SCTE•ISBE CABLE-TEC EXPO® 2020 VIRTUAL EXPERIENCE » OCTOBER 12-15

2020 Fall Technical Forum SCTE·ISBE • NCTA • CABLELABS®

Delivering Cloud-Native Operations with Edge Compute Enabled DAA

Implementing a Kubernetes Distributed Edge

A Technical Paper prepared for SCTE•ISBE by

Marco Naveda
Sr Director, Network Architecture, Office of the CTO
Ciena
5050 Innovation Drive Kanata, ON K2K3K1 Canada
613-670-2730
mnaveda@ciena.com

Dmitri Fedorov
Software Architect, Office of the CTO
Ciena
5050 Innovation Drive Kanata, ON K2K3K1 Canada
(613) 670-2757
dfedorov@ciena.com

Raghu Ranganathan
Principal, Advanced Architecture, Office of the CTO
Ciena
7035 Ridge Rd, Hanover, MD
713-662-9999
rraghu@ciena.com

Table of Contents

List of Figures

1. Introduction

The networking software industry is experiencing an accelerating technology shift from centralized data center application delivery models to a distributed edge computing paradigm. In this new computing paradigm, cloud infrastructure and services are delivered from multiple distinct and geographically distributed locations in closer proximity to the end user or data source. The drivers for this shift are the emergence of advanced Enterprise use cases & applications, network infrastructure convergence and operator digital transformation initiatives. These applications require deterministic latency and ultra-fast response times, distributed processing of large volumes of data near the source and flexible scalability across network and computing ecosystems. Included in this transition, the convergence between wireless & wireline, combined with increased hub-site network capacity in cable Distributed Access Architectures (DAA) and the drive to centralized mobile Radio Access Networks (cRAN), are making distributed edge computing more feasible for operators.

This computing paradigm is enabled by cloud-native principles and application containerization & orchestration technologies that promise new operational efficiencies and agility to develop innovative services. We see this trend across Enterprises and network operators that want to accelerate software delivery while maintaining a consistent quality of experience from applications & data to devices and end-users alike, independent of location. 5G and high-speed fixed broadband connectivity services are acting as catalysts to enhance the underlying network performance and agility using a cloud-native services-based architectures. This in turn poses a challenge of operationalizing an applications-first approach in the operator's network to facilitate open, modular, and portable multi-vendor networking software across their distributed edge compute infrastructure platforms.

Advanced cloud-based management & orchestration systems, however, were not designed for distributed edge computing. These systems were optimized for very large compute environments where server clusters are co-located and mesh inter-connected by an over-provisioned data center fabric. In edge centric architectures, compute & network nodes are geographically distributed and inter-connected by multi-layer access and aggregation networks with varying degrees of capacity, latency, and flexibility. This creates the opportunity to adapt and optimize the use of containerization and cloud orchestration technologies to meet the needs of embedded real-time network functions and edge business applications.

This paper will delve into emerging edge computing infrastructure architectures, available open-source software to enable distributed edge computing, and key technical considerations to accelerate adoption of new software-based operational methods. This paper evaluates the use of open-source projects from the Linux Foundation Edge (LF Edge) and Cloud Native Computing Foundation (CNCF), such as Akraino Edge Stack and Kubernetes, to build and operationalize distributed edge computing networks. This paper will also explore how hyperscale cloud platforms are addressing this technical challenge and how cable MSOs can leverage a rich technology ecosystem to architect open edge technology systems, implement software defined network operations, and monetize new edge-based business services.

2. Drivers for Access Network Modernization

In the ever growing quest for delivering higher reliability, higher speed connectivity services to residential and business customers, MSOs are overhauling their traditional Hybrid Fiber Coax (HFC) networks along three key dimensions: (1) disaggregation and distribution of vertically integrated, proprietary functions of the CMTS previously located at headend or hub locations; (2) deeper fiber-based Ethernet/IP connectivity to remote access infrastructure nodes where MAC and PHY layer processing occurs; (3) convergence of service functions & infrastructure for both wireline and wireless services.

Functional disaggregation based on the DOCSIS DAA specifications allow for splitting of MAC & PHY layers from the CMTS, driving either a Remote MACPHY Device (RMD) or a Remote PHY Device (RPD) deeper into the access plant (Levensalor & Stuart, 2020). DAA extends the digital processing of the headend and hub domains out to fiber nodes, pushing intelligence closer to the end user and offering the opportunity to leverage generalized compute platforms, NFV and SDN. This in turn facilitates flexible deployment, optimized infrastructure and automated operations which would otherwise be increasingly complex due to the distributed nature of this architecture (Evolution to Distributed Access Architectures, n.d.). This is illustrated in **Error! Reference source not found.**. Additionally, since these locations are typically constrained in space and power, there is a need for light-weight, high-performance compute resources and efficient virtualization technologies such as Linux containers to maximize use of CPU and network resources.

Pushing fiber and Ethernet/IP connectivity deeper in the access plant has the effect of increasing the available access network capacity to a smaller number of homes or businesses in a service group. This in turn allows operators to boost service bandwidth and reliability, while guaranteeing SLAs via increased levels of visibility and control in the physical network underlay. A unified and automated physical & virtual network underlay is a critical component of orchestrating edge compute & storage infrastructure in support of dynamic infrastructure and overlay end-user applications and services (The Converged Interconnect Network, 2020).

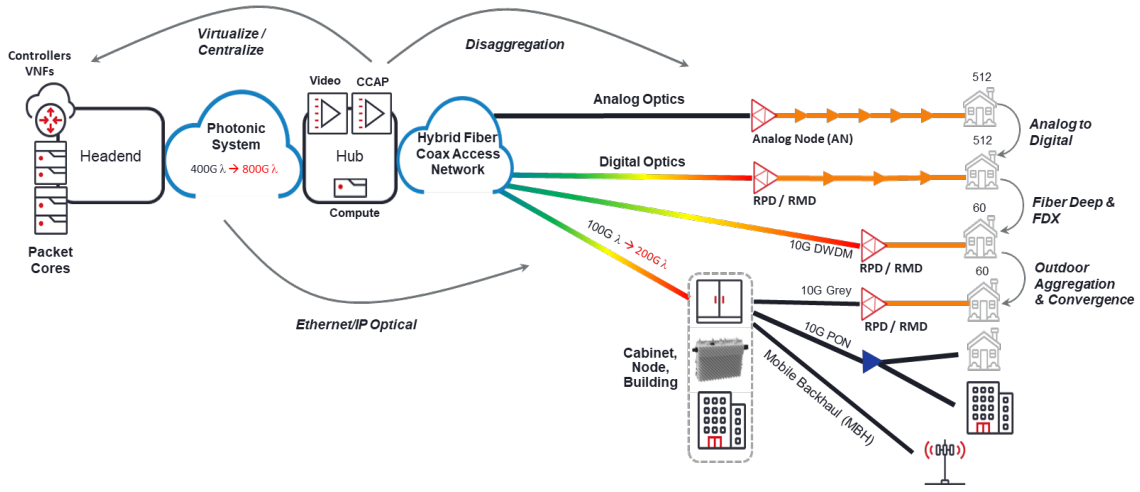

Figure 1 - MSO transition to DAA and fiber deep

Finally, Packet Core functions delivered via a virtualized CCAP such as user management and authorization are non-data/user plane software-centric capabilities that can be delivered out of a centralized location leveraging the economies of scale offered by cloud-based IT platforms. These functions can be placed in headend or regional data center locations but would necessitate a unified approach to manage the end-to-end lifecycle of open, modular, multi-vendor software components that ran as a monolithic application in a single-vendor CMTS system. This comes with the added benefit of enabling the convergence of service delivery functions across different access medium, such as FTTH and 5G RAN.

