WHITEPAPER

Cloud Native Thinking for Telecommunications

Basic tenets and end users' suggestions on how cloud native design and principles can be applied to mission critical telecommunications functions.

CLOUD NATIVE
COMPUTING FOUNDATION

CLOUD NATIVE
COMPUTING FOUNDATION

Table of Contents

1.1 Introduction

Enterprises, governments, and other organizations are rapidly adopting cloud native technologies to build infrastructure and run workloads that are more resilient, higher performance, and more economical. As cloud native is becoming the de facto choice for enterprise and IT, the telecommunications industry will also benefit from deploying cloud native architectures to its platforms, networks, and telephony applications.

This journey to cloud native telecom environaments has already begun. In fact, if you use voicemail on a wireless network powered by Bell Canada, your message queue is officially cloud native. In 2019, Bell Canada ported its voicemail network functions over to a container-based architecture running on top of a Kubernetes orchestration layer alongside its production instance of the ONAP architecture. Voicemail is one of a number of functions that Bell Canada has or will be migrating to Cloud Native Network Functions (CNFs). "The main driver behind cloud native for Bell Canada is service availability, namely, how can we leverage cloud native principles to make sure our services are more agile, more flexible, and more resilient?," explains Daniel Bernier, a Senior Technical Architect at Bell Canada and a member of Cloud Native Computing Foundation's (CNCF) Telecom User Group (TUG).

Telecommunications companies provide some of the most critical services in the world. From emergency 911 services to air traffic control to buy-sell signals that move billions of dollars worth of bonds, telecommunications networks impact our safety, social stability, and economic prosperity. Because the stakes are so high, telecommunications infrastructure has been designed, maintained, and upgraded to minimize risks and maximize reliability and uptime.

Operators like Bell Canada intend to move more of their functionality to cloud native architectures as the telecommunications community further evolves standards and practices around deployment of mission critical applications and services over CNFs. The goal of this whitepaper is to help companies like telecom operators, as well as enterprises running internal telecommunications-like infrastructure, to better understand what cloud native means for telcos and help build consensus around industry adoption of cloud native technologies.

1.2 What is Cloud Native?

After a collaborative discussion, the CNCF published the following definition of cloud native: "Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil."

1.3 Defining Cloud Native Systems

There are some additional key principles to cloud native that are important to grasp in order to think more broadly about architecting cloud native technologies for complex systems. A fundamental principle of cloud native for telco applications is that their underlying telco systems conform to the core precepts of cloud native architecture. The lack of exceptions makes the rule; the entire advantage of cloud native is driven by decomposability and abstraction from physical infrastructure. By deviating even slightly from these core principles, the service provider stands to lose the workload portability and minimal toil that drew them to cloud native approaches in the first place.

1.3.1 What "loosely coupled" means in cloud native systems

Cloud native systems have a clear separation between the microservices. The separation is based on technology, like containers, which allows independent life cycle management of each microservice. Cloud native applications (which are built on these microservices) and cloud native systems more generally benefit from design considerations that segregate services based on the business capabilities being delivered and the organization that owns said delivery. By leveraging technology such as containers, it is possible for individual business units to own the development of their specific microservice independent of collaborative development teams. Additional benefits of cloud native architectures include scalability on a per microservice basis. This simplifies deployment in a wide variety of infrastructure environments, ranging from an endpoint device to edge stations to large data centers. Adopting cloud native architectures also allows service providers to deploy resources in the most efficient manner possible while maintaining necessary and optimal service levels for regulated and unregulated service types.

The delivery of software as microservices harnesses the existing group boundaries within an organization and works with, not against, the different rates of change and business capabilities residing within those boundaries. This enables effective parallel development and communication across an organization, which allows events such as security patches to be rolled out more quickly and easily because deployment need not be executed in lock-step with other parts of the application.

While one microservice per container is the ideal state for cloud native applications, the reality is not always so clean and simple. It is entirely possible and sometimes required to run multiple microservices in a single container, and even to deploy very large containers for aggregations of services that are difficult or impossible to decouple due to software or network limitations. However, the general idea behind cloud native is to decompose microservices into the smallest possible infrastructure unit allowable. This causes a shift in where the burden of orchestration lies and pushes it solely (ideally) into the orchestration layer.

Cloud native applications also are packaged with all of their runtime dependencies during the build phase. The build is then used for deployment. This allows for individual business units and operations teams to care and maintain for their software with less risk of other stakeholders disrupting their service's "uptime". In practice, this allows for scenarios such as the owner of service "X" being able to theoretically scale their chosen software independent of service "Y".

