WHITE PAPER

Communications Service Providers
Cloud-Native Network Functions

Why Use Containers and Cloud-Native Functions Anyway?

Learn how to correctly implement cloud-native network functions for 5G

Authors

Muthurajan Jayakumar (M Jay)
Cloud-Native Solution Architect & Platform Software Engineer, Network Product Group, Intel

Table of Contents

Executive Overview

The telecommunications industry is at an inflection point. Communications service providers (CoSPs) need to adopt new strategies to successfully monetize their networks as 5G deployments transition from nebulous plans to real-world projects. The traditional CoSP approach, with monolithic virtualized network functions (VNFs) that take years to deploy or upgrade cannot keep pace with the new 5G landscape. The nature of the 5G core network—dynamic, configurable and agile—requires a cloud-native approach that uses web-scale, containerized network functions (CNFs) that are resilient, decomposed into microservices and, as much as possible, open source.

As 5G use cases multiply, experts predict the number of services will quadruple those delivered during the last 40 years[1]. Without making a smooth and successful transition to a cloud-native 5G core network, CoSPs risk losing customers to other players, such as cloud service providers.

Intel is working with the telecommunications industry to deliver software-defined networking (SDN) and automation powered by Intel® hardware and software. The Intel® Network Builders is an ecosystem of independent software vendors (ISVs), operating system vendors (OSVs), original equipment manufacturers (OEMs), telecom equipment manufacturers (TEMs), system integrators and CoSPs coming together to ease and accelerate the adoption of 5G.

To be competitive in the fast-moving 5G marketplace, CoSPs need to get cloud-native right the first time; there is no room for failure or hesitation. This white paper provides an introduction to the synergy between 5G, microservices and containers, followed by a discussion of cloud-native architecture design principles and best practices for building cloud-native CNFs.

Introduction

CoSPs will be able to deploy services faster and more flexibly if they can copy the cloud-native architecture behind giant cloud companies, and use it to underpin their virtualized networks. These web-scale benefits, generated by a cloud-native approach to cloud services, include independent and fast deployments, programmability, elasticity and minimized interdependencies among services.

Why CoSPs should go cloud native

Cloud native is an approach to building and running applications that fully exploits the benefits of the cloud computing model. When successfully implemented, this transformation can bring unprecedented speed, agility and resilience in service development and management processes. Cloud-native characteristics include microservices, containers, continuous delivery and dynamic cloud-based management.

In the telecommunications context, cloud native brings tangible benefits in terms of reduced capital expenditure (CapEx) and operational expenditure (OpEx), faster time-to-market service development and deployment, and scalability in both east-west and north-south traffic. These benefits become critical business requirements as the use cases for 5G expand, both for massive machine-type communications and ultra-reliable, low-latency communications. Examples of use cases include smart cities, homes and buildings, voice, 3D video, augmented reality (AR), virtual reality (VR), industry automation, autonomous cars, mission-critical applications, and many more use cases we've not yet even imagined.

For example, who knew, when defining the 4G Long-Term Evolution (LTE) standard, that ridesharing applications like Uber and Lyft would become a dominant usage model? In the same way, as CoSPs, hardware companies and ISVs develop new technologies, the 5G landscape may yet expand to unimaginable usage models.

A primer on containers and microservices

To be cloud native, it does not matter whether applications are deployed in a public, private or hybrid cloud environment. What matters is *how* they are deployed. VNFs that are not cloud-native are characterized by a single core network, dedicated protocols, and are hosted on virtual machines (VMs). Cloud-native network functions are inherently different in design and take advantage of containers.

Containers are a method of virtualization that bundles an application with all its dependencies—required executables, binaries, libraries and configuration files, for example—into a single, isolated package. In a VM, virtualization abstracts the hardware and runs multiple OS instances. In contrast, containers abstract the OS, multiple containers on a server share a common OS kernel. In addition to containers, CNFs use orchestration services that automate service functionality.

CNFs also enable the use of microservices, which are services that work together as a distributed system. Microservices have three primary characteristics:

- **Small**–a microservice focuses on doing one thing well.
- **Focused**–microservice boundaries are focused on business boundaries, making it obvious where code resides for a given piece of functionality.
- **Autonomous**–the resulting simplicity makes the distributed system much easier to decouple for web scale. For example, you can scale the control and data planes separately.

As an example of decomposing a function into microservices, consider the common tasks of parsing packet headers, inspecting the payload, and classifying packets. These tasks are performed by multiple network functions. By containerizing these tasks as microservices, much redundancy is eliminated and a newly defined network function can simply call the microservices as needed.

There is no one-size-fits-all answer to whether each separate microservice is placed in its own container, or several microservices are combined in a single container. If the goal is to reduce inter-microservice communication overhead, then using one container for multiple microservices may be best. On the other hand, if the continuous integration/continuous deployment (CI/CD) benefits outweigh the inter-communication overhead, then each microservice should have its own container. Another reason to possibly use individual containers is if the microservices are supplied by different vendors.