3. Cloud Native and Edge Computing Architectures

3.1. Distributed Edge Computing

As the cost of commodity computing hardware continues to decline and smart devices and sensors shrink in size, it becomes economically feasible to build connected infrastructure that is continuously monitored and optimized using data-driven insights and intelligent automation systems. In addition, 5G technology promises to enable the interconnection of tens of billions of devices, sensors, and things, so we are going to see an exponential increase in endpoints coming online and generating massive amounts of data that need to be processed closer to its source.

When we overlay this technology landscape with new kinds of business oriented real-time, low-latency applications, like self-driving cars, robotic manufacturing, industrial process automation, and augmented/mixed reality, it becomes critical to segment the application space based on bandwidth and response time requirements. See Figure 2 - Edge Application Segmentation. This high-level segmentation provides a framework to think about applications in terms of its functional components, communications requirements and where these components need to be located to meet the end user experience. Since most public cloud regions are within 60-100 msec of high-density population centers, it becomes clear that a good subset of these high intensity applications is not feasible with today's centralized cloud model.

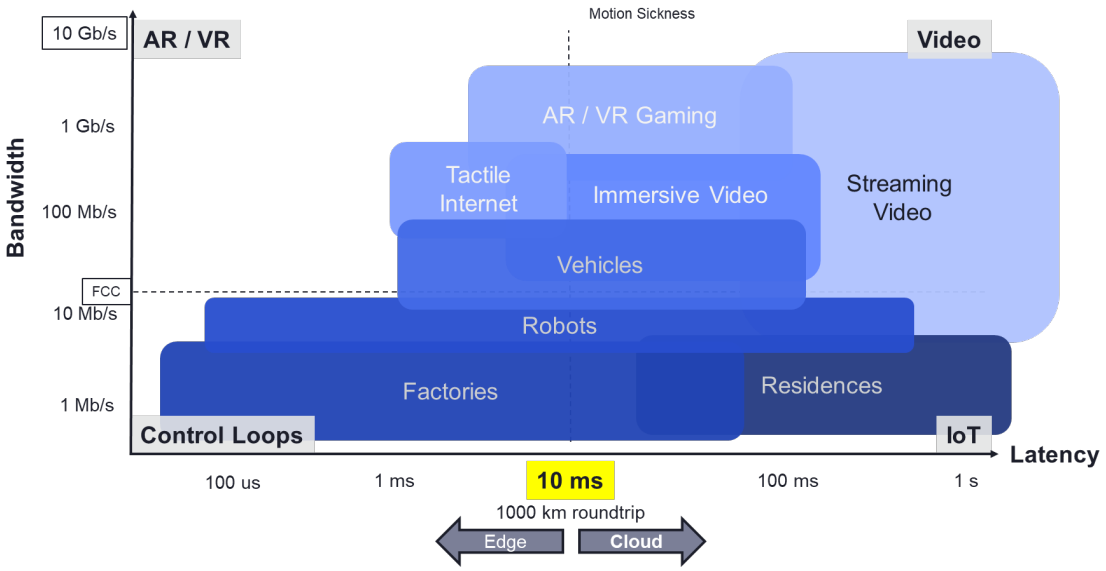

Figure 2 - Edge Application Segmentation

Centralized cloud computing has been a huge success by all measures, and businesses continue to migrate both generic IT and specialized workloads to public clouds. These businesses include Communication Service Providers (CSP) and network operators who are embracing cloud-native technologies to virtualize their networks, modernize operations, accelerate the pace of service innovation, and dramatically reduce costs. The resulting cloud workloads range from business-oriented IT applications such as planning, order management and service assurance, to network-centric software applications such as network service orchestration, virtualized routing and other CPU-based data / user plane functions. These workloads will be distributed across heterogeneous central and edge-based data centers that require a unified approach for software management and operations.

On the technology front, one of the corner-stone technologies of cloud computing – server virtualization or virtual machines – is increasingly being replaced by Linux container technology which has three compelling advantages: rapid workload deployment, better CPU utilization and lighter-weight resource footprint. This container technology supported by tools like Docker have the added benefit of simplifying application life-cycle, from creation and packaging, to testing and delivery across any infrastructure running a Linux OS.

These trends are helping drive an industry shift to create a more distributed and infrastructure optimized computing model, which moves cloud-style consumption of compute, storage, and network resources closer to the end-user or data source. We call this new computing paradigm Distributed Edge Computing (DEC), whereby a software-centric approach facilitates dynamic placement of application components across a heterogeneous environment of connected compute & storage resources, while abstracting the complexities of operating these resources from the application developer. These resources may reside on a smart sensor powered by an ARM processor, an IoT gateway running on a network appliance or a data center located within a hub location of an MSO network. See Figure 3 - Edge Locations.

This DEC approach is also fundamental in the move towards virtualization and distribution of CMTS functions that must be dynamically, but intelligently placed across Hub and access infrastructure with the right resources to meet performance requirements, such as Low Latency DOCSIS (LLD) specifications. LLD targets 1ms queuing and overall less than 10 ms round trip time in the access network (Whie, Sundaresan, & Briscoe, 2019).

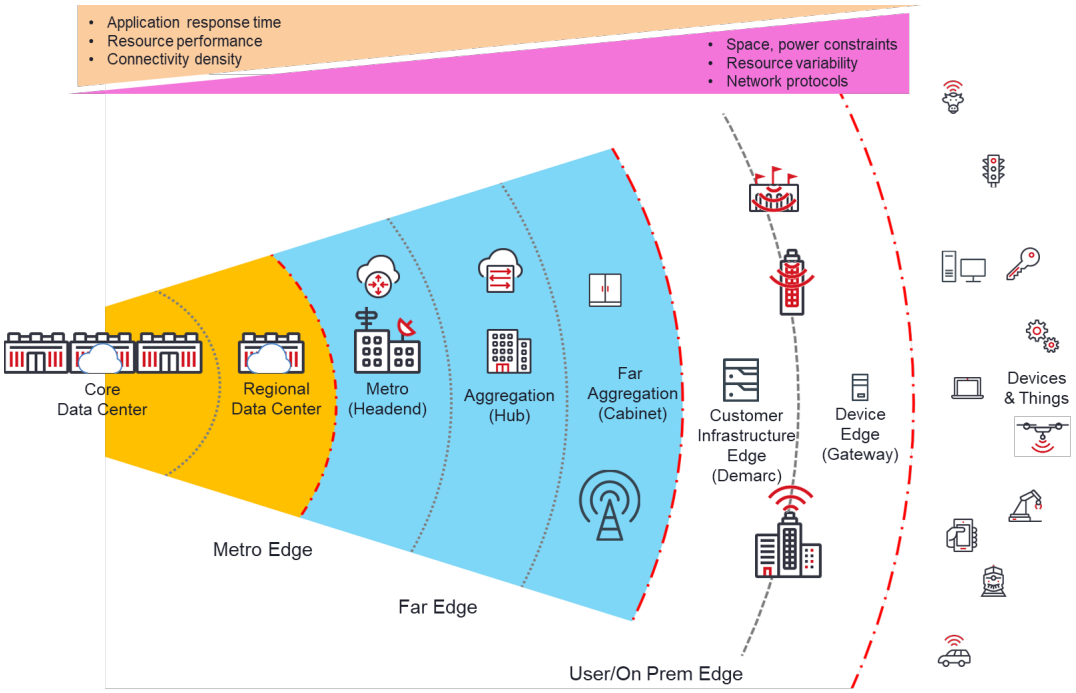

Figure 3 - Edge Locations

3.1.1. Properties of Distributed Edge Computing

There are in fact multiple names and definitions of this style computing, so we propose that there are three key properties shared across the spectrum of edge-centric computing architectures.