These clear lines of demarcation provide a paradigm that makes sense to legacy operations groups who are being forced to rapidly assess how they've delegated roles and responsibilities. For example, a network focused operations group could do upgrades and updates to a microservice hosting routing protocol services, such as ISIS or BGP, without impacting the development or operations of the security team managing the microservices hosting AAA or Access List functionality.

1.3.2 Cloud native applications require orchestration

Because they are loosely coupled, cloud native services are always managed by an application orchestrator, such as Kubernetes. As cloud native applications are built from a loosely coupled collection of services that are delivered via an API, this coordination function is paramount, not only for reliability but also for performance and resilience. From an operational standpoint, an orchestrator is necessary to manage and deploy these services; manual deployment and meshing of these services and applications would be complex, error-prone, and resource intensive.

The orchestration layer is tasked with composing, decomposing, and reshaping or resizing the resources assigned to specific applications provided by the infrastructure. It does this based on assessments and monitoring of performance and availability of the actual infrastructure used by the application. In that manner, the applications are the "brains", directing the orchestration layer towards functional goals and outcomes (for example, low latency or specific uptime and resiliency metrics). The objective of orchestration is to create an entity that aggregates the microservices into services that can be easily mapped to network functions.

Kubernetes is unique in that it provides an example of both a declarative orchestration model as well as an application that can be deployed and consumed in a cloud native manner. From an orchestration standpoint, Kubernetes presents a declarative API and works off an assumption of immutability. Conversely, Kubernetes is an application that is comprised of a decoupled system of microservices deployed via orchestration. Given these design implementations, Kubernetes itself is often deployed via Kubernetes and treated the same as any other cloud native application. The expectations, set forth by this example, would be that a CNF would behave in a similar manner, with its core componentry being immutable and its method of configuration being declarative.

1.3.3 Infrastructure, Deployment, and Configuration of Cloud Native Systems

Immutable infrastructure (the orchestrator and all of the software and hardware that it depends on) is provisioned using baked and versioned templates (e.g. server images) or a combination of templates and bootstrapping (some repeatable and versioned process that is applied to the template e.g. kubeadm). The underlying infrastructure is not changed after it is made ready for use. New changes to the infrastructure are rolled out as new instances of infrastructure. By separating the application layer from the infrastructure layer, it is possible to iterate faster with minimal dependency risk. Additionally, this separation enables the vision of a programmable and software defined infrastructure, allowing end-to-end lifecycle management of the entire stack from the hardware up.

In cloud native systems, the atomic unit of deployment can come in many forms. A container, a physical appliance, a virtual machine or even future looking runtime methodologies such as unikernels and serverless. What truly makes an artifact cloud native is the philosophy employed when developing and deploying it. Container runtimes brought about a paradigm shift to the industry at large by commoditizing the complicated process of separating application dependencies from the infrastructure. A container, deployed upon an immutable and decoupled compute environment, attempts to abstract function from the underlying physical infrastructure and network architecture completely. At present, the most commonly used container environment is an OCI-compliant implementation.

A container is different than a Virtual Machine (VM), which was the initial base unit of cloud computing, in several key ways. A VM hosts a Guest OS capable of delivering a service's required compute and networking. This guest runs on top of a hypervisor consuming the virtual infrastructure and orchestration provided to it. It is also infinitely mutable, and continuously modified in real time making it unique and oftentimes no longer disposable. In contrast, containers run on top of a shared Host OS (a shared kernel) and leverages native Linux tooling to logically partition resources within said host. An application's desired runtime dependencies and metadata can be easily packaged within a container. As containers are agnostic to the type of Host OS they are running on, assuming basic kernel and libraries version requirements are met, the underlying infrastructure becomes transparent to the hosted application.

For Kubernetes specifically, the atomic unit of management is a Pod. A Pod is a set of application containers which are co-located and co-scheduled on the same machine (Node in Kubernetes terms) which run with a shared context and the same Linux networking namespace. For telco (and other) cloud native applications, the Pod is the basic building block of deployment, scaling operations and upgrades.

Cloud native applications consist of sets of cloud native microservices. The configuration of these microservices is not changed after deployment (note, this does allow for changes in application data such as "add a new user"). New features for a cloud native application and the underlying Pods are rolled out as new artifacts and new configuration. Cloud native systems are configured declaratively. This means that the system configuration describes "what" a loosely coupled system should look like, not "how" the system should be created. The "how" of the application is determined by the tooling (e.g. the orchestrator, operators, and CRDs).

1.4 Cloud Native Network Functions

A cloud native network function (CNF) is a cloud native application that implements network functionality. A CNF consists of one or more microservices and has been developed using Cloud Native Principles including immutable infrastructure, declarative APIs, and a "repeatable deployment process".

