The Cloud Native Telco

November 2019

metaswitch

metaswitch

Contents

Executive Summary

Faced with a continuing decline in revenue per bit and a constantly growing demand for bandwidth, network operators must look for creative ways to drive down the cost per bit as quickly as possible, while at the same time seeking out opportunities for new value-add.

NFV was supposed to address these needs by reducing Capex through exploitation of industry-standard hardware, reducing Opex by aggressive automation of operations, and by accelerating innovation through the flexibility of software-based networks and the agility of specialist software vendors, but it has largely failed in this mission. Network operators now recognize that simply replacing physical network functions with equivalent virtualized software appliances, as prescribed by ETSI standards, is only scratching the surface of what is possible when you move to a pure software world. The ultimate promise of NFV will only be realized with new network function software systems that are designed from the ground up to exploit the power of the cloud: so called "cloud native".

5G is crying out for a radical new approach to NFV. The far greater capacity provided by the 5G RAN will drive up traffic volumes dramatically, but with no increase in ARPU, necessitating steep reductions in cost per bit. At the same time, the new revenue opportunities offered by 5G, particularly in the enterprise space, require operators to be far more agile, and demand new techniques such as network slicing that need unprecedented levels of operations automation.

NFV was supposed to help network operators approach the agility and operations efficiency of the Web-scale players. That would represent a massive leap forward from where they are today, and would greatly improve the returns from the massive investments that are being made in 5G. But this cannot be achieved by continuing to pursue a traditional approach to NFV. The cloud native approach is vastly different, and vastly better. It will go a long way to closing the efficiency and agility gaps between telcos and Web-scale players, precisely because cloud native is the way the Web-scale world has always worked.

Every aspect of working with cloud native network functions (CNFs) across their entire life cycle is far quicker, far easier, far less resource intensive and far less error-prone that it has been with traditional virtual network functions (VNFs). The difference is truly transformative. But the massive difference between the cloud native approach and the traditional approach to NFV means that it's hard to get there from here. Essentially, cloud native represents a discontinuity in the telco networking technology landscape. There is no realistic evolution path from VNFs to CNFs: cloud native is a step change.

Embracing a cloud native approach to NFV requires a bold step. It probably means letting go of some comfortable incumbent vendor relationships, and it certainly means re-engineering some key aspects of the procurement process. This won't be easy for many network operators. But if our experience is anything to go by, it will turn out in retrospect to be a far more positive experience than could possibly have been expected up front. And it will bring those network operators who have the courage to take that bold step to a far better place.

Introduction

The vision of Network Functions Virtualization seemed to many in the industry at the time to be a bold one: a decisive move away from reliance on physical networking boxes to pure software implementations of network functions running on commodity compute hardware.

But in retrospect, the original NFV vision was seriously lacking in ambition. By focusing on how physical network functions would be replaced by equivalent "software appliances", it failed utterly to foresee the extent to which the design and architecture freedoms of working purely in software can bring about a radical transformation of the telco technology landscape.

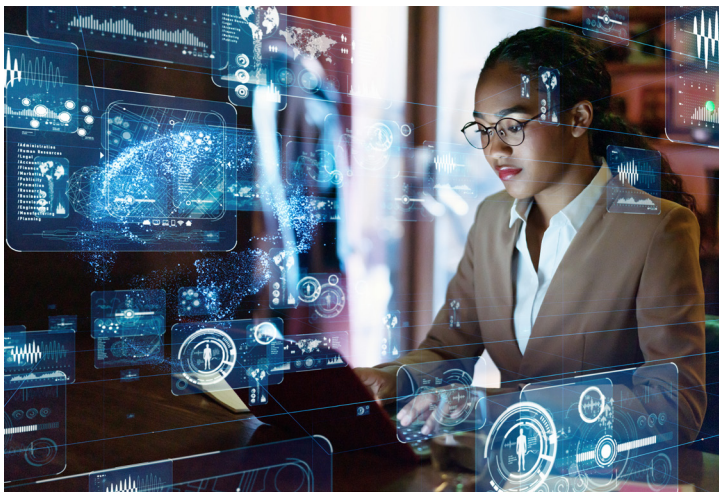
As a long-standing developer of communications and networking software, we have always paid close attention to evolving practices in software design, development and deployment. And it was apparent to us, even in October 2012, that the "software appliance" model of NFV could only ever be an interim, tactical approach. The Web-scale world was already demonstrating an ability to build massively scalable and resilient applications deployed on cloud computing infrastructures, exploiting techniques such as stateless processing and decomposition into microservices, and it seemed to us that this kind of approach was equally applicable to many of the network functions that service providers rely on. When we started building our Clearwater IMS product, starting from scratch in 2012, we fully embraced those Web-scale practices and subsequently delivered the first carrier-grade network function solution that could truly be called cloud native.