5G and microservices offer many benefits to CoSPs

5G is providing new experiences, such as AR and VR to traditional consumers. But the mobile cellular industry is also pushing into multiple new verticals with distinctive service categories, such as future factories, eHealth, automotive, mission-critical computing and immersive media, with the expectation of addressing these markets in the next three years or so.

So, how will the telco industry, which has essentially created and matured three mobile services-voice, short messaging, and internet connectivity-over the past thirty years, deal with the staggering challenge of developing multiple markets in a few short years?

Microservices offer the benefit of distributed systems and service-based architecture. They are appealing due to their small footprint and fast start times. The primary benefits of microservices, for 5G and in general, include the following:

- **Update the network quickly with less risk.** A one-line change to a monolithic application consisting of a few million lines of code requires the entire application to be re-deployed to release the change. This could have a negative impact on business continuity and poses a high risk. In contrast, microservices' fine granularity allows individual services to be upgraded with minimal or no impact to other services. With microservices, the change is made to a single service and that service is deployed independently of the rest of the system. This allows CoSPs to deploy the code faster. And, if a problem occurs, it can be isolated quickly to an individual service, making fast rollback easy to achieve.

- **Deliver bug fixes and new functionality faster.** Ease of deployment translates to the ability to install bug fixes and deliver new functionality to customers faster using the Agile CI/CD methodology. CI/CD enables CoSPs to deliver new functionality quickly and with low risk.
- **Enable cost savings with independent scaling.** With a large, monolithic service, everything must be scaled together. For example, if you want to scale the data plane, you also have to scale the control plane. But with microservices, it is possible to scale only the services that need scaling[2]. This allows CoSPs to run other parts of the system on smaller, less powerful hardware, resulting in overall cost savings.
- **Build greater resilience.** In a monolithic application, if the service fails, everything stops working. With microservices, CoSPs can build systems that handle the total failure of services and degrade functionality gracefully. For example, if there is a memory leak in one microservice, then only that microservice is affected. The other microservices that comprise the distributed system will continue to handle requests. See "Manage engineering trade-offs with the CAP theorem" for more information.

To achieve the scalability, flexibility and performance necessary to cost-effectively deliver 5G services, a cloud-native framework should integrate all CNFs. This enables CoSPs to create, contract for, or require as a condition of use, a standardized "adapter" that exposes all control, parametric, and management APIs and data in a common way. With 4G, a cloud-native core architecture is becoming the preferred option, but with 5G, a cloud-native design is a must-have if CoSPs are to compete with innovative new services. Only by redesigning the software architecture and core functions using cloud-native design principles and IT web-based development and methodologies, can CoSPs gain the necessary agility to rapidly deliver new services and reduce their time-to-market.

Beyond the general benefits of microservices discussed above, the combination of cloud-native and microservices provides the following specific advantages in a 5G network:

- **Modularity and reusability**. Cloud-native microservices support key 5G features such as network slicing. Network slicing is similar to the concept of VMs: a single common physical network is divided into several logical networks. Each network "slice" can be dedicated to a particular type of traffic or use case. For example, with the right orchestration tools, one network slice could be used by CoSPs to deploy and manage 5G low-latency services in an edge cloud, while another slice could run 5G standard services in the core cloud. Microservices are critical for the success of network slicing because they provide the required agility, cost-effectiveness, and ease of use. One microservice can be easily invoked by other services (with appropriate authorization), in different network slices, enabling cost-effective reuse of each service. Since there will be a wide variety of 5G applications with differing bandwidth, latency and connectivity requirements, microservices let applications mix and match capabilities without having to wait for slow and cumbersome VMs to spin up. In the case of ephemeral IoT services this would be completely unsuccessful.
- **Versatile orchestration.** Open source tools, such as Cloudify-an open source cloud orchestration framework-support container-as-service (CaaS) across multiple clouds, both edge and core.
- **Openness.** Together with some management and control functions, such as authentication, authorization, and accounting, the information about a 5G network can be easily exposed to external users such as third parties through open APIs without complicated protocol conversion.
- **Flexibility.** 5G networks involve far more use cases at the edge than previous network generations. The agility provided by cloud-native microservices lets CoSPs dynamically manage edge workloads.

Other benefits that a cloud-native architecture based on microservices brings to 5G include stateless and state-ful services, a common and shared data layer, abstracted infrastructure, and streamed telemetry data. But to achieve these benefits, there is one simple truth: traditional, relatively slow IP routing-like networking in the data center cannot cope with a cloud-native system that supports service discovery and that creates, moves and stops microservice instances in real time. Therefore, CoSPs must learn the basic design principles for building a cloud-native architecture. These are discussed in the next section.