- *Location*: physical location of a compute node in a distributed environment defines communications latency, data sovereignty and ability to perform specialized hardware-assisted processing. This covers a spectrum of compute capabilities from a sensor/device in a mining field to customer premise and operator network infrastructure hosting network and application services within a metropolitan area.
- *Heterogeneous cloud:* distributed edge computing is an optimization of central cloud computing, and as a result it inherits properties such as shared resource pooling, elasticity, and application agnostic infrastructure. This is often overlooked but it emphasizes the need to support a holistic and cloud-native approach to containerized application delivery and multi-location data processing according to resource constraints, performance & quality of experience.
- *Network diversity*: as application components and supporting resources become more distributed, the underlay communications network becomes more diverse, and increasingly critical, in terms of protocols, technology domains, application awareness and transport flexibility. This is in contrast with network applications that run within a server cluster in the same core data center.

3.2. Cloud Tiering Reference Model

3.2.1. 3-tier Cloud Model

From a software application standpoint, edge computing infrastructure fits into a 3-tier system, where the edge tier is located between the hyperscale cloud and devices to perform specialized functions. This 3-tier system limits the extent to which device level applications need to communicate with a centralized cloud for active data storage, processing and other services, as illustrated in **Error! Reference source not found.**.

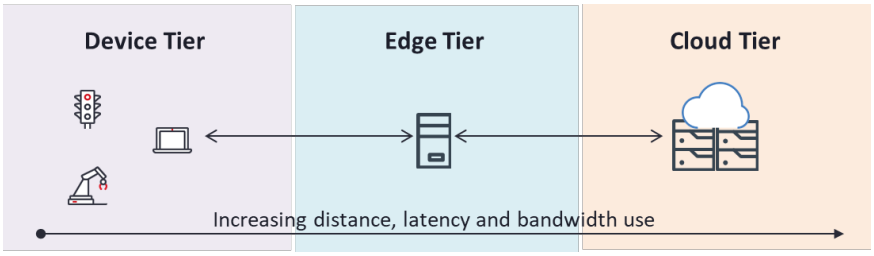

Figure 4 - Three-tier Cloud System

When this edge tier is missing, which by and large is the norm today, the system becomes a traditional cloud computing environment whereby all the application intelligence is centralized and possibly assisted by smart devices. From an application perspective, the edge tier can play a dual role to deliver network services and application services. In a 5G RAN, the edge can host radio signal processing functions and user plane functions (UPF) for local traffic breakout and termination at the application layer. Another key

role is to bring compute resources closer to the device tier to improve application response time, reduce unnecessary data transfers to the cloud tier and enhance communications security.

3.2.2. Elements of the Edge Tier

The edge tier's functional role also varies according to the capabilities and ownership of the underlaying communications network. The edge compute tier sits between the enterprise LAN and operator's metro core to leverage last mile networks and enable local traffic breakout functions for general traffic off-load, time-sensitive end-user applications as well as network-centric protocol processing supporting higher-level applications. As a result, the edge tier can be further decomposed into the Device Edge and the Infrastructure Edge as illustrated in **Error! Reference source not found.**.

The purpose of the Device Edge is to host a Device Edge Cloud or sometimes referred to as On-Prem Edge Cloud where enterprise owned devices can benefit from proximity and security of an on-net cloud environment for local application processing and storage serving the enterprise environment. This is particularly useful where enterprises need to retain control of their network traffic and where their data is processed and stored.

The Infrastructure Edge is located on the operator side of the last mile network and typically hosts an Edge Cloud environment for Telco-centric workloads, or what is commonly called a Telco Cloud. These environments are owned and operated by last mile network operators and are becoming increasingly attractive to offer internal network services, shared infrastructure wholesale services and enterprise business services in collaboration with hyperscale cloud providers. The infrastructure edge is where we see the potential for MSOs to differentiate their network services by creating access on-ramps onto an edge cloud for running gaming, AR/VR, Smart City IoT and other real-time analytics intensive applications near the data source.

The Edge Cloud represents not just an architectural choice, but also a system that encompasses storage and compute assets located at the edge of the network, and interconnected by a scalable, application-aware network that can sense and adapt to changing needs, securely and in real-time.

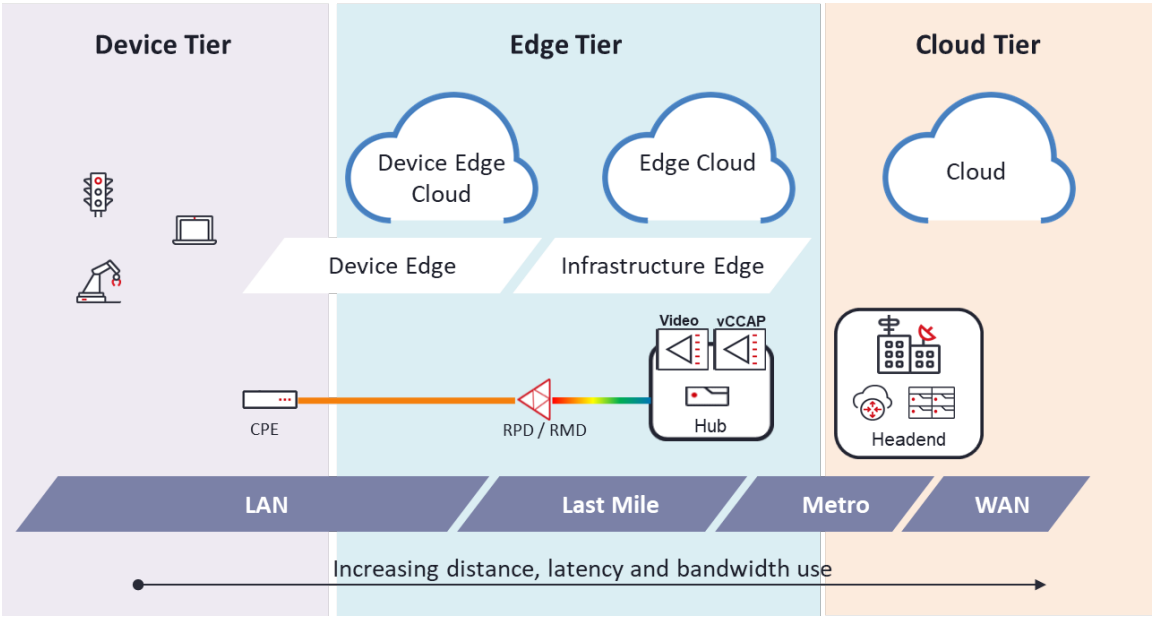

Figure 5 - Edge Cloud Network Reference Model

From an infrastructure perspective, the same cloud networking model can be applied to an MSO network, with the device tier defining the subscriber's domain and the edge tier encompassing the outside plant passive equipment, RPD/RMD nodes as well as hub locations hosting vCCAP systems. The hub and headend locations are equipped with data center infrastructure such as server clusters and cloud-native platforms to deliver the compute and networking environment for virtualized packet core and video functions. These are considered edge data centers due to proximity to the subscriber, footprint requirements and variety of network functions requiring specialized hardware depending on DAA design choices.

On the other hand, RPD/RMD nodes are not multi-server clusters, but they can be considered generalized compute node extensions of the infrastructure at the hub or edge data center, and therefore can participate in the end-to-end orchestration of resources allocated to user plane network functions in the last mile. Due to the nature of MAC/PHY processing functions, such as FEC and MAC scheduling, these nodes can benefit from having specialized accelerators like GPU, TPU, FPGA and smart NICs with very efficient techniques for pooling and scheduling micro-workloads is a distributed fashion. These edge nodes are typically in space and power constrained locations but can also be deployed in edge data center environments where a remote CMTS is desired due to population density.

One key property of these edge nodes and small edge data centers is their highly dispersed physical locations, which require high degrees of autonomous operation and resilience. Given the underlying Ethernet/IP fabric, there is an opportunity to turn a mesh of edge nodes and data centers into one pool of resources for dynamic orchestration of virtualized edge functions, when and where they are needed, thereby reducing the total cost of ownership for the operator.