For most network operators, the NFV journey so far has been a painful one, with many disappointments along the way. The traditional telco equipment vendors, faced with a massive reduction in hardware revenues, understandably dragged their feet and exploited their power of incumbency to pressurize customers to purchase "full stack" NFV solutions. These NFV siloes are typically not open to third-party VNFs, so this approach to NFV increases vendor lock-in, the precise opposite of what was intended by the original architects of NFV. Those network operators who were courageous enough to insist on deploying a vendor-neutral infrastructure found that many VNF vendors struggled to on-board their products, which often delivered poor performance and consumed excessive hardware resources when deployed. And whichever approach was taken to NFV infrastructure, the Opex savings promised by operations automation have proved elusive – mostly because the VNFs were simply ported from proprietary hardware with their traditional Command Line Interfaces, and are manifestly unsuited for any useful degree of automation.

It's not all bad news, of course. There's no question that at least some network operators have been able to drive down Capex by negotiating aggressively on the prices of VNF software licenses, and the elimination of end-of-life events for physical network functions will unquestionably bring further cost benefits in the medium to long term. Nevertheless, most network operators now understand that there's potentially far more to NFV than these relatively meagre benefits, if NFV is approached in the right way. And that "right way" is cloud native.

What Cloud Native Looks Like

It is not the intention here to describe the detailed technicalities of cloud native network function (CNF) architecture and design, but rather to focus on what a cloud native approach to network function virtualization delivers and how this differs from a software appliance view of the world.



Readers who wish to gain more insight into the technicalities of cloud native should refer to our white paper "Cloud Native Network Functions: Design, Architecture and Technology Landscape."

In this section, we will compare and contrast the experience of working with VNFs and CNFs across the full range of life-cycle management activities. We will use the following definitions:

- **PNF** refers to a physical network function.
- **VNF** refers to a virtualized network function designed the traditional way, that is as a software appliance, often ported from a physical network function, and deployed as a Virtual Machine.
- **CNF** refers to a cloud native network function, designed from the ground up to be deployed in the cloud, typically based on stateless processing elements combined with separate state stores, and offering scale out capabilities with N+k redundancy for resilience.

On-Boarding

A CNF will usually be delivered packaged in containers that are designed to be deployed and managed by Kubernetes. This type of application packaging has proven to be extremely portable, and a CNF should just come up and run on any standard Containers-as-a-Service infrastructure that embodies Kubernetes. This includes commercial distributions intended for building private clouds such as Red Hat OpenShift or VMware Pivotal Container Service, as well as public clouds including Azure Containers, Amazon EKS, Google Kubernetes Engine and IBM Cloud Kubernetes Service.

Some CNFs, especially those that perform user plane processing, will be designed to expose multiple network interfaces, and may also be designed to leverage hardware acceleration technologies such as SR-IOV. These require specific versions of Kubernetes that may not be widely supported in public cloud services, but which are supported in commercial distributions of Kubernetes platforms for private clouds. Although slightly more demanding in their requirements, these kinds of CNFs should still just come up and run on these platforms, without the need for specialist tweaking of detailed low level configuration or modification of the software.

By contrast, many VNFs have complex dependencies on specific drivers, OS or middleware capabilities and even specific hardware functionalities. They typically don't "just run" on any arbitrary generic cloud platform, and it may take a vendor weeks of effort from specialists to successfully stand up a VNF on a given NFV infrastructure.