CoSPs who take a bold step to embrace cloud-native functions can expect to enjoy flexibility, web-scale benefits and additional operational efficiencies, as well as taking a business-strategic lead in the industry.

Microservices support 5G core network architecture evolution

5G requirements range from enhanced mobile broadband to ultra-reliable, low-latency communications to massive machine-type communications. Various 5G applications will mix and match these characteristics, requiring a high degree of programmability and the ability to easily combine different functionalities. Microservices support the evolution of the 5G core network architecture by providing this programmability and flexibility through the following:

- The ability to scale and deploy functions quickly
- Orchestration that delivers automated lifecycle management through CI/CD
- The ability to manage discrete functions in independent units

Synergy between 5G and cloud native

The 3GPP Technical Specification Group on Service and System Aspects (SA) has several subgroups. Recently, under SA2 (Architecture), 3GPP proposed a service-based architecture of the 5G core system. The architecture is composed of CNFs and reference points that connect CNFs. For a full discussion of the specification, such as definitions of the 5G core network functions, visit https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/15.02.00_60/ts_123501v150200p.pdf.

Network functions interact with one another using predefined interfaces. All these interfaces are candidates to be REST APIs and, in this sense, the architecture reference points can be simply replaced by a "message bus" in a logical diagram. As shown in Figure 1, the 5G core network uses a service-based architecture (SBA), centered on services that can register themselves and subscribe to other services.

With the direction taken by 3GPP, the architectural intent is that 5G core CNFs become cloud ready. The 5G core platform will be more programmable and will allow many different functions to be built, configured, connected and deployed at web scale. This would be impossible with traditional network function virtualization (NFV), which uses monolithic VNFs that are inflexible and hard to scale. In contrast, cloud-native microservices are far more flexible and scalable.

Intel has prepared a Cloud-Native Transition Playbook that can help CoSPs take advantage of the synergy between 5G and cloud-native technologies. (This playbook is available only under a non-disclosure agreement [NDA]. Contact your Intel representative for more details.)

Kata Containers Helps Boost Security

Kata Containers (which includes the project formerly named Intel® Clear Containers) is an Open Container Initiative (OCI)-compatible container runtime. It enhances the security of container workloads in lightweight virutal machines (VMs), each with its own lightweight OS and dedicated kernel, essentially offering the same security as a VM. Kata Containers provides the compatibility layers to use a hypervisor as a Kubernetes- and Docker-compatible container runtime isolation layer.

Kata Containers is a flexible solution with the following attributes:

- Works seamlessly with Kubernetes and Docker
- Is open source
- Offers open governance under the OpenStack Foundation umbrella
- Supports multiple architectures
- Supports multiple hypervisors: QEMU, KVM, Firecracker, and other hypervisors under development

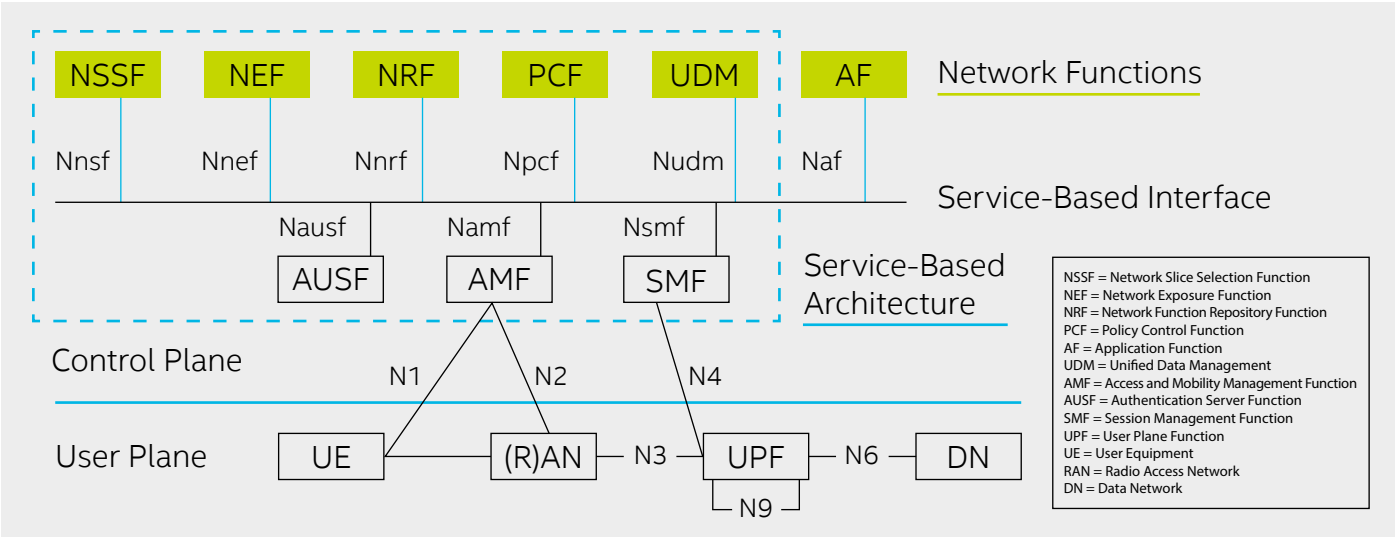

Figure 1. With the proposed 3GPP architecture, network functions become cloud ready, with a programmable 5G core platform and support for web-scale deployment.[3]