An example of a simple CNF is a packet filter that implements a single piece of network functionality as a microservice. A firewall is an example of a CNF which may be composed of more than one microservice (i.e. encryption, decryption, access lists, packet inspection, etc.).

The CNFs themselves, by nature of following cloud native principles, are also composable and can implement and facilitate more complex network functionality if needed.

CNFs and cloud native principles can be used when implementing applications and network infrastructure which needs to meet standards such as 3GPP. For example, an Evolved Packet Core's (EPC) Serving Gateway could be implemented as a CNF and support all 3GPP requirements (eg. S1 protocol stack) allowing integration with other EPC services.

1.5 Cloud Native for Telcos

1.5.1 A Brief History of Virtualisation for Telcos

Until the 1990s, most functionality and applications for telecommunications resided primarily in physical hardware (aka 'boxes'). While software powered these systems, the software functioned more like firmware and was not easy to abstract and run on multiple platforms in virtual environments.

This fostered a "top-down" approach to telecommunications systems that was imperative rather than declarative. This approach was resource intensive and expensive; redundancy meant acquiring multiple physical instances of required systems. This guiding architectural decision encouraged the tight bundling of numerous services and capabilities into individual physical systems. The result was not only high infrastructure and provisioning costs but also inefficient hyper-redundancy in system engineering for overlapping capabilities.

1.5.2 Software Defined Networking And The Emergence of VNFs

In the late 1990s, the concept of virtualization for enterprise computing garnered interest as enterprises tired of spending large sums of money on high-powered servers. Simultaneously, we saw a rapid rise in the popularity of the commodity Open Source Linux Operating System.

In 1999, VMware released its first virtualization product for x86 devices. In the early 2010s, the concept of Network Functions Virtualisation (NFV) emerged as telcos and other users of big networking gear sought to virtualize their networking functionality, enhance resiliency, and take advantage of commodity hardware. With constructs such as NFV-encapsulated Virtual Network Functions (VNFs), Network Function Infrastructure (NFVI), and an orchestration component (MANO), the entire NFV hardware and software stack could be integrated and distributed across geographically dispersed data centers. In 2017, Open Networking Automation Platform (ONAP) emerged as the open source standard platform for orchestrating and automating physical and virtual network elements with full lifecycle management. NFV and ONAP primarily focused on creating and orchestrating virtual analogs for physical equipment – virtual boxes.

This was the same pattern followed during server virtualization when initial efforts focused on replicating physical servers as virtual entities. Cloud native, however, requires a more fundamental rethink and redesign of how telco and backbone/core network systems function. This paradigm shift impacts both the software architectures being proposed and how the telco operator structures and organizes its staff.

In a sense, cloud native for telcos will require a transformation that is actually not dissimilar from the era of forklifting over monolithic legacy applications into virtual environments. This next evolution will require the decoupling of tightly bundled functions and processes to create a more resilient, manageable, and performant infrastructure that derives immutability precisely from its ephemerality. For these considerations to successfully achieve widespread adoption, CNFs will have to first be introduced alongside VNFs and PNFs (physical network functions) and will need to fit into operational models that govern those earlier types of

network functions. Finding sensible integration points between emerging greenfields and legacy brownfields will be crucial to the success and adoption of cloud native practices at larger telcos. CTO offices and architecture groups must find ways to stitch CNFs into existing layer 2 and 3 domains, integrate CNFs into existing management systems currently present in the service providers' networks, and find ways to introduce some of modern DevOps practices, such as continuous integration (CI) and continous deployment/delivery (CD), to the PNF/VNF world. These achievable objectives are crucial as they provide executives both a means of capturing a return on investment with their existing infrastructure and a realistic path forward in adopting a more holistic cloud native approach.

1.5.3 Principles for Cloud Native Telco Infrastructure and Applications

At its core, the evolution from box-centric to process and function-centric requires an evaluation of the OSI Layers through a cloud native lens. The best general practices of cloud native infrastructure deployment and application design should also be applied to networking infrastructure. Two key shifts required for CNFs to function well are deployment pipelines and the use of declarative programming and scripts for configuration. By pipelines, we mean the use of CI and CD platforms. This is fundamental because it allows for rapid iteration, granular versioning, easier rollbacks, canary testing, and many other capabilities that are presently complex and often not possible when deploying VNFs. By declarative language, we mean a shift to intelligent orchestration that is better able to maintain required service levels and maximize efficiencies based on available resources. This will ultimately mean a transition of network intelligence and design – function, description, and configuration – into other parts of Kubernetes (Helm Charts, CRDs, and Operators).