OSS/BSS Integration

From the point of view of IT integration, a VNF often looks identical to the PNF that it is replacing. In this case IT integration may be trivial, because essentially nothing has changed.

A CNF is a fundamentally different thing to manage. Instead of the software equivalent of a "box", a CNF is a system that consists of a dynamic and varying population of software instances of various kinds that work together cooperatively to deliver the service that is required.

There are two possible approaches for managing CNFs. One is to build a software wrapper around the CNF as a whole that makes it look like a box. This wrapper is responsible, for example, for collecting metrics from the population of container instances that make up the CNF and aggregating them for presentation to the OSS. This approach has the advantage of leveraging existing OSS investments, but by treating the CNF as a box it fails to provide useful insights into what is going on under the covers.

The other approach is to embrace the open source cloud native software ecosystem, which offers numerous tools for managing cloud native applications, for example:

- Prometheus – for collecting metrics and storing in a time series database
- Grafana – visualization tool for building dashboards, based on data stored by Prometheus
- Fluentd – unified logging layer for collection, storage and analysis of logs

These powerful tools, which are very widely used by Web-scale operators, make it quick and easy to put together solutions for managing CNFs that respect the essential nature of cloud native applications. They should be considered as the basis of a cloud native approach to next-generation OSS.
There is no one right answer for managing any given CNF: the pragmatic solution is actually to blend aspects of the two approaches described here so as to best meet the needs of service management in the most cost-effective and timely manner.

Deployment

The deployment process for a network function consists of two main steps: instantiating the software on the cloud infrastructure, and injecting "day zero" configuration which is required to get the software running.

You typically deploy CNFs with the aid of Helm chart. This is a document that specifies the desired state of the application deployment, for example which container images to instantiate and how many of each to deploy. It also specifies the required network connectivity and the network policy that should be applied to each container workload.

CNFs require an absolute minimum of day zero configuration to come up and start running. IP addresses are automatically assigned to the default network interface of each container, and containers that need to talk to each other discover the relevant IP addresses via DNS or some more advanced technique such as a service mesh. Note that you can also hard-assign specific IP addresses to containers where this is a fundamental requirement for service delivery.

You can automate the deployment of VNFs to some degree, for example in an OpenStack environment with Heat templates. However it is rare for VNFs to support automated network address assignment, and a tedious process of manual IP address assignment is typically required. VNFs also don't typically have any service discovery mechanisms, so to the extent that different components of a VNF need to talk to each other, you have to configure the relevant IP addresses for such communications on each and every instance.

Configuration Management

Most complex network functions offer numerous options that must be configured appropriately for the network function to properly fulfil its mission in the network.

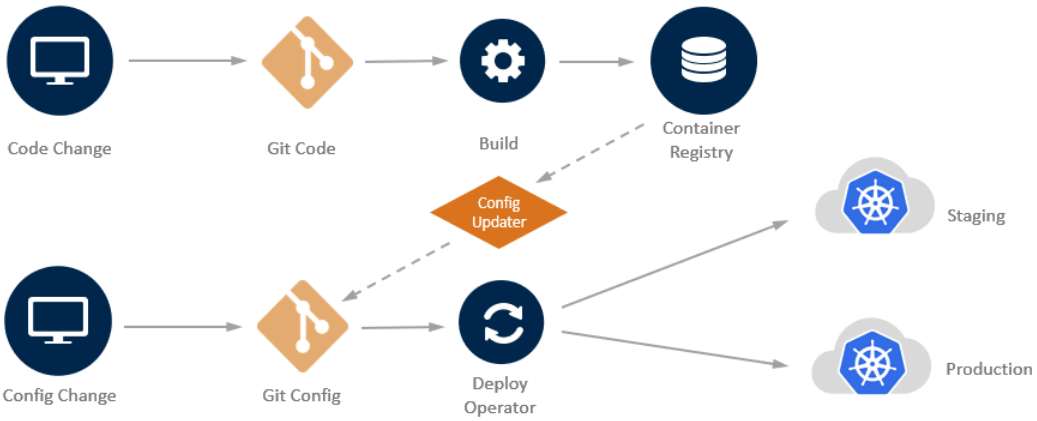
Configuration of a CNF is document-driven. You capture the desired configuration in a document, typically in YAML or similar format, and you maintain this document in a standard version-controlled repository, typically based on Git. When you commit a change to this document, it is automatically applied to the CNF as a whole. This means that all running container instances that comprise the CNF have their configuration updated automatically.
If there is a problem with the configuration change, you can roll it back simply by reverting to the earlier version of the config doc in the Git repository. All configuration changes are tracked, and the version control audit trail provided by Git enables anyone to see who made what changes and when. This approach is known as "configuration-as-code".

