension to their existing OpenStack ecosystem to support CNFs. This approach for supporting CNFs within VMs can provide fully featured multi-tenancy and security. However, it comes at a huge cost of lost workload performance, complex networking, much lesser container density and additional workflow for managing VMs running Kubernetes (unable to completely utilize Cloud Native lifecycle practices). This forces the industry to gravitate towards building co-existing OpenStack and Kubernetes platforms running VNFs and CNFs respectively in an independent fashion. This co-existence model provides a better platform for heterogenous workload with an easy support framework. However, horizontal legacy integration between CNF and VNF is challenging and lacks a unified resource view, leading to operational difficulties. Because of these limitations, the potential end state in the evolution of the Cloud Native deployment platform is Kubernetes replacing OpenStack as a single stack solution to manage heterogeneous workload. There are definite advantages to using Kubernetes stack as represented in CNF testbed results shown in Figure 4.4.[58]

[58] https://github.com/cncf/cnf-testbed/blob/master/comparison/doc/cncf-cnfs-results-summary.md

| | OpenStack | Kubernetes |
|---|---|---|
| **Infra deploy time** | ~65 minutes | 16 minutes* |
| **NF deploy time** | 3 minutes, 39 seconds | < 30 seconds |
| **Idle state RAM** | 17.8% | 5.7% |
| **Idle state CPU** | 7.2% | 0.1% |
| **Runtime NF RAM** | 17.9% | 10.7% |
| **Runtime NF CPU** | 28.8% | 39.1% |
| **Snake case PPS** | 3.97 million PPS | 4.93 million PPS |
| **Snake case latency** | ~2.1 milliseconds | ~2.1 milliseconds |
| **Pipeline case PPS** | N/A | 7.04 million PPS |

**Figure 4.4. Performance Comparison of VNF on OpenStack and CNF on Kubernetes.[59]**

### 4.1.1 CHARACTERISTICES OF AN IDEAL CLOUD NATIVE NETWORK FUNCTION

The telecommunications standards from 3GPP are in flux and gravitating towards Cloud Native architecture. As discussed in section 3, standards are not finalized for all of the 5GC CNFs to fully adhere to Cloud Native application architecture. For example, Rel-16 only prescribes control plane network functions to be of Cloud Native design.

Current CNFs being developed can include a number of sub-optimal patterns such as:[60]

**Continued reliance on proprietary interfaces:** The network functions, even those defined in Rel-16, will have proprietary interfaces based on 3GPP specific protocols which leads to design pattern not completely adhering Cloud Native architecture

**Stateful CNFs:** In the near term, it may be hard for some of core network functions, like packet gateways, to be decoupled in nature. Stringent carrier grade service level agreements also play role in applications being developed as stateful ones. This is because it is hard to deliver fast, low latency storage in distributed architectures

[59] https://docs.google.com/presentation/d/1nsPINvxQwZZR_7E4mAzr-50eFCBhbCHsmik6DI_yFA0/edit#slide=id.g5036f143e9_3_672

[60] https://docs.google.com/presentation/d/1iAgzRp5eFv7LWmpR2u1Wy0LdhvB85SkKJBxXFSNH8XE/edit#slide=id.g51566346dd_1_7.

**No support for horizontal scalability:** Horizontal scaling tends to be more complex than vertical scaling as it is fundamentally dependent on the application architecture. This can be a challenge for transient CNFs as it is a break from a Cloud Native application architecture

**Proprietary installer:** Installation and upgrades for CNF are being managed by proprietary scripts. These cannot be built into automated pipelines with configuration stored at multiple locations. This is due to the application not supporting configuration management using environment variables

Operators and their vendors should work towards defining the best practices around CNFs specifically for 5GC, which should include following features:[61]

- **Compatible:** CNFs should work with any Certified Kubernetes product and any Container Network Interface (CNI)-compatible network that meet their functionality requirements. This capability will allow for the CNFs to be easily deployed in a hybrid cloud due to infrastructure abstraction provided by the container orchestrator. This would also enable cloud bursting capabilities if needed
- **Stateless:** Handling of state separate from business logic of the CNF
- **Security:** It is critical from a security perspective that all CNF-related workload containers should be able run with unprivileged (non-root) access. This prevents container breakout
- **Scaling:** It should support horizontal scaling (across multiple machines) and vertical scaling (between sizes of machines)
- **Configuration and Lifecycle:** Cloud Native management solutions like ConfigMaps,[62] Operators,[63] or Helm[64] are required to model, deploy and manage CNF workloads
- **Observability:** Metrics, logs and traces should be a pillar of observability for deployed CNFs. Observability should comprise:
    - **Monitoring:** All performance metrics previously available via a proprietary interface should be shared through an OpenMetrics[65] interface that Prometheus and other monitoring tools can use
    - **Tracing:** Support OpenTelemetry-compatible tracing
    - **Logging:** Support Fluentd-compatible logging
- **Hardware support:** Built-in features of Kubernetes, like Extended Resource [66] and Device Plugins,[67] should be utilized to provide extensibility for supporting high-performance hardware such as GPU, FPGA, NIC and InfiniBand in a unified manner for CNFs

### 4.1.2 CONTAINER AS A SERVICE

It can be gathered from the discussion regarding the best practices and features for CNFs in the previous section, that a lot of responsibility falls on the platform hosting them. This is where Container-as-a-Service (CaaS) comes into the picture. CaaS can be considered as the latest cloud service model, somewhere between Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS). CaaS is available as a complete package for supporting container infrastructure. This allows for the management and deployment

[61] https://docs.google.com/presentation/d/1iAgzRp5eFv7LWmpR2u1Wy0LdhvB85SkKJBxXFSNH8XE/edit#slide=id.g51566346dd_1_13
[62] https://cloud.google.com/kubernetes-engine/docs/concepts/configmap
[63] https://kubernetes.io/docs/concepts/extend-kubernetes/operator/
[64] https://helm.sh/
[65] https://openmetrics.io
[66] https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/#extended-resources
[67] https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/

of applications and clusters through containerization techniques hosted on any cloud platform(s) making up the Cloud Native infrastructure.

The core of the CaaS is the container orchestrator solution. This provides all the functionalities and features for supporting complex container deployments in a production environment with support for self-healing (auto-starting, rescheduling and replication). Container orchestrator should be ideally hosted on an operating system optimized for running container workload with hardened security. RancherOS, CoreOS, SUSE MicroOS and RedHat Atomic are examples.