4. Cloud Native Technologies

4.1. Cloud Native Applications

With the development and proliferation of hyperscale cloud platforms from Amazon, Microsoft, Google and others, a new kind of software development and delivery paradigm has emerged called *cloud native* applications. This approach is based on the principle of decomposing an application into a set of microservices that can be developed and deployed independently to accelerate & optimize the DevOps life cycle of software systems. These microservices are packaged into light-weight containers which are scheduled to run on compute nodes by a container orchestrator. There are many advantages of containers

vs virtual machines, but the principal ones are portability, low resource usage, dynamic horizontal scalability, and fast restarts.

Cloud native applications are built to run and scale in public, private, and hybrid clouds and they use the following constructs to deliver on the promise of a developer-centric approach to enable cloud scale agility, scalability and flexibility of software systems. See Figure 6 - Cloud Native Constructs.

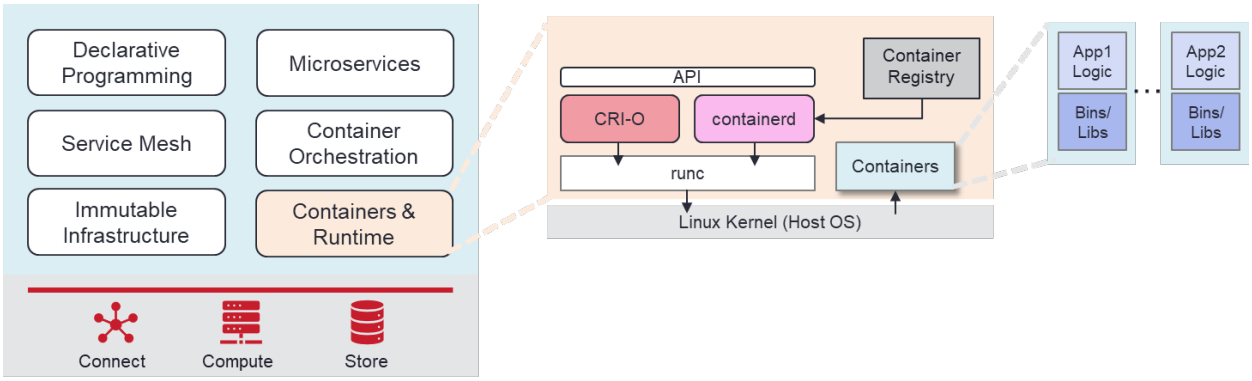

Figure 6 - Cloud Native Constructs

Containerization is an operating system virtualization paradigm in which the kernel supports multiple isolated user space instances or namespaces. From an application perspective, a container is an executable binary packaged with its lib dependencies and intended for execution in these private namespaces with resource constraints such as CPU, memory and storage. The lifecycle of a container is managed by what is commonly called a container runtime. There are several container runtime implementations, each with their own approach at managing the end-to-end lifecycle of a container. In Linux, the execution phase of a container is generally performed by runc (github.com/opencontainers/runc, n.d.), an Open Container Initiative compliant implementation, but alternatives such as *rkt*, pronounced "Rocket" (coreos.com/rkt/, n.d.), are also available. The configuration and image management are performed by applications such as docker, containerd and cri-o which interact with runc. In an effort to simplify the integration with the different container runtime flavours, the Kubernetes Container Runtime Interface (CRI) offers an abstraction layer to interact with the underlaying container runtime.

Using containers instead of virtual machines increases CPU utilization and significantly reduces disk space requirements. This is because containers running on the same host share the operating system (OS) while virtual machines have their own OS, providing complete isolation between apps. This property makes containers a more attractive choice for running software-based network functions and applications at the Edge Tier's compute, power and space constrained hardware.

Service mesh is a dedicated infrastructure layer for service-to-service communication. This infrastructure includes not only network connections between containers and services they form, but also a means of discovering services. This layer makes possible direct communications between containers under policy control (therefore the term *mesh*).

Microservices are loosely coupled fine-grained services with lightweight communication protocols. This architectural style was developed as an alternative to large, tightly coupled services. It brings not just

modularity and scalability vital for applications at the edge cloud, but also supports incremental integration with legacy systems and distributed, parallel development of software (Namiot & Sneps-Sneppe, 2014).

Immutable infrastructure is the concept of never requiring server infrastructure to be modified to support new requirements, but rather new servers are built to replace the old ones. This approach reduces operational complexity by eliminating the need to deal with differently upgraded systems and makes possible quick and fully automated recovery from faulty software that was rolled out to customers. Also, when implemented as the foundation for container images and their workloads, immutable infrastructure removes from consideration the difference between development, test, and production environments.

Declarative Application Programming Interface (API) is a design style that avoids specifying *how* to perform described functions, instead, it describes *what* needs to be done. This style allows understanding and consuming of services without knowledge of the services implementation. This reduces integration complexity and promotes service modularity and scalability, simplifying operations.

Cloud native applications consist of loosely coupled, resilient, manageable, and observable container-based microservices. DevOps teams use automation to make high-impact changes frequently and predictably with minimal effort within large scale data centers. Applications designed for the Edge Tier infrastructure use the same cloud-native principles, but must take into account the resource constraints and location context characteristics mentioned in Properties of Distributed Edge Computing.

Edge Native applications are impractical or undesirable to run in centralized data centers at public, private, and hybrid clouds. These applications are developed with proximity and specialized resources in mind as well as different security, compliance and networking requirements due to location. Edge-native applications use the infrastructure edge to provide large-scale data ingest, data reduction, real-time decision support, bandwidth savings or to retain sovereignty over critical data.

In the spectrum between Edge Native and Cloud Native Applications, Edge Enhanced is a set of applications that can operate in a centralized data center, but would gain performance, typically in terms of latency, or functionality advantages when operated using edge computing. These Edge Enhanced applications may be adapted from existing cloud native applications or may require no changes if the edge cloud environment is abstracted by the container runtime engine and associated container networking facilities.

4.2. Virtual and Cloud Native Network Functions

Cloud-native principles can be applied to enterprise and consumer-oriented applications, as well as communications and networking software that we refer to as Telco or MSO workloads. These workloads can be broadly classified according to the function and services offered in the network operator's stack:

- *Management*: Business Support System (BSS) and Operations Support System (OSS) are software solutions that help operators manage the customer experience, network offerings and network operations for planning, engineering, ordering, activating, and assuring communications services. This type of application is non-real-time and does not interact with the user / data plane & network traffic directly.
- *Control*: network management, signaling & control plane software solutions provide network device inventory, discovery, configuration, performance, fault and resource management capabilities in support of user / data plane services, but are not directly involved in processing the subscriber's / customer's traffic. This includes systems such as user authentication & session management, service policy enforcement, and DHCP services for IP address assignment. These

network functions are considered management and control functions that can be deployed as virtualized or containerized applications without significant differences in network user plane performance.

- *User plane*: these are multi-layer network traffic processing functions that operate at the network layer and below for the purposes of packet inspection, encapsulation, transformation, QoS treatment, forwarding, and filtering among other functions. These are the most-real-time intensive network functions that can operate in a subscriber's cable modem or uCPE, RPD/RMD node or headend location. This is the type of networking software stack that is impacted the most when migrated to a virtualized or containerized environment due to its performance and reliability requirements, but more importantly the life-cycle management processes and techniques used by the vendor and network operator community.

Network Function Virtualization (NFV) has been in production networks for several years now, including data plane functions such as virtual routing (vRouter), virtual firewall (vFW) and virtual Broadband Network Gateway (vBNG). These software-based data plane functions, or Virtual Network Functions (VNF), evolved from their dedicated physical appliance counterparts. When a network function is implemented as a software stack that runs in a virtualized compute environment, as a Virtual Machine (VM), it is referred to as "virtual" or "virtualized network function" (VNF). In many cases, several VNFs will operate in an edge cloud as part of a network service chain that provides a composite service to the end customer.