In more advanced CNF setups, you can subject configuration changes to "canary testing". This involves applying the config changes to some specific subset of the CNF's container instances and verifying that they are operating as expected before rolling out the change to the entire CNF.

By contrast, you generally configure VNFs by performing operations on individual VNF instances via a Command Line Interface, following a MOP (Method of Procedure). You often have to perform this manually, by typing commands. Effort may be reduced by pre-scripting a sequence of commands, and cutting and pasting script fragments into a CLI console. You have to separately configure each instance of a VNF that supports a particular service. It hardly needs to be said that this process is time-consuming and error-prone. In many networks, the majority of network outages are caused by mis-applied configuration changes. To back out a mis-applied configuration change, you typically have to type in a series of commands to overwrite the bad config with the previous good config.

If VNF configuration changes are expected to be frequent, it is sometimes worth the effort to automate these. Some VNFs expose APIs that support programmatic configuration update. To leverage these APIs to automate configuration changes, you need to develop additional software that can call the APIs, and that can track the changes that have been made.

CI/CD Pipeline with Declarative, Version-controlled Configuration



Healing

Software and hardware failures will occur from time to time, and both CNFs and VNFs will incorporate mechanisms to ensure that the service they support continues to be delivered following such failures without any unacceptable interruption or degradation. Healing refers to all of the supporting processes that ensure that the system is maintained in its normal resilient state.

When comparing the healing aspects of CNFs and VNFs, it is useful to invoke the "cattle vs pets" analogy. A CNF is like a herd of cattle: an undifferentiated population of instances that collectively provide some service, where the health of any individual instance has little bearing on the overall output of the herd. When an individual container instance in a CNF is misbehaving in any way, we simply kill it and instantiate a new one. Kubernetes automates this process in a very straightforward way.

By contrast, a VNF is like a pet. You care deeply about the health of each individual VNF instance, which is usually protected against complete failure by a paired standby instance. When a VNF reports an issue, for example by emitting an alarm, you may well attempt to get the VNF back into a good state by performing a variety of manual procedures on it. If a VNF instance fails completely, the service will be protected by the backup instance, but it is still necessary to restore the failed instance in order to return the system to its normal resilient state. This may require a complex, multi-step procedure to bring the failed VNF instance back to life. Automating these operations is usually far from easy.

The process for applying software upgrades to CNFs is straightforward and invariably highly automated.

It's perhaps worth pointing out that the resiliency model for CNFs is invariably N+k whereas for VNFs it is usually 1+1. Consequently, CNFs tend to consume a lot less compute resource than VNFs.

Scaling

The cattle vs pets analogy is also useful for comparing and contrasting CNFs and VNFs as it relates to scaling.

If you want more milk, you simply add some extra cattle to your herd. Likewise, if you want more capacity out of your CNF, you simply instantiate more containers. Kubernetes can take care of this automatically. When new container instances are brought up, they obtain IP addresses automatically and they discover other instances that they need to communicate with automatically. They also automatically inherit their configuration from the current version of the config document stored in the repository. In other words, scaling is a trivial operation. And an individual CNF may scale very large indeed: to tens of millions of subscribers if required.

It's perhaps worth pointing out that CNFs are typically composed from multiple microservices. Kubernetes monitors the load on each microservice and can scale each of them independently according to demand. This ensures optimum use of hardware resources at all times.