1.5.4 Evolving the Stack From VNFs to CNFs

The emergence of Kubernetes as the most widely used container orchestration layer allows telecommunications operators to shift their infrastructure over from more bespoke, expensive, and higher friction stacks to a de facto standard platform while also improving performance and resiliency. The extensibility of Kubernetes should allow a gradual transition from legacy VNFs built on OpenStack or VMWare VMs over to a Kubernetes telco stack that accommodates all aspects of their business - including legacy VNFs, which can be easily managed via an abstraction layer on top of Kubernetes (KubeVirt/Virtlet/Openstack).

Back-office applications, such as BSS and OSS functions, can run directly on Kubernetes and will enjoy all the same benefits that enterprise applications derive from Kubernetes. Operators should also be able to run real-time, critical functions on Kubernetes. CNFs must be able to run on Kubernetes and benefit directly and strongly from cloud native principles. It is important to note that, in theory, Kubernetes and containerization should allow for true infrastructure-agnostic immutability and portability. This will only be true if conforming Kubernetes versions and configurations are used and supported by operators. That said, creating a Kubernetes conformance model for telecommunications is non-trivial, requiring validation and certification of key elements such as network software (CNI plug-in, network service mesh, etc.) and other aspects.

1.6 Cloud Native For Telcos In Practice

Likely deployment patterns for cloud native applications into telcos will move from functions that are more fault intolerant or not as latency sensitive (voicemail, OSS/BSS, etc.) to more critical functions such as the backbone cores. Telcos will gain on multiple fronts by deploying cloud native systems. Those benefits include operational consistency, application resilience, simplified and responsive scaling at the microservice level, simplified integration with enterprise-facing cloud native applications, and improved portability between public, private, and hybrid cloud environments. Another key improvement would be transparency, observability, and control, all of which would be increased by operating network functions under a Kubernetes umbrella as opposed to operating purpose-built boxes. Telco operators should also benefit from built-in or natively integrated tools and metrics, such as Prometheus, Jaeger, and OpenTracing because they are simpler to instrument and observe at the service level in Kubernetes running microservices.

As cloud native involves decomposing tightly coupled processes into loosely coupled microservices, rearchitecting applications as CNFs often will entail decomposing functional aspects into services running in discrete containers or as separate microservices. For an example of how cloud native telecommunications principles might work in practice, consider a UTM (universal threat management) firewall. These are generally large physical or virtual appliances that encompass multiple functions including DDoS protection, IP filtering, VPN provisioning and termination, stateful packet inspection, and anti-virus / spam filtering.

They are complicated to troubleshoot and a clear single point of failure for multiple critical functions. UTMs are also by design imperative. This makes them brittle and poor handlers of edge cases. UTMs cannot be easily managed via CI/CD and standard DevOps practices.

Through a cloud native lens, all of the functions of the UTM could be decomposed as microservices, each functioning as its own application running in it is own container (or containers). These services could be controlled and monitored from a single management plane, giving the operator the same visibility into what is happening and the same ability to manage the composed whole, but offering additional granularity on service performance. This should also allow for easier bursting to handle outlier events (usage/traffic spikes) and for running security infrastructure as an agile pipeline rather than a cumbersome snake. Additionally, if one microservice fails, other services in the meshed CNFs would remain functional, resulting in improved overall resiliency and failure tolerance. For example, a failing DDoS microservice would not impact IP filtering, anti-virus, or other functionality in the UTM application bundle. Similarly, this architecture will support the cloud native vision of network services running on common infrastructure that can scale and enhance performance and reliability in the most optimal manner while enabling an operational environment to address DevOps models in the most adequate manner.

The manipulation of namespaces and multi-tenancy within the orchestration layer will be crucial for operators if they are to deploy these examples at scale. For there to be any hope of gaining a return on investment, smarter deployments of the infrastructure will be as important as the deployment of the CNF itself. Building upon the UTM example, if it and subsequent CNFs adhere to the cloud native design philosophies, then it should be feasible to deploy them into a single Kubernetes cluster, leveraging the available tooling to ensure logical isolation/segmentation.

If the CNF has strong opinions on what the infrastructure must provide (i.e. "CNF-X will only run in a 3rd-party-provided Kubernetes deployment"), then the economies of scale in both infrastructure costs and operational simplicity are reduced. For example, if an operator must run multiple types of Kubernetes for different CNFs, then the infrastructure and required orchestration software is no longer a commodity. In this scenario, each CNF suite may require more masters and instances of the etcd key-value datastore than is economical, to name one example. Additionally, the different flavors of Kubernetes will require different operational playbooks and potentially will inject more complexity into analytics, application performance monitoring, and infrastructure management. Looking at a service provider's far edge, it is not feasible to assume that a cupboard-sized data center will be cost effective if the expectation is for that miniature data-center to host an instance of Kubernetes per application. A conforming "Plain Vanilla" flavor of Kubernetes for telcos would ideally run on all different infrastructure components, including X86, ARM, and RISC, due to the different infrastructure composition of centralized data centers and edge data centers.