Network functions can run inside a virtual machine or a container. When a network function is built and deployed as a cloud-native application it is referred to as "cloud native network function" (CNF) or "containerized network function". See Figure 7 - VNF vs CNF. This means that the software is distributed as a container image and deployed, managed, and orchestrated by tools like Docker and Kubernetes. Both VNFs or CNFs support lifecycle operations that enable frequent and automatic deployment and updates of the software; this is fundamentally different from traditional processes where network operations update network elements on a controlled and infrequent basis.

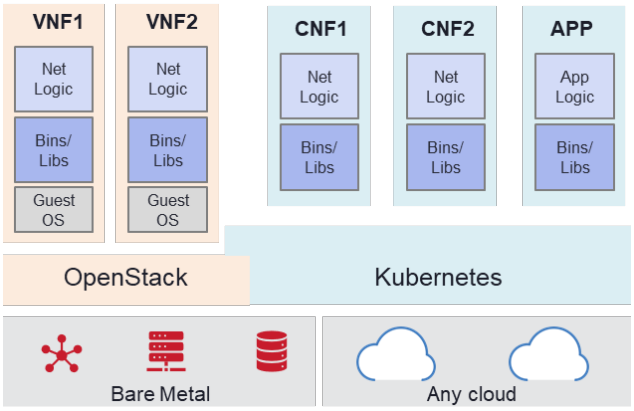

Figure 7 - VNF vs CNF

It is important to note that even though the software capabilities of a VNF may be containerized, the VNF itself is not orchestrated as a containerized application because the delivery mechanism for deployment is a VM image. In other words, containers inside the VM are not exposed to an external container orchestration system. Container orchestration or lifecycle management inside a VNF is typically handcrafted by the VNF vendor using tools like Docker compose and therefore container resource management is limited to the resources allocated to the VNF instance at deployment time. The

implication is that mechanisms for the operator to deploy, scale, monitor and heal these software-based network functions are very different and they operate at different abstraction levels, VM-based VNFs running on virtualization systems such as OpenStack or VMware vs container-based CNFs running on a container runtime such as Docker and orchestrated by Kubernetes.

These differences are critically important at the infrastructure edge where real-time user plane functions need to make optimal use of resources and require direct access to acceleration hardware, where containers can be much more efficient. Another key consideration for edge network functions is the location of the control plane responsible for lifecycle operations across a distributed set of compute elements hosting VNFs or CNFs. In the case of containerized functions, this control plane is increasingly based on Kubernetes which is based on a collocated server cluster concept. More on this in the next section, but the reliability and make-up of underlaying network infrastructure will impact the control plane design.

4.3. Container Orchestration

4.3.1. Kubernetes

Kubernetes is one of the most widely used container orchestration and management platforms in the Enterprise IT industry (Alusha, 2019). The rise of Kubernetes (K8s) has enabled cloud providers to offer managed K8s services that allow enterprises to create a hybrid / multi-cloud environment for their applications. K8s is an open-source system for automating deployment, scaling, and management of containerized applications (Kubernetes, automated container deployment, scaling, and management, 2020). K8s simplifies the deployment of scalable, distributed applications by managing the lifecycle of containers, including scheduling, load balancing and distribution across different server nodes.

Kubernetes was designed for large scale cloud environments, and it works well out-of-box only when the infrastructure edge consists of one or more Kubernetes clusters and their master nodes have a fast and reliable connection to worker nodes. The Kubernetes master node (or control plane) is relatively heavy, and while its worker nodes are less resource demanding, they are still not lightweight at all. While specific sizes vary depending on its distribution and version, Kubernetes best practices for running large clusters (Kubernest Best Praactices, n.d.) recommends at least 4 GiB of RAM and a single core Intel Xeon CPU for a master node that controls up to 5 worker nodes. When the number of nodes grows, so grows the RAM and CPU requirements for the master node, getting to 60 GBytes and 36 Intel Xeon CPU cores for more than 500 nodes.

A Kubernetes cluster consists of the components that represent the control plane and a set of machines called nodes, sometimes referred to as worker nodes (Kubernetes Overview, n.d.). See Figure 8 - Kubernetes Components, taken from (Kubernetes Components, n.d.).

A pod is the smallest scheduling unit in Kubernetes and represents a set of containers that are tightly coupled, share resources and therefore run in the same worker node. Kubernetes runs workloads by assigning pods to nodes based on the resource usage and limits from the application. Each pod has its own IP address, and a default Kubernetes control plane runs its own DNS service for service to address resolution.

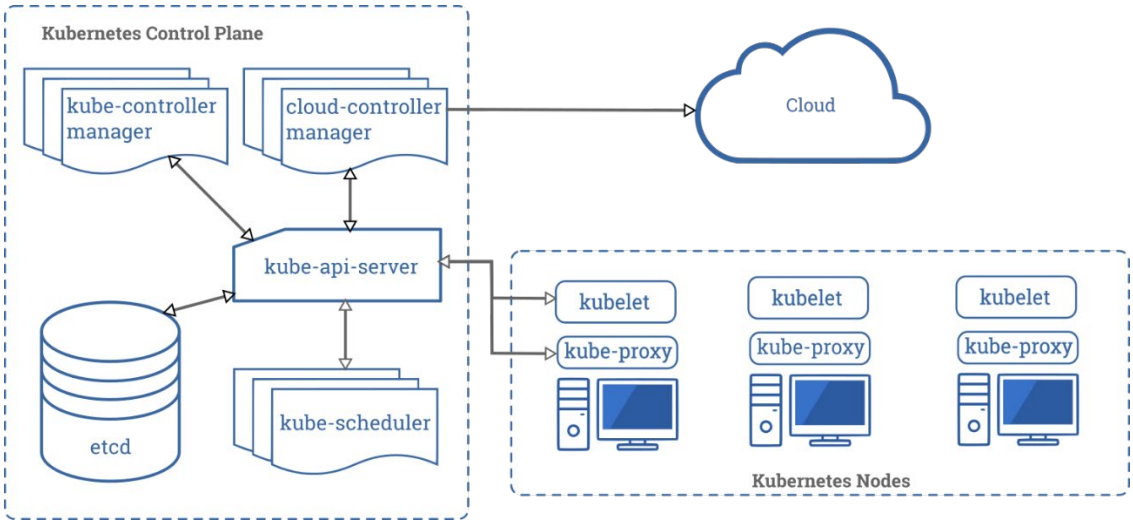

Figure 8 - Kubernetes Components

The Kubernetes control plane (master node) consists of:

- the database (etc. by default) that stores all cluster data,
- the API server (kube-api-server) that exposes the Kubernetes API and serves as its frontend,
- the scheduler (kube-scheduler) that watches for newly created pods and selects a node for them to run on,
- the controller processes runner (kube-controller-manager) that runs node, replication, endpoints and other controllers responsible for managing different elements of the cluster,
- the cloud-specific processes runner (cloud-controller manager) that runs processes specific to the cloud provider,
- for high availability deployments, the master node is configured on three separate machines.

The Kubernetes data plane (worker node) consists of:

- kubelet is the agent that accepts pod specifications from the control plane and runs them on the local container runtime (e.g. docker),
- kube-proxy is a network proxy that implements the Kubernetes Service concept, where one or more pods can sit behind a network service for load balancing purposes.

Kubernetes uses Network Plugins that run at the node level to configure container & pod network interfaces in the Linux OS and perform IP address management. By default, the kubelet is assigned with a plugin that supports a cluster-wide IP network. Container Network Interface (CNI) is a CNCF project that provides the specification and tools required to implement plugins to manage the allocation and deallocation of network resources for a container. Kubernetes supports plugins that adhere to the CNI specification and support can be extended as required by introducing new plugins for specific network functionality such as supporting container communications over VXLAN and MACVLAN (github.com/containernetworking, 2020).