CaaS fits the NFVI layer of the reference ETSI NFV architecture for supporting containerized workloads. The telecommunications industry is gravitating towards Kubernetes as their de-facto container orchestrator solution for CaaS. All the major telco vendors have come up with CaaS solutions of their own. RedHat's OpenShift, Ericsson's Cloud Container Distribution[68] and Cisco's Cloud Container Platform[69] are examples. Their CaaS solutions support CNF workloads with telco-grade capabilities and have Kubernetes as the container orchestrator for each of their solutions.

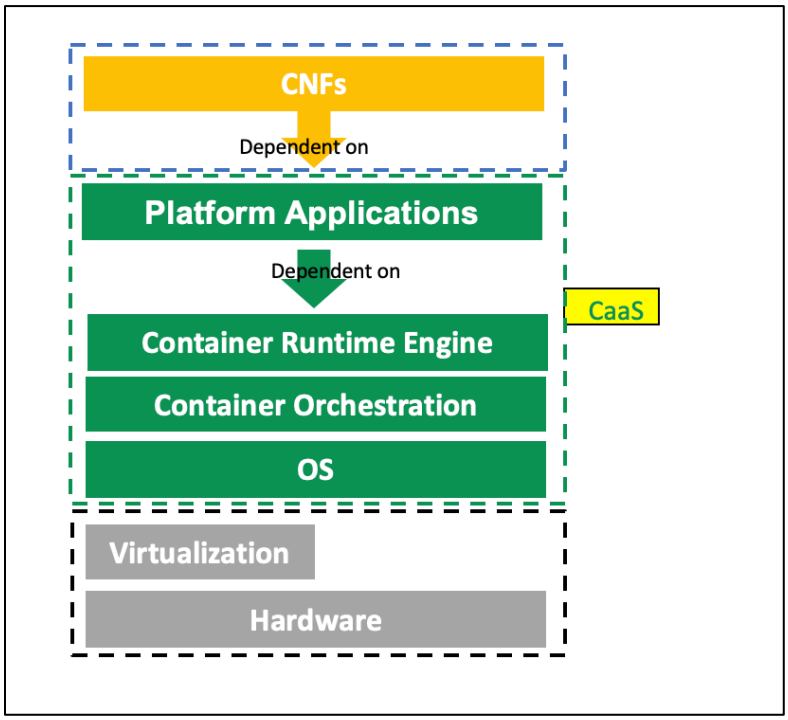


**Figure 4.5. Cloud Native Telco Stack.**

The CaaS platform provides the following advantages:

- Easier to support application's lifecycle through providers' API or web User Interface (UI
- Greater degree of agility due to standardized functionality for on-premise or public clouds
- Bundled as a turnkey solution for easy deployment and cluster management

[68] https://www.ericsson.com/en/portfolio/digital-services/cloud-infrastructure/cloud-container-distribution
[69] https://www.cisco.com/c/en/us/products/cloud-systems-management/container-platform/index.html

- Offers all the power and leverage of standard, Open Source container technologies

Along with the container orchestrator, Telco-grade CaaS bundles the following platform applications (Figure 4.5):

- an image catalog (also called the registry)
- cluster management software, load balancer
- pre-configured service mesh infrastructure layer
- pre-integrated CNI plugins required for data plane acceleration
- storage and security solutions.

The CaaS installer creates production-ready clusters configured for high-availability. Platform applications are also responsible for providing observability and integration with DevOps pipelines. All of this makes CaaS an uber-critical component for building a Cloud Native telco stack for hosting CNFs. Ownership of the CaaS platform by the operator will make management and operation of 5G Cloud platform infrastructure easier, as it will allow for multi-vendor integration of CNFs on the same platform, prevent vendor lock-in and support vertical integration.

## 4.2 CURRENT REALITY AND PROBLEMS

While the previous section highlights the current best practices of Cloud Native applications to CNF (and for that matter VNF), it does not articulate the immaturity that this technology has with respect to its application in cellular networks. Figures 4.6 and 4.7 depict the problems currently faced with Cloud Native applications to network functions (let alone 5G).

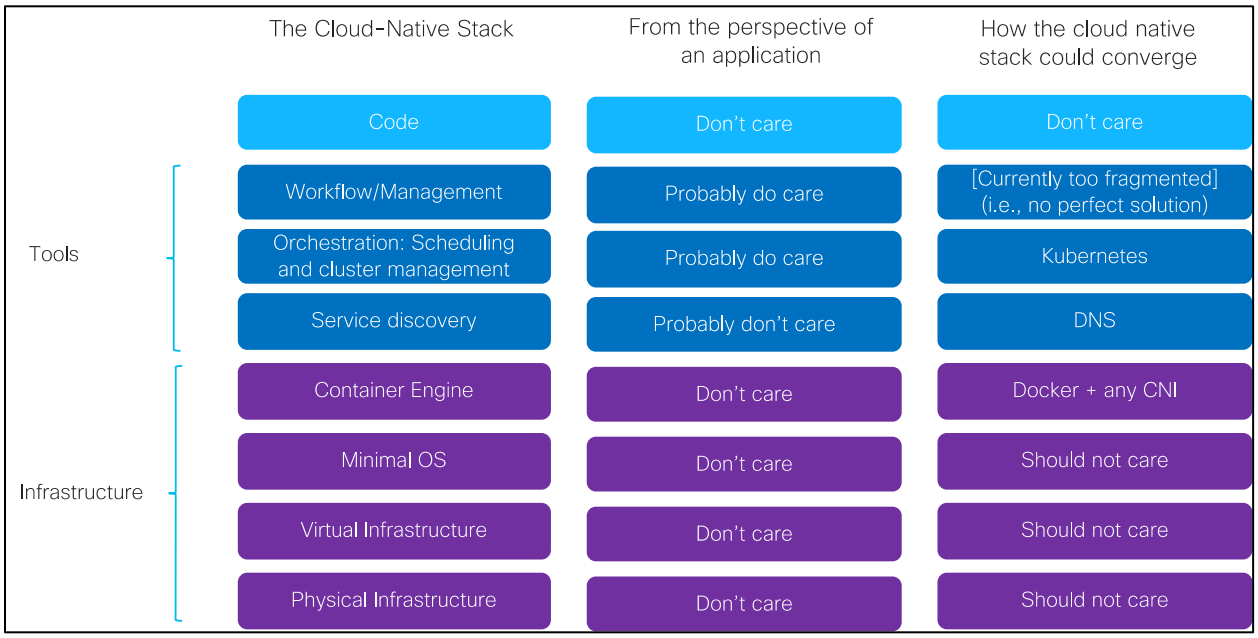


**Figure 4.6. Cloud Native Stack: Application and Consolidation of a Cloud-Native Tooling.**[70]

[70] (c) Cisco 2019

Figure 4.6 shows the Cloud Native stack, broken down into code, tool and infrastructure layers. It also shows two additional columns. The first one from the perspective of the application (in the case of 5G, the CNF or VNF) indicating where the application is sensitive to the Cloud Native stack. The second additional column represents how the multitude of tools and infrastructure could harmonize given time, in much the same way that Linux is viewed today.