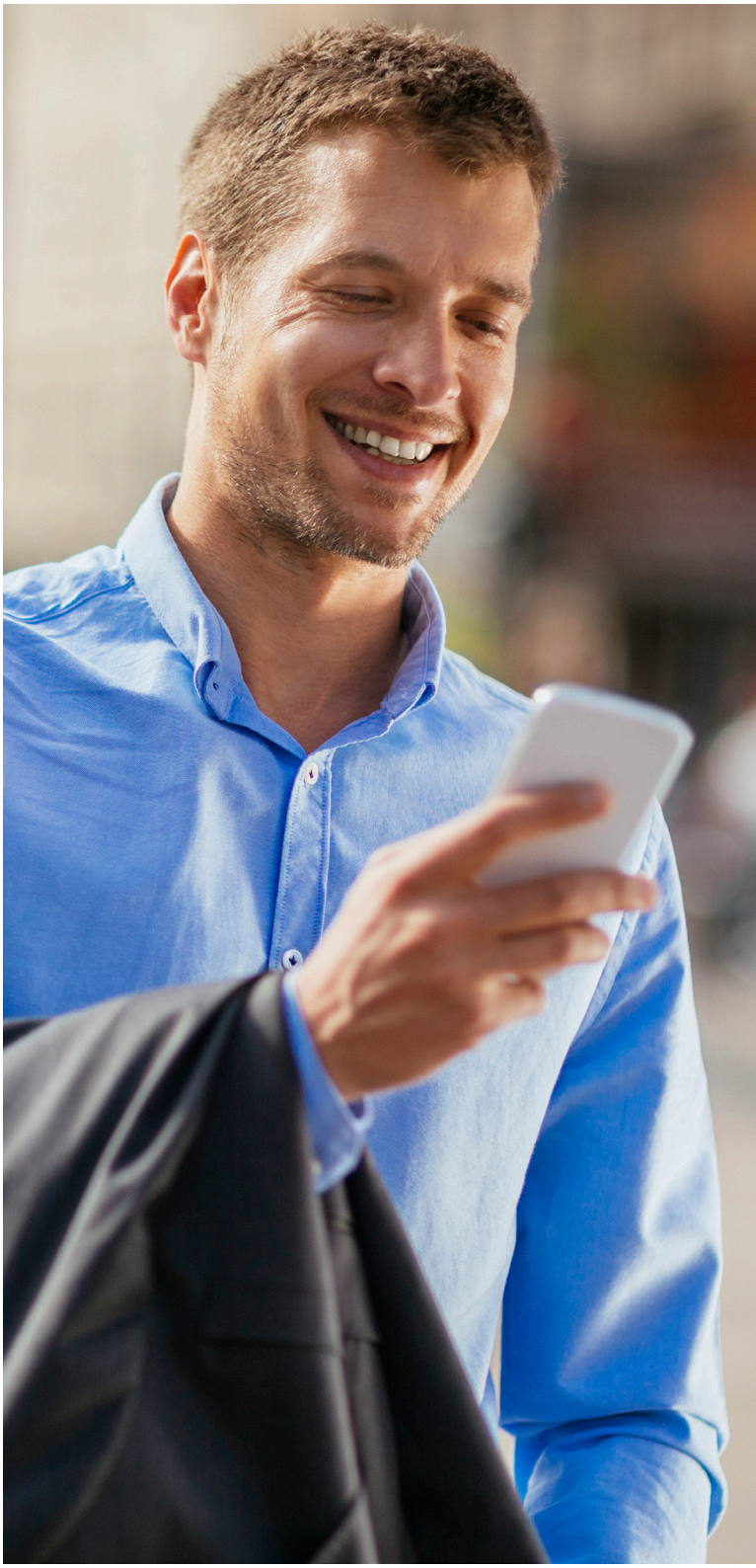
Software Upgrade

The process for applying software upgrades to CNFs is straightforward and invariably highly automated. In general, it involves progressively adding new container instances at the up-level software version to the "herd" while turning down instances that are running the old version. Even with massive systems serving millions of subscribers, this can usually be completed in a small number of hours and is a completely hands-off and non-ser-