Vanilla container-level networking is not suitable when orchestrating containerized data plane network functions (CNF) in a service chain or when more complex network overlays are required, such as creation and management of VXLAN tunnels that extend beyond a single Kubernetes cluster. Network Service Mesh is a cloud-native project that adds network capabilities to the Kubernetes ecosystem to enable dynamic cross-connections between local and remote CNFs (network service mesh, 2020). It offers an

API to establish connectivity between network services in an abstract way and provides policy-based service function chaining.

4.3.2. Kubernetes Distributions

Since a default Kubernetes installation (often referred to as "vanilla" Kubernetes) cannot be used without installing additional components, it is recommended to use one of the free or commercial Kubernetes distributions. A well-chosen Kubernetes distribution instead of the "vanilla" one reduces the operator's dependency on Kubernetes experts and offloads a lot of installation and configuration work, this is definitely a preferable choice, at least until the operator develop in-house expertise and automation tools with Kubernetes. A broad categorization of distributions is illustrated in Figure 9 - Kubernetes Distribution Models.

There are several dozen Kubernetes distributions recognized by the (CNCF), and it is recommended to select from this list according to the operator's requirements (CNCF Cloud Native Landscape, n.d.). We describe a couple of distributions below that are relevant for edge cloud.

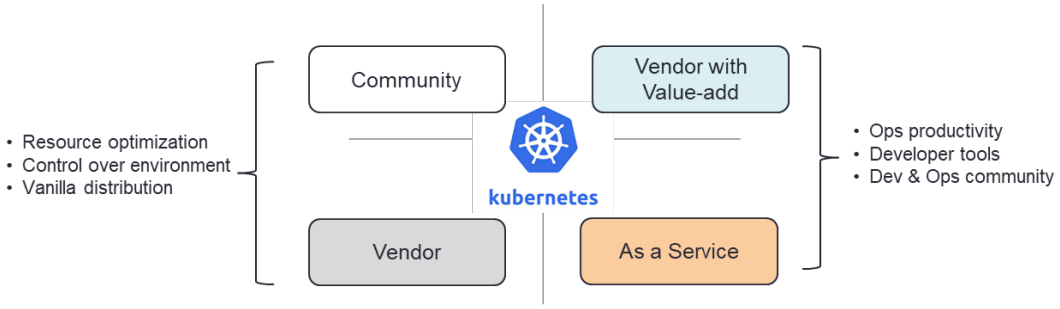

Figure 9 - Kubernetes Distribution Models

4.3.3. Vanilla Kubernetes Distribution

Kubernetes sources are available at GitHub (https://github.com/kubernetes/kubernetes) and many organizations take this source "as is", build it for different platforms and distribute it without any significant addition. This form of distribution makes it more convenient than building it from the source and has a clear correlation between such distribution and a tagged Kubernetes version (see https://github.com/kubernetes/kubernetes/tags). Canonicals "Charmed Kubernetes" distribution (https://jaas.ai/canonical-kubernetes) is an example of a vanilla distribution. This distribution has broad applicability across Telco and MSO workloads without specialized compute requirements for containers.

4.3.4. Kubernetes with Value-add Capabilities

Some commercial organizations take the Kubernetes source and add significant functionality to it, making a commercial distribution with dedicated support. Kubernetes distributions such as Red Hat OpenShift (https://www.openshift.com/) and Rancher (https://rancher.com/) provide installers for fully automated cluster deployment as well as abstractions and tools for DevOps processes and CI/CD pipelines.

Both Red Hat OpenShift and Rancher use special, container-oriented Linux distributions for worker nodes (CoreOS and RancherOS respectively). Some of these distributions offer a Cluster API capability to manage multiple Kubernetes clusters that can be deployed on-prem, private cloud or public cloud.

4.3.5. Kubernetes as a Service

Kubernetes as a service is offered by all major cloud providers, AWS, Microsoft Azure, IBM and Google Cloud Platform. In this kind of distribution, the cloud service provider takes care of everything related to the Kubernetes version, allocation, and installation of master (control planes) and worker nodes, and all the underlying compute, storage and network resources. This type of managed Kubernetes service is managed from the cloud providing the ability to unify on-prem and cloud hosted clusters and flexibly deploy containers across a hybrid cloud.

This type of Kubernetes deployment makes practical sense for back-end management workloads that oversee an operator's network but are not suitable for infrastructure edge user plane workloads. In some cases, the cloud provider supports edge cloud environments that can be collocated in a hub/headend or operator edge data center to run enterprise services or network control plane functions at the metro level. When considering this service, one must clearly understand the operational benefits, managed services costs and the tradeoffs associated with lack of control and visibility of the underlying infrastructure.

4.3.6. Lightweight Kubernetes for Edge Cloud and IoT

A Kubernetes cluster consists of one or more master nodes and one or more worker nodes. Each node has Kubernetes binaries that dictate the role and behavior of the node. These binaries come with significant overhead, which makes them impractical for certain edge deployments where the underlying hardware is limited in compute and memory resources. Lightweight Kubernetes distributions address this class of hosting environment by stripping off functionality that is not required in small scale, single node use cases.

One of such distribution is k3s by Rancher built for IoT and edge compute (k3s.io, 2020). This distribution builds Kubernetes from a reduced source tree and changes its binaries structure with the single goal of making it as small as possible. It replaces the *etcd* database with reduced storage based on *sqlite3* and removes in-tree storage drivers and cloud providers. Also, it reduces the memory footprint by running many components inside a single process. This results in a "lightweight" distribution that is suitable to run both control and user planes on devices with limited resources, making it a great fit for Edge Cloud nodes to coincide with RPDs/RMDs.

Canonical makes another "lightweight" Kubernetes distribution called MicroK8s (MicroK8s, 2020). This distribution also targets IoT and edge computing. It makes most of Kubernetes options default, resulting in simplified installation, configuration, and updates.

4.3.7. Kubernetes for Distributed Node Clusters

One more specialized Kubernetes distribution (or at least its approach) that should be considered. By default, Kubernetes requires a reliable connection between its master and worker nodes. KubeEdge is designed for environments where this connection may be intermittent and the Kubernetes master node (or control plane) is at the Cloud Tier while worker nodes are at the Edge Tier.

This distribution replaces the worker node's *kubelet* software with its own lightweight node agent called *edged*, and adds a special software layer on top of the master node. It changes how a worker node communicates with its master node located at the Cloud Tier, enabling it to work across non-reliable network conditions which the default Kubernetes control plane cannot tolerate.

Consider this alternative when there is a requirement for hosting the control plane in a central cloud location. The tradeoff for this is increased complexity and a significant deviation from the default Kubernetes, which locks the operator to this specialized distribution.

5. Building out a Distributed Edge Cloud

It is important to highlight the technical considerations in the deployment of virtualized infrastructure and network software applications such as in a DOCSIS DAA network. A proposed framework for this discussion is shown in Figure 10 - DAA Orchestration Framework.