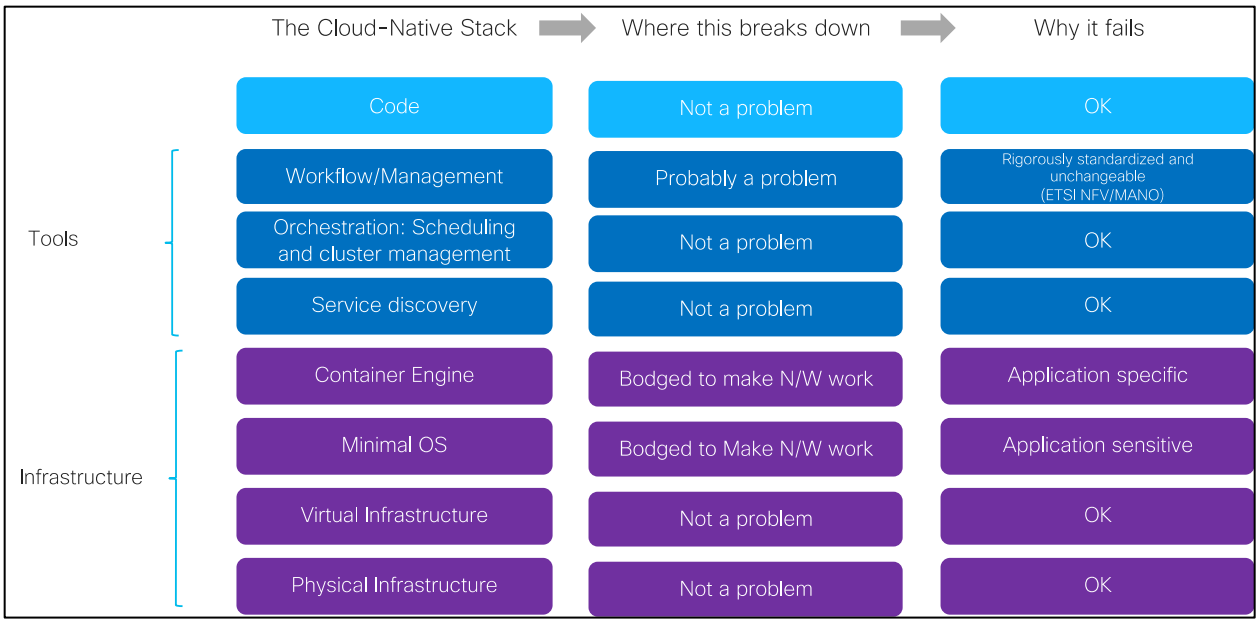


**Figure 4.7. Cloud Native Stack – from a Perspective of where the Current Cloud Native Infrastructure Breaks Down and How it Impacts Cloud Native Applications (CNFs and VNFs).[71]**

Figure 4.7 also shows the Cloud Native stack and two additional columns. The first additional column shows where the Cloud Native stack has problems because of the need to match the Cloud Native stack to the variety of operational environments into which it is deployed. The second additional column shows the consequential impact on the application.

What this figure highlights, is that when cloud application developers/operators deploy into a web-scale public cloud, they do so once. Meaning, that they choose their cloud platform and develop it exclusively. For example, they may choose Amazon Web Services (AWS) and then use the tooling provided by AWS to write and operate their application.

As a result, these application developers/operators see a homogenous system. That is not the case when a CNF/VNF vendor deploys into a specific Service Provider (SP) and finds a subtle variety of complex integration and deployment problems that need to be addressed. How do these subtleties and complexities arise? The following sections 4.2.1 and 4.2.2 offer explanations.

### 4.2.1 CLOUD NATIVE PHILOSOPHY-RELATED ISSUES

In one sense, the Cloud Native issues appear because the whole of the Cloud Native development philosophy has been applied without consideration of the actual deployment and operational environments.

[71] (c) Cisco 2019

In brief, the positive and negative aspects of Cloud Native from a network function perspective are summarized as follows:

- Positive - Cloud Native has undeniably improved development, delivery and test, in-service upgrades and improved version management
- Negative – the context in which Cloud Native was designed is being misrepresented or abused in two senses-
    - Cloud Native was designed for people who write and operate the applications. In today's cellular network, this clearly is not the case
    - Cloud Native was designed for applications in which long interruptions are tolerable, therefore, good reliability is measured in minutes of outage per month. This is also clearly not the case for communications networks where the expectation is that outages last less than 5.26 minutes per year

Moreover, this situation is compounded by the big and often overlooked point in the Cloud Native paradigm: that Cloud Native does not necessarily mean only containerization of workload. The implication (and in a sense also a benefit) of containerization is that it does bring more commonality to the platform, therefore, the container has to bring along less for itself (the extreme of this are serverless, lambda functions, and functions as a service).[72] What this also means in the SP environment, is that for Cloud Native to work, it will require greater harmonization of the tooling and infrastructure across all the SPs. There is a long time before this point is achieved.

### 4.2.2 AUTOMATION AND ORCHESTRATION ISSUES

As described in in Section 1 on DevOps and CI/CD, automation at every stage of lifecycle with appropriate tooling is key to making Cloud Native a reality. This calls for a more robust strategy in place like re-alignment of an organization to support services/product (DevOps strategy) and re-tooling for onboarding, managing and operating CNFs in SP's infrastructure.

It appears that standardised technology choices have been made for NFV orchestration without really understanding their impact on Cloud Native operation. For example, ETSI VNFM orchestration is locked to the Heat and Topology and Orchestration Specification for Cloud Applications, an Organization for the Advancement of Structured Information Standards (TOSCA) standards. Heat is a clone of AWS CloudFormation and a main project in OpenStack.[73] CloudFormation was intended for applications. It was not designed to keep the application running, nor scale, nor heal, and etcetera. In short, CloudFormations, in its current form, was never really evolved to support "dynamic" orchestration. The net effect is that ETSI took a tool intended to be the very beginning of orchestration and overloaded it with all of the other existing orchestration techniques available, leading to a bad design.