vice-affecting operation. As with configuration changes, software upgrades can be canary tested by upgrading a small proportion of the running instances and monitoring KPIs to verify that the service is continuing to operate correctly, before rolling out the upgrade to the rest of the population. It is worth pointing out that CNFs are generally composed of multiple microservice components, and it is usual to apply software upgrades to one microservice at a time. The APIs exposed by microservices are always versioned and designed to be forward and backward compatible. This allows for major upgrades that affect multiple microservices to be applied in steps, one microservice at a time, without any disruption.

Because it is so easy to apply software upgrades to CNFs, it is common practice to implement agile principles and deploy frequent incremental improvements to the CNF software. The process of progressing software upgrades from development through automated testing and into production, so called "DevOps", can be automated to a very large extent. This can radically improve innovation velocity and reduce the burden of pre-upgrade system testing.

Upgrading VNF software is not nearly so straightforward. VNFs are typically based on large, monolithic software architectures with long release cycles. Each new release contains a great deal of new function and carries with it the risk of destabilizing the service provided by the VNF, so a long and comprehensive testing process is needed before rolling out. And applying software upgrades to VNFs usually requires complex procedures to be applied to one VNF instance at a time with multiple CLI-driven steps that can be very hard to automate. It can take many months to roll out a VNF software upgrade in a large network.



The Cloud Native Dilemma

It should be clear from the previous section that the positive impact of moving from VNF to CNF is far, far greater than moving from PNF to VNF.

Many network operators have understood this, and have made strong statements to the effect that they intend to move as quickly as possible to a cloud native approach to NFV. So why has there been so little progress to date in adopting cloud native practices in NFV?

The answer lies in two key realities:

- Real-world network functions are extremely complex, mainly because they have been evolving for many years in response to thousands of detailed technical requirements from network operators. A typical network function deployed in the network today includes multiple million lines of code, and has been in continuous development for 15 to 20 years.
- Cloud native software architecture differs so fundamentally from traditional software appliance architecture that it is not feasible, in practice, to re-factor existing software so as to embody cloud native principles. In other words, CNFs have to be developed from scratch, and cannot be evolved from legacy codebases.

Building a CNF from scratch to achieve full feature parity with the equivalent PNF or VNF requires a massive investment in both time and resources. Vendors who already have an equivalent VNF in their portfolio have little incentive to make this investment. None of their competitors has a CNF, so network operators have little choice but to deploy one of the available VNF products – no matter how strongly they may desire to pursue a cloud native NFV strategy.

Moving to cloud native is a discontinuity. There's no way around that. Network operators who are truly convinced that cloud native is the future need to plan for that discontinuity. This is going to be hard; incumbent vendors are promising an "evolutionary" path to cloud native, which sounds much less painful – but they aren't going to get there any time soon, if ever.

The almost universal recognition among network operators that cloud native is the right way to do NFV has persuaded all of the traditional telco equipment vendors to "cloud native wash" their VNF portfolio. Without exception, these vendors are claiming their products are cloud native today. But taking a VNF, built using software ported from a PNF, and packaging it in a container, does not make it a CNF. Nor does the simple addition of HTTP-based APIs to a VNF to support a service-based architecture make it a CNF. To re-iterate, the characteristics of a true CNF are as follows:

- A dynamically scalable, N+k redundant system based on a collection of loosely-coupled microservices.
- Packaged in containers and deployable without modification on any standard Containers-as-a-Service cloud infrastructure.
- Orchestrated by Kubernetes and leveraging key components of the cloud native software ecosystem including Helm (deployment), Prometheus (metrics collection) and Fluentd (log collection).
- Document-driven configuration management.
- Absence of Command Line Interface; all external interactions with the CNF are via open Web Services APIs.

Any network function product that does not exhibit these characteristics has no right to call itself a CNF, and will not deliver on the promise of cloud native that we described above. And the only way for a network function to embody all of these characteristics is for it to have been built, from the ground up, according to cloud native architectural principles. There's not a single commercially available network function on the market today that can claim to have done this, apart from Metaswitch's Clearwater IMS product.