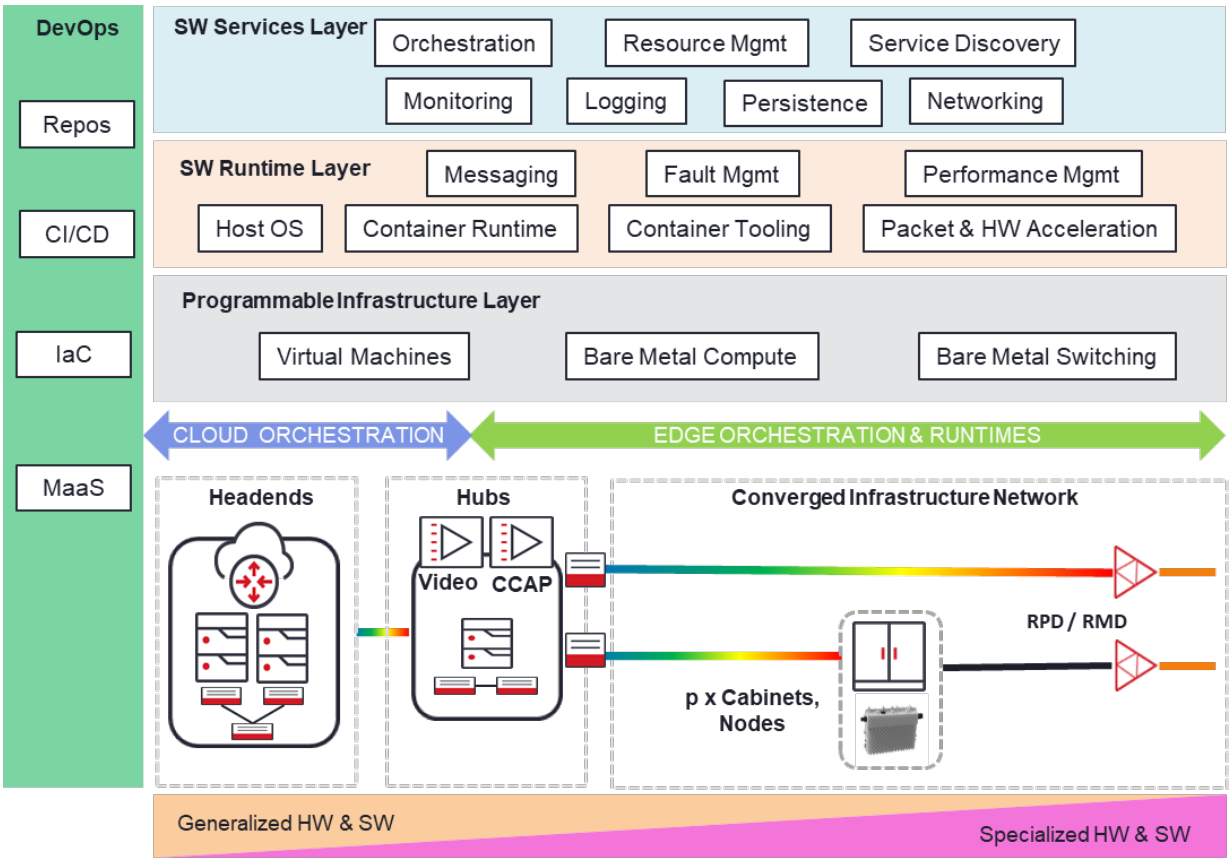

Figure 10 - DAA Orchestration Framework

5.1. Cloud vs Edge Orchestration

As discussed before, Kubernetes orchestration of containers/pods has some underlying assumptions about homogeneity, network underlay and proximity of worker / compute nodes. The Kubernetes scheduler needs reliable and fast connectivity to its compute nodes, in this case across the access network to reach Hub and RPD/RMD nodes. Kubernetes networking also assumes that the underlying switching infrastructure support a flat IP domain interconnecting the physical host nodes.

There are primarily three design alternatives:

1. Define Kubernetes clusters according to homogeneous resource type and functional requirements: e.g. nodes with specialized resources to support RPD/RMD functions belong to the same cluster. This means multiple edge clouds consisting of either general purpose compute or specialized compute & accelerator are managed by different Kubernetes instances. This adds complexity to managing distinct clusters, but it allows for independent scaling of cluster and the interconnection network
2. Define Kubernetes clusters according to a flat physical topology of the interconnection network and including heterogenous resources within the same cluster, i.e. general and special purpose resources. This simplifies the management of the centralized Kubernetes control plane itself, but complicates the scheduling algorithms needed to make container/pod placement decisions
3. Hybrid approach of the above two options to allow for incremental scaling and performance management of the network as services and subscriber density increases.

Similar issues arise with the default deployments of OpenStack because it was not designed for distributed edge clouds; resource heavy control planes need to be deployed at multiple locations with another management layer required to coordinate between different clouds. An end-to-end network service orchestration capability can be used to unify and stitch together services across multiple edge clouds.

5.2. Network Functions and Runtimes

One of the most important techniques for working with network functions at the edge is to treat them as individual building blocks or "microservices" with well-defined interfaces for configuration and user plane stitching. This provides operational flexibility to introduce new services and software updates, when and as needed, in an automated fashion.

Whether network function is control or user plane oriented, they will rely on standard communications protocols for interfacing with platform services or other network functions. Data transfers using RESTful mechanisms is the most widely used style implemented as JavaScript Object Notation (JSON) over HTTP/1.1. While this provides simplicity for clients, it does require a web server embedded in the network functions. Messaging using Remote Procedure Call (RPC) is protocol agnostic and offers direct communication between clients and server applications. Brokered messaging is another alternative usually used in high performance publish-subscriber communication models that requires a message broker. And lastly for monitoring applications the Message Queuing Telemetry Transport (MQTT) protocol is typically used. It is critical that VNF and CNF vendors align on standard protocols and interfaces to simplify integration and allow for optimization of messaging systems in the runtime environment.

Selection of the runtime environment for VNFs and CNFs is dependent on whether network functions are optimized for central cloud, edge cloud or specialized appliance environments. While these choices provide similar deployment agility from a Kubernetes scheduler, the runtime OS, guaranteed priority scheduling, and HW acceleration abstraction capabilities are critical for real-time sensitive network functions. Servers with accelerators (GPUs, FPGAs) or specialized ASIC-based appliances with generalized compute need a high reliability runtime to guarantee network and CPU resources, and provide a high-performance data channels between VNF/CNFs. RPD and RMD nodes with limited resources can use lightweight Kubernetes distributions and a specialized runtime for access to packet processing, HW acceleration features and policy-driven CNF chaining.

5.3. Automated Operations & Business Network Intent

Manual deployment and operation of edge cloud infrastructure is not practical, especially with the scale of MSO networks, services and subscribers. Software delivery, VNF/CNF instantiation and replacement must be fully automated by a software services, orchestration, and monitoring layer. This is one of the reasons Kubernetes adoption has risen quickly; changes in the infrastructure, application scale up/down and recovery on failures are fully automated based on application intent provided through a manifest.

Modern automation tools can be classified as supporting declarative or imperative style of programming. With declarative style, the infrastructure is described as "what" needs to be built, vs the imperative – "how". In order to support immutable infrastructure concepts described earlier, a declarative "what" style of tools (like Terraform) that support the notion of "Infrastructure As Code" should be used for managing infrastructure. This approach decouples infrastructure description from the tools used to build it, making it possible to automate across different compute and network systems. This is beneficial to maintain a common baseline of infrastructure software across a heterogeneous environment of Headend/Hub and remote RPD/RMD nodes.

A network fabric supporting edge cloud needs a holistic, data-driven closed loop approach that automates key business processes that span IT systems like planning, fulfillment and assurance and network lifecycle operations such system connectivity & network functions configuration, scaling and healing. Describing network intent in the form of operational state allows network algorithms to determine whether or not the network is deviating from its intended state and therefore take corrective actions, such as proactively re-routing traffic to avoid congestion or alerting the operator of impending network outages before they occur. Kubernetes intent-driven orchestration of containers is well suited to support this model as long as the scheduling of end-user applications can influence the allocation of the underlying network infrastructure, whether physical or virtualized. This enables optimal use of network resources based on application needs from core cloud to edge cloud to access.

One key area that requires special extensions to Kubernetes is the scheduling & placement of containerized functions based on not only CPU, memory, and specialized HW availability but also networking constraints that exist in a distributed edge cloud environment across headend/hub and remote nodes. The basic Kubernetes scheduler assesses in real-time CPU and memory resource requests and limits to decide on resource allocation and contention management. This works well in data center applications where the cluster fabric is overprovisioned and fully meshed, so placement decisions can be made virtually ignoring available bandwidth or effective round-trip-time between nodes. A network aware scheduler is required to maintain not only container resource usage and limits for compute and memory, but also topological constraints, specialized accelerators, network capacity, packet loss, latency and jitter metrics.