Why is this a problem? By comparison, for the cases ETSI MANO covers:

- When starting an application, it cannot sequence the bring-up

[72] Serverless architecture (also known as serverless computing or function as a service, FaaS) is a software design pattern where applications are hosted by a third-party service, eliminating the need for server software and hardware management by the developer. Applications are broken up into individual functions that can be invoked and scaled individually: https://www.twilio.com/docs/glossary/what-is-serverless-architecture

[73] https://wiki.openstack.org/wiki/Heat

- When running an application, it cannot repair failures beyond replacing components.

For the cases ETSI MANO does not cover:

- When trying an in-service-upgrade for an application, it is unable to do the fine-grained automation required (plus, the VNFM, as defined, does not cover upgrade)

Moreover, simply making a VNF into a CNF solves none of these problems if the old ecosystem is utilized. Unfortunately, locking into just the ETSI MANO-based standard can possibly exacerbate the problem, rather than fix it, when moving to Cloud Native computing architecture. The standards must be evolved over time to accommodate management and orchestration of CNFs in more Cloud Native way.

By contrast, Kubernetes is more flexible in dealing with the network functions deployed as microservices in a robust fashion, and need to be contended with as Cloud Native networking is remedied. Kubernetes also provides a basic service concept of declarative infrastructure where state of object (create, delete, update) is managed by a manifest (YAML-based template) and check live state of object, for example, "keep three instances running at all times." This characteristic of Kubernetes brings self-healing to CNFs which has been a tall order for NFVO in current NFV deployment. YAML-based object declarations also bring Infrastructure as Code (IaC) to fore where CNF deployment and its configuration are managed as source code, allowing for better CI/CD integration and automation.

Helm charts are used to define, install, and upgrade Kubernetes applications with packaging and configuration capabilities to make it very easy to share (similar to ansible modules). Kubernetes Operator[74] is an application specific controller which extends Kubernetes with service-specific behavior, for example, when updating databases, as "do these specific steps in order to preserve the data." In short, Kubernetes both provides basic orchestration and has an extending ecosystem of new types of orchestration that can be used for augmentation (Figure 4.6). An example of such augmentation would be "automate Day 2 lifecycle management of container applications similar to capabilities such as Generic-CNF manager to:[75]

- Leverage Custom Resource Definitions to deploy Kubernetes native services that can access Kubernetes API events
- Simpify Operator Software Development Kit (SDK)[76] creation of Operators in Go (or leverage Helm or Ansible automation)"

[74] https://coreos.com/blog/introducing-operators.html
[75] https://events.linuxfoundation.org/wp-content/uploads/2019/07/OSS2019-HS-k8sNativeInfra-OperatorFor5Gedge.pdf
[76] https://github.com/operator-framework/operator-sdk

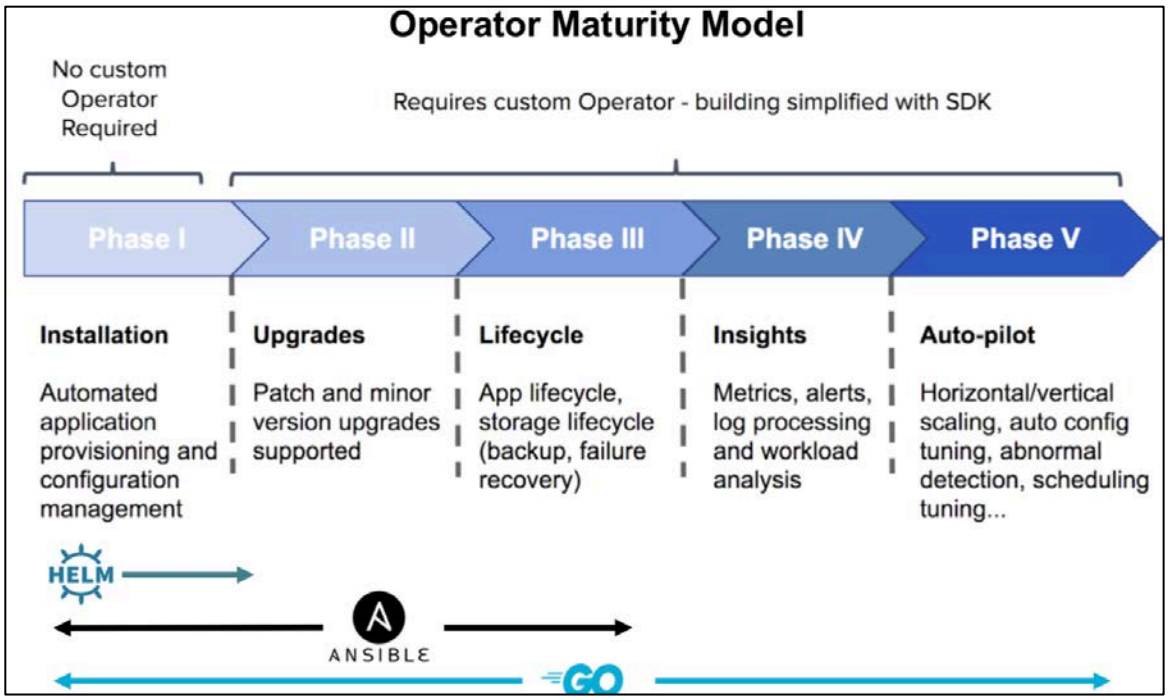


**Figure 4.8. Kubernetes Operator maturity Model.**[77]

### 4.2.3 HOW CAN WE MOVE FORWARD?

While the previous sections indicate some issues, there is an opportunity to re-evaluate how to move forward and bring the benefits of Cloud Native and the present world of applications into the world of NFV. How different is a VNF/CNF from an application (therefore, necessary re-evaluation). In terms of Cloud Native infrastructure, Kubernetes was designed to make the running of conventional enterprise applications easy. It was also designed to enable Cloud Native techniques in the building of applications. Two questions arise:

- Can a network function built on Kubernetes, be equally easy to build and operate?
- Can Cloud Native take advantage of the same techniques?

To answer, an understanding of, what are the missing Cloud Native support structures from the perspective of the service provider network, is needed.