Path to the Cloud Native Telco

We have to face the fact that the cloud native approach represents a clean break with the past. There is no incremental way to get there based on solutions being offered by incumbent vendors. So we have to find a way to embrace the inevitable discontinuity that will be involved. The obvious time to do this is during an upcoming investment cycle in new network technology. The 5G mobile packet core is an excellent example:

- The timing is good. Many network operators plan to put standalone 5G into production in the 2021-22 timeframe, at which point the cloud native software ecosystem will have matured nicely.
- The technology fit is good. The 3GPP standards for the 5G core define a service-based architecture, which is a pre-requisite for a cloud native approach.
- The vendor landscape looks promising. There are a handful of challenger vendors developing 5G core products, at least some of whom have some real understanding of cloud native.

The 5G mobile packet core is also a network function that truly cries out for a cloud native approach. In particular, the concept of network slicing requires the ability to automatically deploy instances of mobile packet core components rapidly and at widely varying levels of scale, in both core and edge clouds. This mission can only be achieved cost-effectively with a true cloud native approach. We envision the path to a true cloud native 5G core as follows.

Agile Procurement Process

The traditional procurement process based on RFPs is not a good way to set out on the path to cloud native. It tends to focus on vendor willingness to promise to meet a vast number of detailed functional requirements within a strict timescale at the lowest initial cost rather than focusing on what is really critical here: the ability of the vendor to deliver truly cloud native solutions cost-effectively and in a timely and agile manner. We advocate a process where a small number of qualified vendors are invited to participate in hands-on lab trials of their cloud native technologies. During this process, vendors who fail to demonstrate convincing cloud native capabilities are progressively eliminated.

Consulative, Agile Co-Development Process

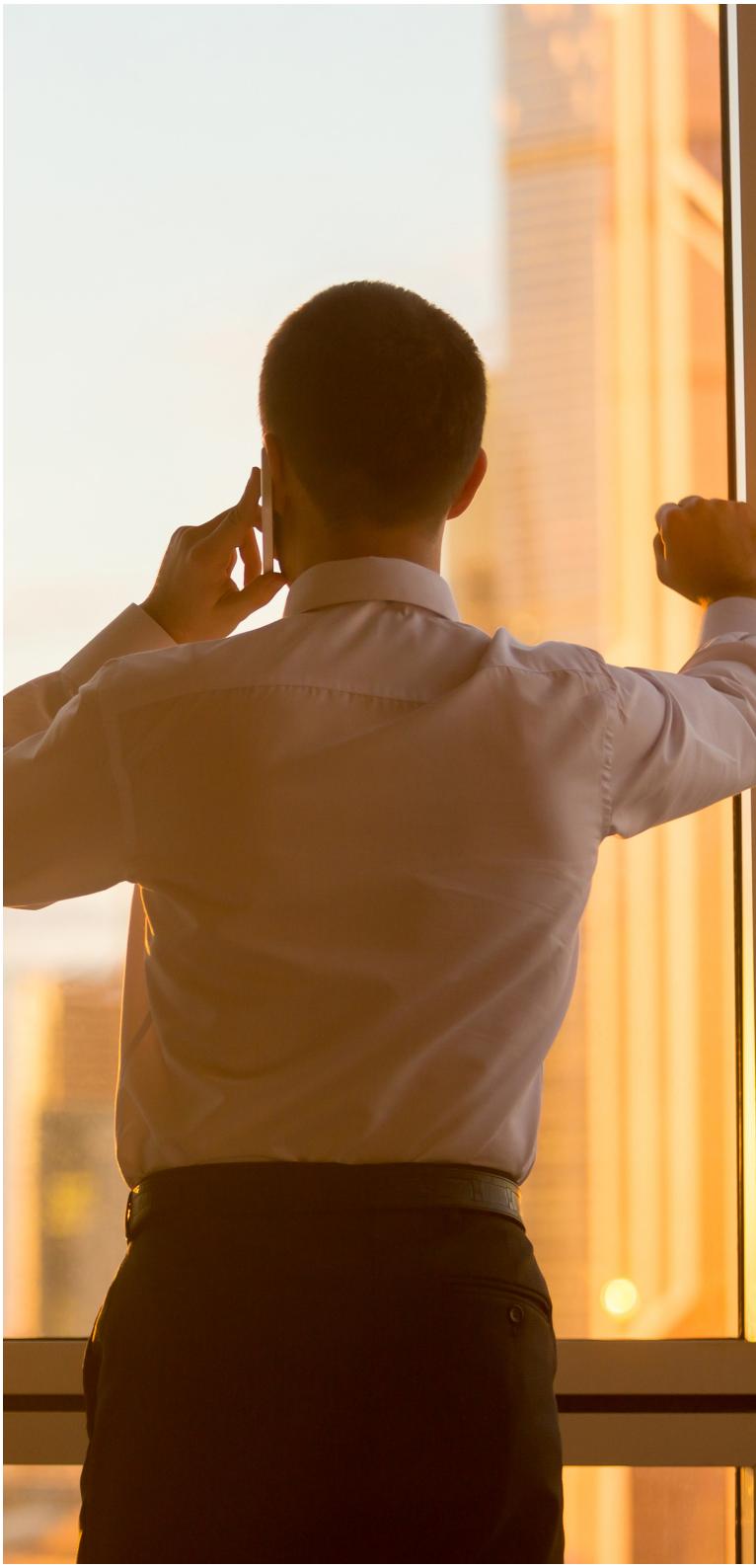
With planning input from the network operator, the selected vendor(s) would work according to agile development principles to deliver a series of incremental software releases into the network operator's labs, progressively fleshing out the solution to meet the operator's specific requirements. While this is progressing, vendors would work consultatively with the network operator to refine the cloud native infrastructure design and to flesh out the plan for operations management of the cloud native 5G core. During this process, the network operator would have an excellent opportunity to learn about cloud native and its operational practices at first hand. Vendors that fail to maintain an acceptable velocity of functional enhancements to their CNF would be eliminated from the process.