1.7 Conclusion

Telecommunications services demand a higher bar for resiliency, security, and performance than nearly any other set of services. Our society increasingly relies on our telecommunications infrastructure for critical capabilities that keep us safe and allow us to conduct research, commerce, and our daily lives.

This high bar was attained at a high price. Traditionally, telecommunications infrastructure and applications have been designed in tightly composed physical or virtual units in order to better control the software stack and with a model of redundancy and resiliency derived from ready overcapacity. This has also ensured that deployment of new applications is complicated and often slow; tightly-bundled applications are not easy to modify and cannot be provisioned or versioned with modern DevOps tools. As a result, telecommunications progress was usually quite expensive, both in implementation and physical infrastructure. The reliance on custom-made telecommunications platforms meant it was necessary to retain and rely on architects and consultants with arcane expertise in those bespoke platforms.

The great promise of cloud native for telecommunications is to turbocharge the development and improvement of telecommunications applications by finally allowing these critical services to run on largely commodity physical infrastructure, and to finally decouple them from purpose-built boxes. In addition, customized, tightly-coupled, and hardware-bound telecommunications platforms can give way to Kubernetes, enhancing modularity, scalability, observability, and interoperability. This transition will enable the telecommunications industry to benefit from the dynamic and vigorous community of software development and innovation rapidly growing around Kubernetes. In general, adopting cloud-native ways should help telecommunications operators become more agile, responsive, and adaptive both internally and externally. These are already key goals for the telecommunications industry. Moreover, the telecommunications industry stands to benefit even more than enterprises because the implications for reducing CapEx and improving the speed of application development (and by extension, service improvement) are profound.

Telcos have traditionally struggled to keep pace with startup technology companies in deploying newer software-driven technologies because they played by a different set of rules - the “Rules of the Boxes” and strict regulatory rules - defining service delivery and reliability. In the cloud native realm, boxes can be broken up into distributed CNFs composed of microservices that follow cloud native principles.

As a result, telcos will be able to freely test, design, deploy, provision, revise, and ultimately innovate at a far faster pace. With process and organization adaptations to cloud native thinking and design, telcos will be able to leverage this new freedom to build more resilient infrastructure and applications and to accelerate the roll out of new features and new capabilities more quickly, with less risk and less fear of disrupting critical regulated systems and functions.

Work is ongoing to bring the most demanding and mission critical telecommunications systems to cloud native, including those that must fulfill regulatory requirements (such as emergency call handling). However, in theory, CNFs running on Kubernetes should provide equivalent regulatory compliance for the same reasons that cloud native applications for enterprises tend to be more fault-tolerant, more responsive, and more resilient; because loosely coupled immutable systems are inherently better at handling highly complex applications with strict SLAs and multiple dependent and interactive functions. This is the significant promise of cloud native for telecommunications applications and functions.

How to Get Involved

This whitepaper was developed by the CNCF Telcom User Group. The group meets monthly to discuss gaps in the industry as it moves towards more cloud native ways of thinking and working.

The CNCF also hosts the CNF (Cloud Native Network Function) Working Group. It will build and iterative upon the initial ideas and work presented in this whitepaper. I It's mission is to increase interoperability and standardization of cloud native workloads in the telco space. It will provide an open source test suite to demonstrate implementation of cloud native best practices for both open and closed source CNFs. The WG oversees and maintains what it means to be a cloud native conformant CNF. It also develops the process and policy around the certification program. Work on the mechanics of the conformance tests occurs in the CNF Conformance Test Suite. Both groups are open for anyone to join and contribute to.

Contributors: Ahmed ElSawaf, Bill Mulligan, Dan Kohn, Daniel Bernier, Dave Cremins, Ervins Kampans, Frederick Kautz, Gergely Csatari, Henrik Saveedra Persson, Herbert Damker, Ildiko Vancsa, Jeff Saelens, Khawaja Daniyal, Lei Wang, Rabi Abdel, Saad Sheikh, Tamas Zsiros, Taylor Carpenter, Tom Kivlin, W. Watson, Petar Torre. Editors: Alex Salkever


CLOUD NATIVE
COMPUTING FOUNDATION