### 4.2.4 ADDRESSING PLATFORM SHORTCOMINGS

Running network functions on Cloud Native infrastructure platform comes with its own challenges due to the nature of the workload. One of these challenges stems from the way platforms are run. Enterprise clouds are (usually) oversized and have spare capacity, whereas SP clouds are sized to their workloads. Moreover, in central Data Centers (DCs) workloads are large, and a small percentage of inefficiency can

[77] https://events.linuxfoundation.org/wp-content/uploads/2019/07/OSS2019-HS-k8sNativeInfra-OperatorFor5Gedge.pdf

generate a lot of Capital Expenditure (CAPEX) but can be usually tolerated. Whereas, in SP edge DCs, every dollar spent is multiplied by hundreds or thousands of deployments. Therefore, efficient running of applications saves huge cost in service provider DCs, in particular those at the edge of the network, where the compute resource constraints are the biggest challenge.

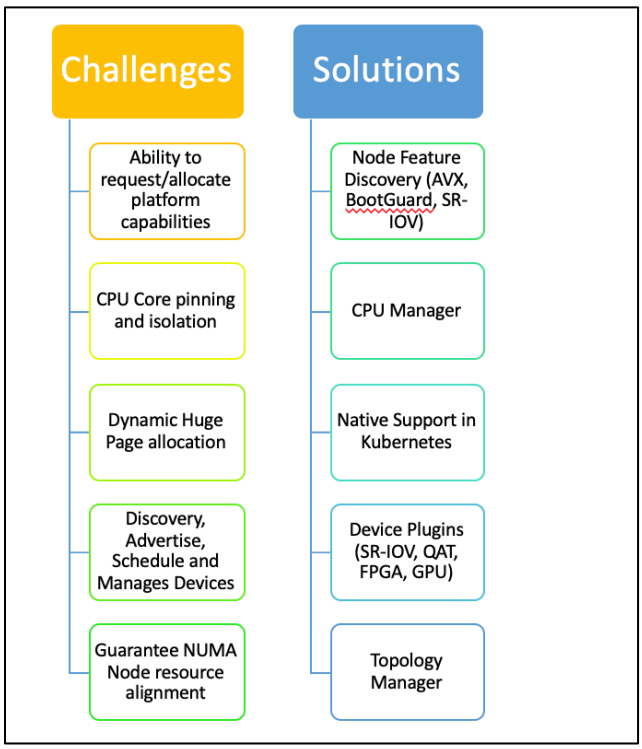


**Figure 4.9. Key Kubernetes Solutions for NFV Compute Related Performance Requirement.**

Running specialized network functions like Deep Packet Inspection (DPI), Gateway, Radio Access Network (RAN) software and etcetera, demands that the container orchestrator be more platform-aware, so that it can provide:

- deterministic and performant workload placement, along with
- capabilities to manage more exotic hardware like Application-Specific Integrated Circuit (ASIC), SmartNICs (Network Interface Cards), Graphic Processsing Units (GPU), Field-Programmable Gate Array (FPGA), and etcetera

A CPU manager for Kubernetes that provides basic core (compute) affinity for NFV workload is now available as part of the latest Kubernetes General Availability (GA) release (v1.5) and can help with workloads of the following specific characteristics:[78]

- Sensitive to CPU throttling effects
- Sensitive to context switches
- Sensitive to processor cache misses
- Benefiting from sharing processor resources (for example, data and instruction caches)

[78] https://kubernetes.io/blog/2018/07/24/feature-highlight-cpu-manager/

- Sensitive to cross-socket memory traffic
- Sensitive or requiring hyper-threads from the same physical CPU core

The noisy neighbor problem[79] of running NFs in containers has been a priority concern and a deterrent in the adoption of CNF. This issue is being solved by the CPU isolation feature in Kubernetes which also helps improve performance (Figure 4.10). Furthermore, other challenges like Non-Uniform Memory Access (NUMA) node, node feature discovery, dynamic Huge page allocations, and etcetera, can now be managed natively from the Kubernetes (Figure 4.7). This shows that quite a bit of work is being done by the Open Source community (LF, CNCF) to make Kubernetes the choice for network functions.

| Packet size | Throughput without CPU pinning and CPU isolation (average, Gbps) | Packet rate without CPU pinning and CPU isolation (average, Mpps) | Throughput with CPU pinning and CPU isolation (average, Gbps) | Packet rate with CPU pinning and CPU isolation (average, Mpps) | Performance gain |
|---|---|---|---|---|---|
| 64B | 6.59 | 9.85 | 12.20 | 18.30 | 185.27% |
| 128B | 11.00 | 9.31 | 20.36 | 17.22 | 185.03% |
| 256B | 10.19 | 4.61 | 18.86 | 8.56 | 185.12% |
| 512B | 13.48 | 3.18 | 22.79 | 5.35 | 169.15% |
| 1024B | 13.51 | 1.60 | 25.07 | 3.00 | 185.55% |
| 1518B | 13.51 | 1.10 | 25.10 | 2.00 | 185.77% |

**Figure 4.10. Benchmark Testing Showcasing Performance Improvement Coming from CPU Pinning and CPU Isolation.[80]**

### 4.2.5 ADDRESSING NETWORKING RELATED ISSUES

As indicated in the preceding section, there are also some networking and network performance issues that need to be addressed. Kubernetes CNI is a plugin-based approach to networking with specifications and a set of libraries written in Golang, exposing 3rd party networking plugins. CNI's only goal is to provide network reachability to containers and deallocate networking resources when containers cease to exist. CNI is very rigid about exposing the network interface, as only one interface with addressable IP assignment per Pod is allowed.

Enterprise networking is trivial with Kubernetes, because it has found way of exposing the kernel's BSD (Berkeley) Socket as an API. However, NFV data planes pose a different problem. For example, applications do packet-by-packet networking in an efficient user-space (Data Plane Development Kit or DPDK) data plane. Where there are high volumes of traffic, switching packets to and from containers is often too expensive, since the resulting infrastructure networking is a huge consumer of CPU resources. Kubernetes currently does not have native feature functionality for fast data path networking. Moreover, NFV requires additional network features such as:

- Exotic protocols (by enterprise application standards)
    - Stream Control Transmission Protocol (SCTP)
    - GPRS Tunneling Protocol (GTP)
    - Packet Forwarding Control Protocol (PFCP)

[79] The noisy neighbor problem is a term used to refer to the uneven cloud network performance of virtual machines and applications that results from sharing infrastructure.

[80] https://builders.intel.com/docs/networkbuilders/enhanced-platform-awareness-feature-brief.pdf.

    - Bidirectional Forwarding Detection (BFD)
- Unusual behavior (by enterprise application standards)
    - Uplink bonding via Link Aggregation Control Protocol (LACP) or Border Gateway Protocol (BGP) anycast
    - EXPRESS (EXplicitly REquested Single Source) Count Management Protocol (ECMP) between containers
- Unusual addressing (by enterprise application standards)
    - Multiple addresses / loopbacks per container
    - Bump-in-wire

Kubernetes was not designed for this type of construct. For example, it has no indigenous Cloud Native feature to address these use-cases ensuring that the network function operates as expected. This has led to innovative solutions in terms of CNI plugins currently filling these gaps.