Cloud-native architecture and design principles

The 3GPP proposal for service-based architecture for 5G functionality is architected with software decomposition, designed as microservices. By way of example, consider two of the network functions shown in Figure 1: The Session Management Function (SMF) and the User Plane Function (UPF). The following sections explore two examples, comparing the case where cloud-native architecture and design principles are followed versus the scenario when the functionality is implemented as a VM-based monolithic application (not cloud native). These comparisons show exactly how cloud-native design benefits 5G networks.

Canary testing

When a new version of a service is pushed into production and is tested by a small number of end users, it's called "canary testing." Canary testing is used to verify the new functionality works as intended and does not cause problems with other services. To perform canary testing, the network must provide independent lifecycle management per microservice. For example, canary testing in the SMF is enabled by running two versions of the SMF simultaneously while the UPF stays the same.

- **Problem with a VM-based VNF.** If both the SMF and UPF are implemented in a single monolith (i.e. not decomposed) application, even a single line change in the SMF leads to the full monolith being retested and released, because of the dependency between the SMF and UPF. Typically, CoSPs are reluctant to undertake such a heavy-weight, risky process for just a few lines of code; Therefore, they delay the release process until more changes are needed. But, when even more changes occur between releases in a monolith, the risk factor of something breaking increases. Thus, the monolith approach lacks the possibility of canary testing, independent life cycle management, and the agility of CI/CD. It is almost like a forklift-type upgrade, where you have to switch off the unit, stop all the traffic that was going to that unit during upgrade and then switch back the traffic.
- **Solution and benefit of a cloud-native CNF.** In the microservice architecture and cloud-native development environment, CoSPs have the option of performing canary testing, which enables updates of software without having a maintenance window. Instead, the CoSP can make in-service changes that do not depend on a clustering mechanism but are performed at the application level. This means that the CoSP can run different versions of software in the cloud at the same time. For example, as shown in Figure 2, a new version of the SMF microservice can be deployed as a canary test that solicits feedback from a small number of focused expert developers and users. The CoSP can continue to test and improve the new version and fix bugs in the real production environment. Once the performance of the new version is satisfactory, it can be scaled out and the old version can be removed—all without affecting other microservices.

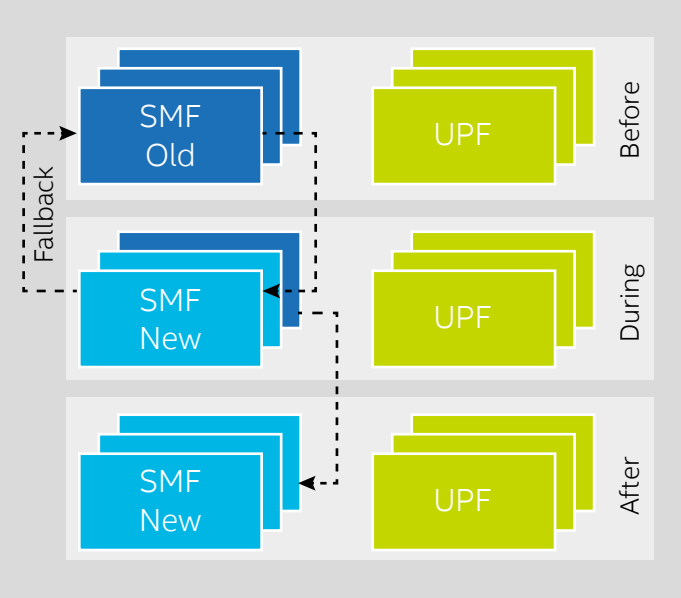

Figure 2. Cloud-native design enables canary testing, where CoSPs can run two versions of software simultaneously.

Resilience and service availability

When a service instance outage occurs, it would be ideal to gracefully degrade the service instead of completely losing service.