Fast Fail

The network operator's investment in any given vendor relationship during this process is strictly limited, at least during the early stages. If a relationship is not working to the benefit of both parties, it can be terminated with minimal damage. This is in contrast to the traditional procurement approach, where the network operator has so much invested in the relationship with the selected vendor that it cannot afford to fail, even when things are going badly wrong. In the case of 5G core, the network operator always has the option of falling back on the incumbent EPC vendor and their evolutionary solution for 5G core.

Minimum Viable Product

The biggest challenge in taking the direct path to cloud native is the need for the CNF to provide an acceptable level of functional capability to meet operational and service requirements for a given use case, and the time and investment required to get there – bearing in mind that the CNF is being developed from scratch. There is a tendency on the part of network operators to demand a vast number of functional capabilities to satisfy marginal needs, many of which can be characterized as "we've always done it this way." Network operators should be prepared to re-examine assumptions about the capabilities that are truly essential for any given use case, and be prepared to make compromises. In other words, "don't let the perfect be the enemy of the good."



Who Dares Wins

Unlike the simple virtualization of network functions in the form of software appliances deployed in virtual machines, the cloud native approach is truly transformative. Fully automated life-cycle management of network functions, instant deployment of 5G core network slices at any scale, management of configuration as code, portability across private and public cloud stacks, and DevOps-style continuous innovation will together bring stunning improvements in network operations efficiency and customer-facing responsiveness. That is the promise of the cloud native approach to NFV. Network operators who have the courage to embrace cloud native, and do so successfully, will surely emerge as the winners of the future.

But network operators should be under no illusions as to the true nature of cloud native, and the degree to which it represents a discontinuity. Those who prefer to stay stuck in the rut of comfortable incumbent vendor relationships will simply not get there. Only those who are prepared to grasp the nettle, face their fear of the unknown and embrace the discontinuity implied by cloud native will make it. They may actually be very pleasantly surprised along the way to find out how easy cloud native can be. Certainly very different. And much, much better.