From a CNI perspective, there are solutions which address such deficiencies, but have their own drawbacks, such as:

- Contiv+Vector Packet Protocol (VPP) CNI plugin. However, this does not support container-to-container chains, pod-to-pod and node-to-node, nor container-to-container
- Flannel CNI plugin. However, this is Internet Protoco Version 4 (IPv4) only
- Cilium CNI plugin. This only addresses Layer 3/4
- Open Virtual Networking with Kubernetes using Open vSwitch (OVS). This is overlay network only
- Open vSwitch CNI plugin.This is not high performance, requires MULTUS, and manual L2/L3 overlays

Most of the CNIs mentioned do not address every problem related to CNF networking requirements. Since the way CNI is tailored is to address reachability by any means, it is not necessarily the most appropriate API for enhanced networking required by CNFs.

Plugins like MULTUS and DANM[81] allow for meta-plugin layers to the handle setting up of connectivity by attaching multiple network interfaces to pods. This enables the provision of multiple interfaces to be exposed as required by a CNF to function. The problem with these plugins is that they are short term solutions for multiple network attachments per pod. That said, there is a CNI plugin for DPDK with SR-IOV support. There is also a vhostuser CNI plugin which can support DPDK accelerated OvS or VPP.

There are use cases, for example, where there is a need for finer-grained control of their lower-level networks at layer 2/layer 3. That is what the Network Service Mesh project aims to offer: the same discoverability, durability, configurability, and security as a service mesh but for layer 2/ layer 3. As the pathway to multi-cluster connectivity, Network Service Mesh can be used to essentially abstract the concept of a distributed service. Network Service Mesh is not yet delivering all of these, as its development is in the nascent stage. Network Service Mesh is designed to provides these “missing” Kubernetes networking

[81] For those interested, DANM actually does mean “Damn, Another Network Manager”. https://github.com/nokia/danm#introduction

capabilities by using a simple set of APIs designed to facilitate connectivity: between containers running services or with external endpoints.

## 4.3 A NOTE ON SECURITY

5G Americas has already published two whitepapers that highlight 5G and security. These two papers cover the whole gamut of security in 5G from the perspective of the inherent 3GPP security architecture, from the resulting threat surface[82] to the extension of security in a 5G slice environment.[83] *The Evolution of Security in 5G: a "Slice" of Mobile Threats* is particularly relevant, in as much as the resulting slice instance could be the result of many SBA and micro-serviced functional interactions on a cloud platform.

It is not the purpose of this white paper to go into the detail of 5G security (the previous two 5G Americas whitepapers have done this), or how security in cloud platforms have developed[84] [85] [86] [87] [88]. What is of greater relevance to this paper, are the cloud platform infrastructure contributions that are necessary for the security of the system.

For example, because of the potential multi-tenant nature of slices, the security needs of such an environment are primarily satisfied though security isolation.[89] In essence, this means implementing either policies for how CNFs/VNFs are run in the cloud, and/or, mechanisms that enforce where the application is run (the physical host, type of hardware, and etcetera).

While security isolation advocates a pragmatic way of assessing how a cloud-native solution should be handled in shared infrastructure, it does not paint the complete picture of how security in Cloud Native environments is handled. Since Cloud Native applications also have a CI/CD connotation to them, then security for these applications takes on a multi-layered approach.[90] Paraphrasing that example heavily, we can see that security in cloud-native CI/CD environments has three layers- development, deployment and operational:

**Development layer-** The characteristics of this layer would include:

- Secure DevOps - security policies and architectures to develop applications and workloads on the cloud with security applications to the application from inception
- Automation - integrated automated provisioning of security policies, security technologies and vulnerability scanning as the application passes through staging to deployment

**Deployment layer**- The characteristics of this layer would include:

---

[82] 5G Americas, The Evolution of Security in 5G, October 2018.
[83] 5G Americas, The Evolution of Security in 5G: a "Slice" of Mobile Threats, July 2019.
[84] Security as a Service Model for Cloud Environment, Vijay Varadharajan and Udaya Tupakula, IEEE Transactions on Network and Service Management, Volume 11, 2014.
[85] ISGcloud: a Security Governance Framework for Cloud Computing, Oscar Rebollo, et al, The Computer Journal, Volume 58, 2015.
[86] Using Virtual Machine Allocation Policies to Defend against Co-Resident Attacks in CloudComputing, Yi Han et al, IEEE Transactions on Dependable and Secure Computing, Volume 14, 2017.
[87] Towards Achieving Data Security with the Cloud Computing Adoption Framework, Victor Chang and Muthu Ramachandran, IEEE Transactions on Services Computing, Volume 9, 2016.
[88] Cloud-Trust – a Security Assessment Model for Infrastructure as a Service (IaaS) Clouds, Dan Gonzales et al, IEEE Transactions onnCloud Computing, Volume 5, 2017.
[89] A Study of Security Isolation Techniques, Rui Shu, et al, ACM Computing Surveys, Vol 49, No. 3., October 2016.
[90] IBM, Deploying unified security in a hybrid cloud world: https://www.ibm.com/downloads/cas/M49Z0DBL

- Identity and Access Management – the application of uniform identity and access policies, for the deployed components across multiple platforms
- Data Protection – security controls must be consistent across multiple platforms
- Automated security and cloud workload management – security and software service solutions that leverage automation to (continually) scan for vulnerabilities and applicationly polices and security fixes across the cloud ecosystem at the appropriate scale
- Visibility and intelligence – access and buildup of behavioral information on the application (visibility) such that anomalous behavior can be detected (intelligence)
- Cloud network security – the same level of security exists across the system over which the application exists/runs

**Operational layer**- The characteristics of this layer would include:

- Security operations and threat management - central policy management and visibility across cloud and workloads supported by a unified security framework

Given the nature and architectural implications of Cloud Native functions, security for cloud-native functions has to be considered over the whole lifecycle of the functions.

## 5. CONCLUSION

The Cloud Native landscape as a technology has a rich Open Source-supporting ecosystem that has enabled the Cloud Native approach to build and run applications that fully exploit the various benefits and cost efficiency of the cloud computing model. This new model includes services architectures, infrastructure-as-code, automation, continuous integration/delivery pipelines, monitoring tools, and etcetera.