- **Problem with a VM-based VNF.** In the case of a monolith VM-based design, resilience is digital—it is either 1 or 0 because you must basically restart your application to get back to service. In this scenario, servers are often referred to as "pets"—if one fails there is emotional distress, and they are treated as indispensable[4]. Also, historically monolith VNF resilience and service availability has been measured by mean time between failure (MTBF) for specific hardware. But that metric is not useful as a baseline in a cloud-native environment, because a cloud-native CNF doesn't necessarily run on the same hardware—microservices that make up the CNF can be hosted on multiple machines, and can even move from machine to machine based on network demand and instance failures.
- **Solution and benefit of cloud-native CNF** In cloud-native deployments, resilience is on a gradient scale. Servers are often referred to as "cattle," where there are lots of them and if one fails, there's plenty more where that one came from. In the cloud-native scenario, the relevant resilience metric becomes MTBF not for the hardware (server) itself, but for the run-time instance—which could be a container on top of a VM which is in turn on top of hardware. In short, since cloud-native CNFs are not tied to specific hardware, MTBF becomes irrelevant. What's more, a cloud-native architecture takes into account that microservices are designed for multiple failures (see Figure 3).

As long as one microservice instance of a particular kind is still available, the service doesn't fail completely but instead simply loses top-line capacity. Cloud-native design supports any combination of failure at any time without escalating to a full restart of the application—the service simply gracefully degrades while the CoSP works to restore the capacity (which the cloud can handle in a very efficient way). It is highly unlikely that all the "cattle" servers will fail, and it would take quite some time for that to happen. While the combination of MTBF of all instances is tricky, it isn't so important in a cloud-native environment due to the ability to automatically recover after service failure.

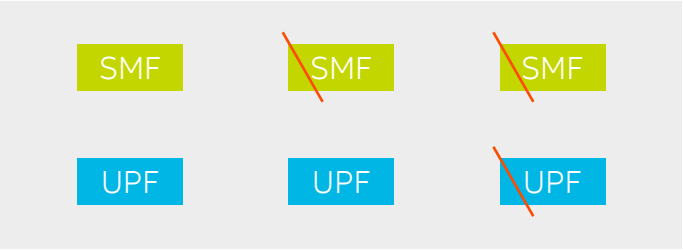

Figure 3. Cloud-native design accommodates multiple service failures without full loss of service or application restart.

Automatic recovery after service failure

As discussed in the previous section, a cloud-native design changes the game from "getting the service back" to "getting back to full capacity." It is CaaS monitoring that enables this.

In Figure 4, you can see one of the two instances of microservice A has failed. But there is still another instance of the same microservice running, so service is not lost entirely. The CaaS manager, such as Kubernetes, uses automated procedures and policies to recognize that one instance has failed and will deploy a new instance in a new container. When this process is complete, microservice A is back to full capacity thanks to auto healing.

Considerations for the journey to cloud-native

There are several things for CoSPs to consider as they transition from monolithic services to cloud-native microservices. These include tool choice and tool evolution and the relationship between containerization technology and cloud-native technology. Most importantly, the journey to cloud-native doesn't have to be lonely—Intel is committed to working with CoSPs to further their chance of success.

5G drives enhancements to current cloud-native tools

5G networks demand enhanced platform awareness, where container management frameworks have a greater awareness of the underlying platform's capabilities. With enhanced platform awareness, telco workloads can benefit from access to certain platform features to improve their performance or to increase the stability and the predictability of their behavior. Here are several examples:

- **CPU pinning** to avoid unpredictable latency and host CPU overcommit by dedicating CPUs to telco containers
- **Non-uniform memory access (NUMA)** awareness to improve the utilization of compute resources for telco containers that need to avoid cross-NUMA node memory access
- **Huge pages** to accelerate memory management by using larger page sizes.

For more information on platform awareness, read the Intel application note, "Enhanced Platform Awareness in Kubernetes" at https://builders.intel.com/docs/networkbuilders/enhanced-platform-awareness-in-kubernetes-application-note.pdf.

Intel has developed several Kubernetes plug-ins that bring these enhancements to the 5G core network, all available on GitHub. Here are some links:

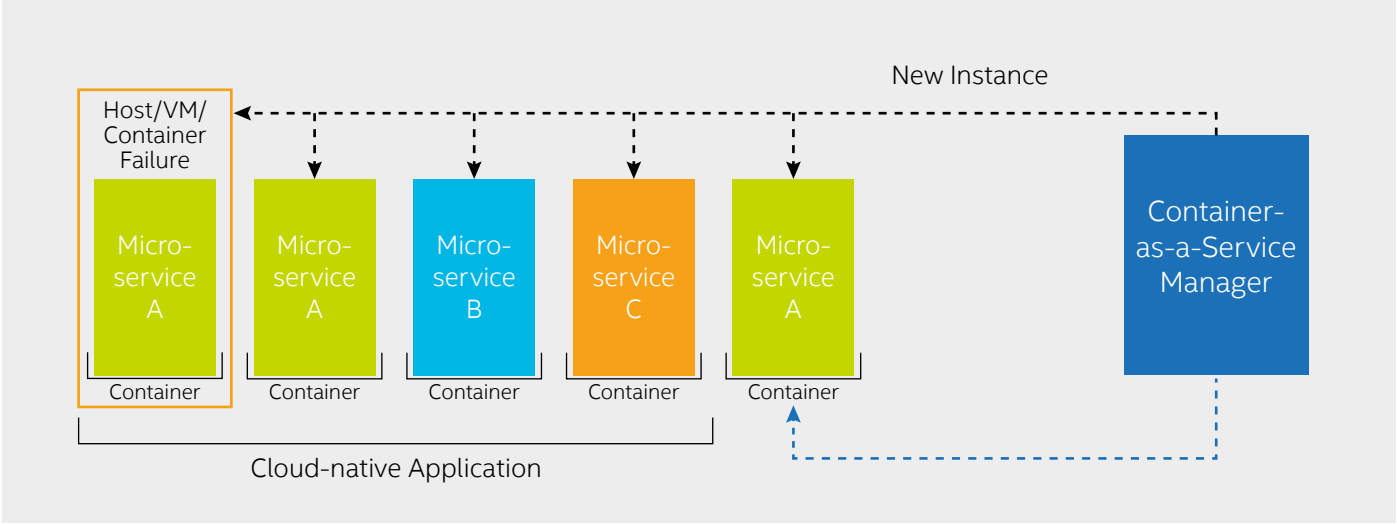

Figure 4. The CaaS manager detects one instance of microservice A has failed and automatically spins up another instance in a new container, returning the microservice to full capacity without loss of service or human intervention.

- Kubernetes SR-IOV Plug-in (I/O acceleration) https://github.com/intel/sriov-network-device-plugin
- Kubernetes Topology Manager Plug-in (NUMA enhancement) https://github.com/kubernetes/enhancements/blob/master/keps/sig-node/0035-20190130-topology-manager.md
- Kubernetes User Space Plug-in (east-west traffic) https://github.com/intel/userspace-cni-network-plugin
- Kubernetes Multus Plug-ins (separation of control/data) https://github.com/intel/multus-cni

Containerization isn't necessarily cloud native

Just because a service runs in a container doesn't make it cloud native. Containers, by themselves, are simply a new method of virtualization. Containers enable developers to package an application with only its dependencies, which makes containers smaller and faster than VMs. But that's not enough to create a cloud-native environment.

To move from mere containerization to cloud-native containerization, you need applications that are purpose built for the cloud (microservices, as discussed earlier), combined with an automated container orchestration and scheduling tool such as Kubernetes.

CoSPs don't have to do it all alone

CoSPs can take advantage of the tribal knowledge available with the Intel® Network Builders program—an extensive ecosystem of industry players that, working together, can simplify and accelerate 5G deployments. Also, Intel is enabling standard, reusable, shared platforms for NFV that are easy to upgrade, maintain and scale. High-performance Intel® hardware, such as 2nd generation Intel® Xeon® Scalable processors, Intel® solid state drives (Intel SSDs), Intel® Optane™ persistent memory and Intel® field programmable gate arrays (Intel® FPGAs) can contribute power and scalability to a CoSP's 5G efforts. And Intel is continually developing Intel® Select Solutions and other resources (such as the Cloud Native Transition Playbook, available under NDA from your Intel representative) that can ease the path to network transformation.

Cloud-native engineering best practices

When a monolith application is decomposed into multiple microservices, the microservices need to communicate among themselves (whether in an on-premises data center, in a public cloud data center, or in a hybrid cloud environment). But some questions arise:

- What happens if the network is not reliable and is error-prone?
- What happens if microservice A tries to reach service B, but microservice B is completely down?
- What if microservice B is not completely down but is slow?
- What if microservice A keeps sending its communication to microservice B but microservice B is already oversubscribed and while trying to recover, gets overwhelmed by the barrage of communication from microservice A? How does a microservice recover from such a deluge of communication?

These questions rarely arise in monolithic VNF scenarios, but are important to address in a cloud-native network deployment. The following sections discuss some solutions, including applying the Consistency, Availability and Partition-Tolerant (CAP) Theorem, using "circuit breakers" and using a service mesh with "sidecars."

Manage engineering tradeoffs with the CAP theorem

The CAP theorem[5] helps network architects determine which engineering tradeoffs are acceptable for a given network. This could be a data center network or a 5G network wherever distributed, packet-switched networking is occurring.

First, let's start with some definitions:

- **Consistency.** The response to a read request provides the most current write information.
- **Availability.** Every read request receives a valid response, but may not necessarily reflect the most current write information.
- **Partition tolerance.** The system continues to operation even if a failure in communication (i.e. a partition) occurs between two nodes.

The CAP theorem (see Figure 5) stipulates that a network can have only two of these characteristics—never all three.