Applications built and deployed with the Cloud Native pattern have some common characteristics:

- They are composed of microservices, therefore, each application is a collection of small services that can be operated independently of one another. Microservices are often owned by individual development teams that operate on their own schedule to develop, deploy, scale and upgrade services
- Cloud Native applications are packaged in containers, aiming to provide isolation contexts for microservices. Containers are highly accessible, scalable, easily portable from one environment to another and fast to create or tear down. This flexibility makes them ideal for building and running applications composed of microservices
- Cloud-native applications are built and run on a continuous delivery model, supporting fast cycles of build, test, deploy, release, and develop. This helps software service developers and infrastructure IT operations teams to collaborate with one another for building, testing and releasing software updates as soon as they are ready, without affecting end-users or developers of other teams. Such a model encourages the adoption of DevOps principles, fostering collaboration between software service developers and infrastructure IT operations. This model also encourages the creation of a behavioral culture where building, testing and releasing services happlicationens more rapidly, frequently and consistently

Cloud Native applications are dynamically managed, often built and run on modern platforms such as Kubernetes, which offer hardware decoupling, and which is critical in terms of deployment automation, scaling and management.

In the context of application developer-operators placing their workloads into public clouds run by large Web-scale cloud service providers, there is no doubt that this approach has been successful.

However, in reviewing the requirements of 5G and applying the Cloud Native model, it is an attractive direction for many reasons including flexibility, cost savings and efficiency. The model does not map completely to the SP's mobile network operational needs. That said, the standardization community for 5G has adopted a modular approach to the way the 5G core is architected, encouraging a Cloud Native approach to its implementation. There are gaps, and the telecommunications industry is still on the Cloud Native journey as exemplified by Figure 5.1.

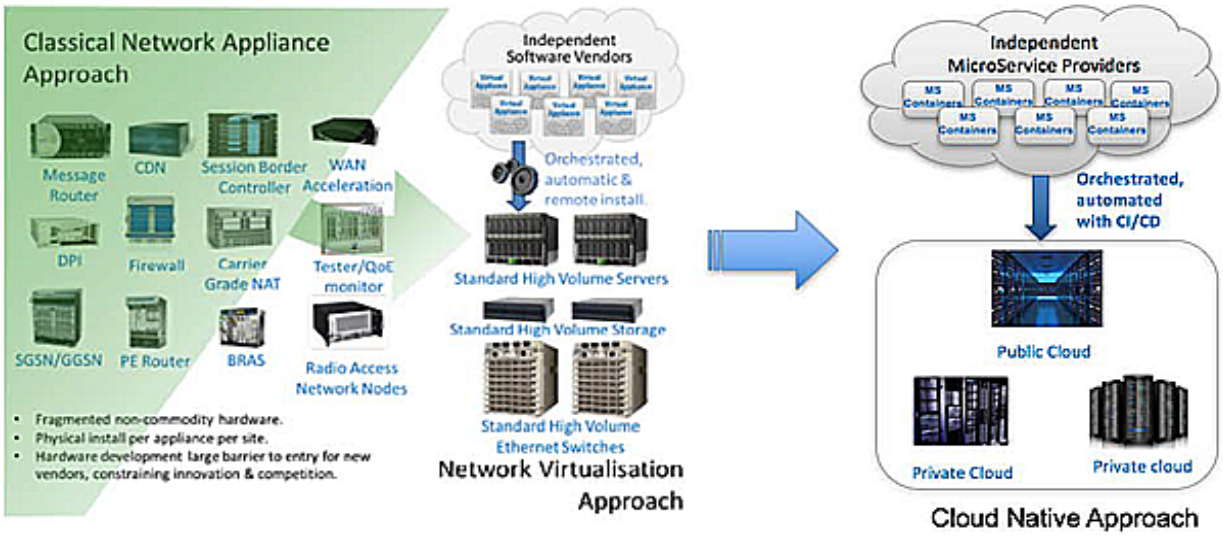


**Figure 4 Migration from Classical Networking Appliance to Cloud Native Network Function.**[91]

Cloud Native is being implemented and migration strategies are being employed as depicted in Figure 5.2. Some best practices and (predictably) some issues around the reality of deployment are explained in this report. There are application dependencies on several layers of the Cloud Native stack, in particular, in the workflow management, container engine and minimal OS layers. These dependencies need to be overcome if Cloud Native is to fulfil the expectations 5G implementation has placed on them. Moreover, because the latest focus of the 3GPP standard (Rel-17) is on the application of 5G to enterprise and verticals, and when we consider market evidence such as that shown in Figure 5.2, there is a window of three to four years during which issues need to be resolved, if the required network flexibility is to truly unlock the (illusive) new 5G business models.

[91] © 2017 SCTE-ISBE and NCTA

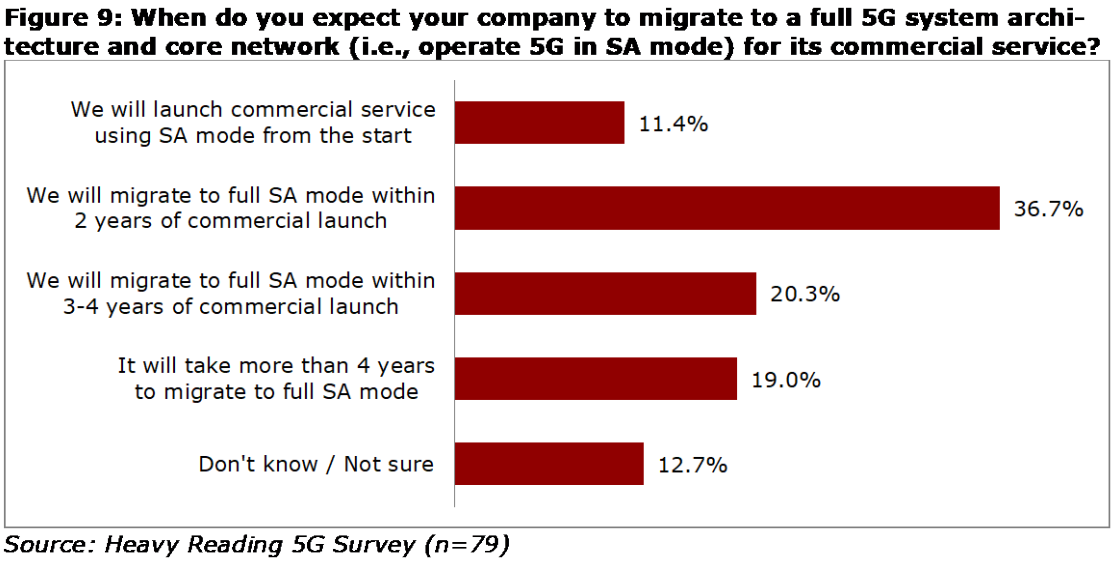


**Figure 5.25. Expectation on the Migration to 5G Core.**[92]

The shift to the cloud-native mindset, especially for the telecom sector, will not be easy. It will take a phased approach to reap the full benefits, with the onus on the operators deploying 5GC on a properly configured and controlled Cloud Native stack. It will require a massive effort in terms of re-architecting the network functions and re-tooling for automation and orchestration to be cloud ready. Only then will the true flexiblilty and cost efficiency benefits of moving to the Cloud Native infrastructure be realized.