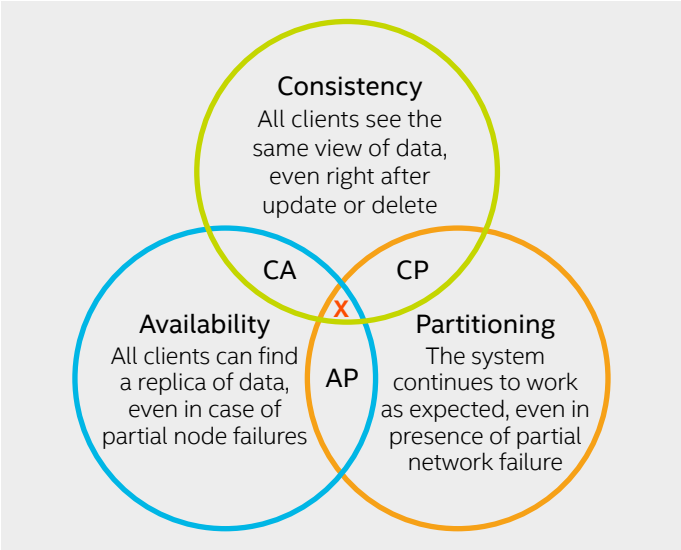

Figure 5. According to the CAP theorem, a network cannot be consistent, available, and partition tolerant all at the same time—and partition tolerance is required for real-world networks.

Consistent and available but not partition tolerant

In a perfect world, there would be no network outages (failures), and therefore all microservices residing in various parts of the network would be consistent and available. But no distributed system is safe from failure. Therefore, CoSPs must assume partition tolerance and design networks for that. The result is, then, that network architects must choose one of the other two options: available but not consistent, or consistent but not available.

Available but not consistent

If you want availability, you have to trade consistency. For example, suppose the initial value of variable A in all network nodes is 10. Now assume that the network becomes partitioned and because of that node 1 and node 2 cannot communicate with each other. At this point, variable A is updated in node 2 to a new value (20). In the "available but not consistent" situation, if variable A is read from node 1, it will return the old value of 10 because you want availability—that is, you don't want the node 1 read request to wait forever for the latest value (which it can't get because the network is partitioned). To ensure availability, you have traded consistency.

Consistent but not available

If you want consistency, you have to trade availability. Suppose the initial value of variable A in all the nodes is 10. Now assume that the network becomes partitioned and because of that node 1 and node 2 cannot communicate with each other. At this point, variable A is updated in node 2 to a new value (20). In the "consistent but not available" situation, if variable A is read from node 1, it will try to communicate to all the nodes to ensure it has the latest updated value—but since the network is partitioned, it will not be able to reach node 2 and hence the read request will hang (not respond). To ensure consistency, you have traded availability.

Applying the CAP theorem

As noted, partition tolerance is pretty much a given for any system, so network design should assume this. Network architects can achieve the appropriate level of consistency or availability through the use of synchronous or asynchronous communication. The CAP theorem is often misunderstood and applied too strictly; it is important to note that the tradeoffs between consistency and availability are not black-and-white tradeoffs, but occur over a spectrum. The goal should be to achieve a range of flexibility that combines consistency and availability in a way that makes sense for a particular application or microservice.

Manage network quality with circuit breakers

In a monolith application, calls between different aspects of the application are performed in-memory. But in a cloud-native deployment, many individual microservices communicate with each other remotely across the network. Microservice A might call microservices B, C and D, and microservice B might call several other microservices, and so on. When a particular microservice is unavailable or oversubscribed it may not be able to respond quickly (or at all) to a request. How long should the requesting microservice wait for a response? What if the failure to respond causes a cascading failure that involves multiple microservices and seriously compromises application performance? The cloud-native application should be able to gracefully recover from these types of scenario.

A CaaS tool called a "circuit breaker" can help manage the network quality. Essentially, this is a monitor that tracks a microservice's ability to respond and takes appropriate action. If the microservice is responding quickly to requests, the circuit breaker is "closed" to let requests pass through; if a certain number of requests fail, the circuit breaker "opens" and returns an error to the requesting microservice—thereby avoiding overwhelming the failed or faltering microservice. After a defined period of time, the circuit breaker allows a test request to pass through to see if recovery is complete—if so, the circuit breaker resets, or if the request fails again, the circuit breaker remains open. Other actions might be triggered by tripping the circuit breaker as well, such as spinning up a new replacement instance of the faulty microservice and re-routing requests to that new instance. For a coding example of circuit breakers, visit https://microservices.io/patterns/reliability/circuit-breaker.html.

Separate business logic from transport with a service mesh and sidecars

The business logic should not be worrying about transport-related tasks. Several cloud-native concepts elegantly insulate business logic from transport nuances. These concepts include service mesh and sidecar (see Figure 6).