[92] https://www.interdigital.com/white_papers/5g-deployment--service-strategies-an-operator-survey

# APPENDIX

## A. ACRONYM LIST

| Acronym /Abbreviation | Explanation |
|---|---|
| **3GPP** | Third Generation Partnership Project, global standards organization |
| **4G, 5G** | 4th Generation, 5th Generation |
| **5GC** | 5G Core |
| **5G-MoNArch** | 5G Mobile Network Architecture |
| **5GNR** | 5G New Radio |
| **5G PPP** | 5G Public and Private Partnership |
| **AF** | Application Function |
| **AMF** | Application Management Function |
| **API** | Application Programming Interface |
| **ARPU** | Average Revenue Per User |
| **ASIC** | Application-Specific Integrated Circuit |
| **AUSF** | AUthentication Server Function |
| **AVX** | Advanced Vector Extension |
| **BFD** | Bidirectional Forwarding Detection |
| **BSD** | Berkeley Socket |
| **CAPEX** | Capital Expenditure |
| **CI/CD** | Continuous Integration/Continuous Delivery |
| **CIM** | Cloud Infrastructure Manager |
| **CNCF** | Cloud Native Computing Foundation |
| **CNF** | Cloud Native Network Function |
| **CNFFG** | Cloud Native Network Function Forwarding Graph |
| **CNFI** | Cloud Native Network Function Infrastructure |
| **CNFM** | Cloud Native Network Function Manager |
| **CNFMS** | Cloud Native Network Function MicroServices |
| **CNI** | Container Network Interface |

| | |
|---|---|
| **COTS** | Common Off The Shelf |
| **CRUD** | Create Read Update Delete |
| **(C)SP** | (Communication) Service Provider, e.g. AT&T, Telefonica, T-Mobile, Sprint, and etcetera |
| **CUPS** | Control and User Plane Separation |
| **CPU** | Computer Processing Unit |
| **DC** | Data Center |
| **DDD** | Domain-Driven Design |
| **DevOps** | Development and Operations |
| **DPDK** | Data Plan Developers Kit |
| **DPI** | Deep Packet Inspection |
| **EPC** | Enhanced Packet Core |
| **eSBA** | enhanced Service Based Architecture |
| **ETSI** | European Telecommunication Standards Institute |
| **FPGA** | Field Programmable Gate Array |
| **FUD** | Fear Uncertainty and Doubt |
| **F/W** | Firewall |
| **GA** | General Availability |
| **GPUs** | Graphic Processing Units |
| **GTP** | GPRS Tunneling Protocol |
| **IaC** | Infrastructure as Code |
| **IDL** | Interface Description Language |
| **IP** | Internet Protocol |
| **IPv4** | Internet Protocol version 4 |
| **IT** | Internet Technology |
| **LF** | Linux Foundation |
| **NEF** | Network Exposure Function |
| **NFV** | Network Function Virtualization |
| **NFVI** | Network Function Virtualization Infrastructure |

| | |
|---|---|
| **NFVO** | Network Function Virtualization Orchestrator |
| **NGC** | Next Generation Core |
| **NGMN** | Next Generation Mobile Network Alliance |
| **NSSF** | Network Slice Selection Function |
| **NRF** | Network Resource Function |
| **NSA** | Non-Standalone |
| **NUMA** | Non-Uniform Memory Access |
| **MANO** | Management And Orchestration |
| **MC** | Mobile Core |
| **MOU** | Memorandum of Understanding |
| **NGC** | Next Generation Core or 5G Next Generation Core (also 5GC) |
| **OS** | Operating System |
| **PCF** | Policy Control Function |
| **PFCP** | Packet Forwarding Control Protocol |
| **PPS** | Packets Per Second |
| **QAT** | Quick Access Toolbar |
| **QoS** | Quality of Service |
| **RAM** | Random Access Memory |
| **RAN** | Radio Access Network |
| **RESTful** | Representational State Transfer conforming Web services |
| **SA** | Standalone |
| **SBA** | Service Based Architecture |
| **SBI** | Service-Based Interface |
| **SCTP** | Stream Control Transmissiom Protocol |
| **SDK** | Software Development Kit |
| **SDN** | Software Defined Networking |
| **SmartNICs** | Smart Network Interface Cards |
| **SMF** | Session Management Function |
| **SOA** | Service Oriented Architecture |

| | |
|---|---|
| **SR-IOV** | Single Rool I/O Virtualization Standard |
| **TCP/IP** | Transmission Control Protocol/Internet Protocol |
| **TOSCA** | Topology and Orchestration Specification for Cloud Applications (Organization for the Advancement of Structured Information Standards) |
| **TUG** | Telecom User Group |
| **UDM** | Unified Data Management |
| **UE** | User Equipment |
| **UPF** | User Plane Function |
| **VM** | Virtual Machine |
| **VNF** | Virtual Network Function |
| **VPP** | Vector Packet Protocol |
| **YAML** | YAML Ain't Markup Language |

## ACKNOWLEDGEMENTS

The mission of 5G Americas is to advocate for and facilitate the advancement of 5G and the transformation of LTE networks throughout the Americas region. 5G Americas is invested in developing a connected wireless community for the many economic and social benefits this will bring to all those living in the region.

5G Americas' Board of Governors members include AT&T, Cable & Wireless, Ciena, Cisco, CommScope, Ericsson, Intel, Kathrein, Mavenir, Nokia, Qualcomm Incorporated, Samsung, Shaw Communications Inc., Sprint, T-Mobile USA, Inc., Telefónica and WOM.

5G Americas would like to recognize the significant project leadership and important contributions of leaders Milind Gunjan of Sprint and Sam Samuel of Cisco, along with many representatives from member companies on 5G Americas' Board of Governors who participated in the development of this white paper.