A service mesh is a dedicated infrastructure layer built into an application, using an array of network proxies, that controls how different microservices share data with one another. Using a service mesh, application developers can document how each microservice interacts with all the other microservices. As an application grows, a service mesh can make it easier to optimize communication and avoid downtime. Logically, a service mesh is simply a collection of proxy sidecars; functionally speaking, the service mesh isolates the transport layer from the application layer. A service mesh benefits both CoSPs and application developers. CoSPs maintain the necessary control to apply uniform microservice communication policies, while developers don't need to write extra logic.

The individual proxies that comprise a service mesh are sometimes called "sidecars," because they run alongside each microservice (such as SMF and UPF), rather than within the microservice itself. If you are using Kubernetes as your container orchestration tool, a sidecar is a utility container in the Kubernetes pod and its purpose is to support the main container. Standalone sidecars serve no purpose; sidecars must be paired with one or more main containers.

Typically, sidecar containers are reusable and can be paired with numerous types of main containers.

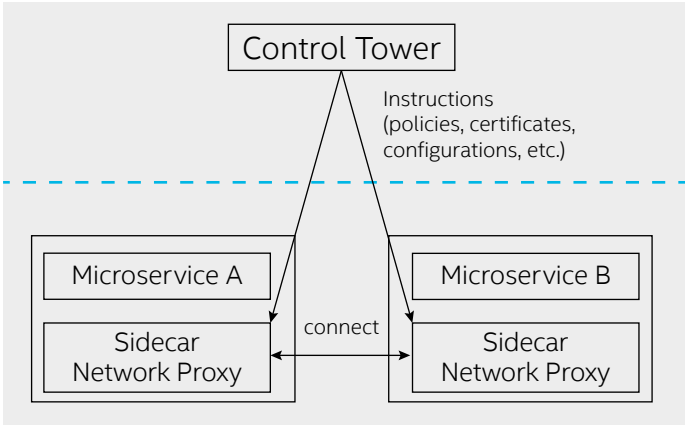

Figure 6. Microservices work together as a distributed system, where each microservice focuses on a single small task. Additional functions can be accomplished using "sidecars," which are additional containers in a Kubernetes pod.

One popular service mesh/sidecar architecture is the open source Istio project, launched by Google, IBM and Lyft. For more information, refer to https://istio.io/docs/concepts/what-is-istio/ and https://www.altoros.com/blog/using-istio-to-unify-microservices-with-a-service-mesh-on-kubernetes/. Istio is available from GitHub at https://github.com/istio/.

Conclusion

The benefits to CoSPs of a cloud-native 5G core network, especially in a standalone 5G architecture, are numerous, and are made possible by a network architecture based on microservices. These benefits include openness, flexibility, modularity and reusability and versatile orchestration. A well-designed cloud-native 5G network involves engineering tradeoffs (such as those addressed by the CAP theorem) and design principles such as Istio service mesh and sidecars.

The cloud-native industry and CaaS tools are rapidly evolving—to see the latest developments and read deep-dive discussions, visit https://landscape.cncf.io/. For more information, please contact your Intel representative or visit intel.com/cloud.

References

You may find the following resources helpful as you explore the world of cloud-native 5G networks:

- Intel's Cloud Native Transition Playbook, **available under NDA from your Intel representative**
- Intel® Network Builders program
- An introduction to cloud-native 5G concepts
- ETSI GR NFV-IFA 029 V0.8.0, Report on the Enhancements of the NFV architecture towards "Cloud-native" and "PaaS", work in progress
- ETSI GS NVF-EVE 011 V3.1.1. Specification of the Classification of Cloud Native VNF Implementations, work in progress,
- 3GPP TR 23.799, Study on Architecture for Next Generation System, V14, Dec 2016
- The Difference between Cloud-Ready and Cloud-Native
- Cloud Native Computing Foundation Tools Overview
- Network Operator Perspectives on NFV Priorities for 5G
- Vertical and Horizontal Scaling in AWS
- Why Going Cloud Native in 5G Creates Hard Choices for Software
- **"Building Microservices" by Sam Newman provides an in-depth discussion of many of the concepts covered in this white paper.**

Solution Provided By:

[1] https://blog.netapp.com/cloud-infrastructure-devops-journey

[2] Note that while the Control and User Plane Separation (CUPS) architecture provides the overall vision of separation between the control and user (data) planes, it is containerized microservices that enable that vision to become reality.

[3] https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/15.02.00_60/ts_123501v150200p.pdf

[4] For more discussion on "pets versus cattle," refer to http://cloudscaling.com/blog/cloud-computing/the-history-of-pets-vs-cattle/

[5] See, for example, https://en.wikipedia.org/wiki/CAP_theorem

0520/FP/CAT/PDF 343066-001EN