5.4. Application Repositories

Each application and virtual network function need to be placed in one or more repositories where the orchestration layer can retrieve and deploy across the infrastructure. This repository provides a controlled place where version-controlled artifacts can be maintained. This capability is vital for mitigation of issues that arise after a botched deployment. Automation must be able to restore a system to the last working state without network operator intervention. It is highly desirable to have this repository integrated with a software development Continuous Integration/Continuous Delivery (CI/CD) process or vendor pipelines to accelerate tactical deployment of new features and fixes.

Another proven practice in automated software operations is the blue-green model of deployment, where two identical software stacks are maintained, with the first (blue) being alive and the second (green) being on standby. Switching between stacks should be automated and without any downtime to the end-user or network service. When new components are released, they are added only to the green stack which becomes operational, while the blue remains on standby. Once the green stack has demonstrated no production issues, the blue stack is updated. However, if the green stack fails, then "old" blue stack becomes operational, while the green one is taken offline for troubleshooting.

5.5. Undercloud Architecture

In advanced software-driven networks, there is an expectation that the end-user applications riding over the network can influence network behavior and resource allocation to meet QoS, security and reliability requirements. The dynamic nature of containerized multi-cloud applications necessitates a flexible and intent-driven approach to automatically allocate, configure, monitor and scale compute and network connectivity resources between the physical infrastructure underlay and the network functions overlay as depicted in Figure 11 – Intent-Driven Undercloud.

The Physical Infrastructure Underlay (PIU) is made up of all the physical resources installed at headend/hub and remote node locations for the transmission, switching and processing of video and data services delivered to subscribers. This physical underlay is SDN-controlled and highly instrumented to enable high fidelity telemetry to be used by the Containerized Network Overlay (CNO) to ensure network applications and functions are meeting the demands placed on them by the End-User Application Service Mesh (ASM).

The CNO is made up of CNFs, and VNFs where appropriate, to deliver network control and user plane services such as vBNG, 5GC and UPF, vRouting and vCMTS. Due to the multi-layer nature of the Converged Interconnect Network (CIN), a Network Service Mesh (NSM) associated with the networking Kubernetes clusters can be used to drive policy-based service chaining and configuration of the CNO and PIU layers. These service chains, both control and user-plane centric, support specific services that are specified as "network intent" by the Application Service Mesh. This technique ensures that the CNO layer is making closed loop decisions with dynamic network information to operate within the policies specified by the operator while maintaining the intent of the network services specified by the ASM. This coordination of intents is based on the use of application manifests that specific resource requirements and constraints to the Kubernetes control plane.

It should be noted that the CNO and ASM layers are orchestrated and managed by different Kubernetes control planes which may be operated by the network operator or a separate application/cloud provider with integration into the operator's Intent-Driven Undercloud (IDU) and direct network peering into the Headend/Hub sites.

The Intent-Driven Undercloud (IDU) is a policy layer that maps application networking intent into a Network Service Mesh (NSM) intent that oversees multi-domain, multi-vendor, multi-layer functions such as available inventory and capacity of resources, resource management, orchestration and monitoring of end-to-end connectivity services. The IDU initiates real-time deployment & configuration, automatically allocating physical compute, store and network resources and stitching an end-to-end connectivity service that supports the CNO layer.

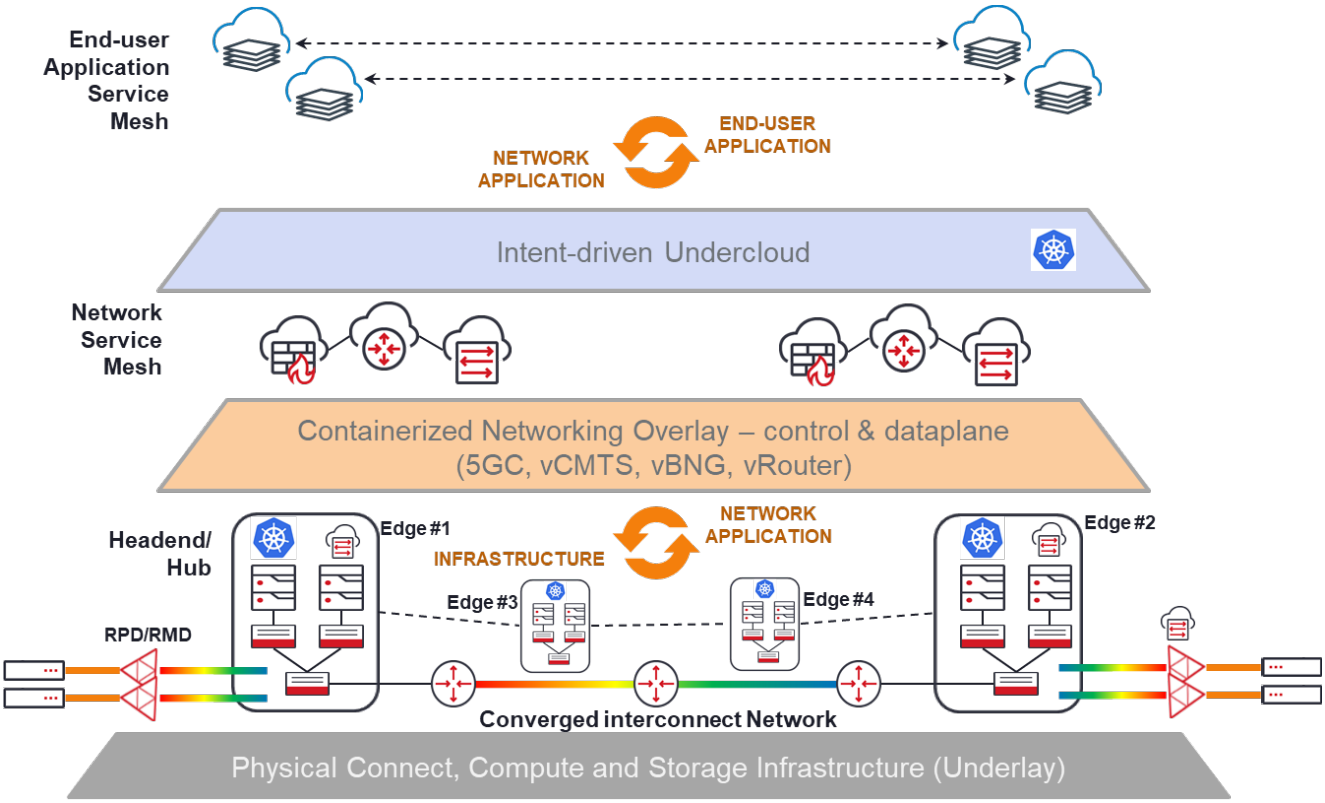

Figure 11 – Intent-Driven Undercloud

5.5.1. Undercloud Control Plane

When orchestrating the undercloud with Kubernetes, each cluster is part of a headend/hub data center, where the master nodes (or control plane) reside. This control plane site uses the undelay CIN network for reliable and fast connections to the remote worker nodes or RPD/RMD nodes. This centralized architecture enables the master nodes to be configured for high availability and to scale where compute resources are homogeneous and plentiful.

Two approaches can be used to cluster compute nodes in the DAA network:

1. Organize clusters by resource type constraints
2. Organize clusters by physical connectivity constraints

The first approach, as illustrated in Figure 12 - Homogeneous Resource Scheduler, takes advantage of native Kubernetes scheduling features for deploying CNFs within the same pool of resources allowing an external end-to-end service orchestration component to abstract the physical topology dependencies in the access network and request CNF deployments according to network centric constraints such bandwidth availability, latency, packet loss, and jitter. This method allows for independent scaling & replacement of cluster nodes, clear separation of concerns between localized compute functions and decentralized network connectivity and enables independent scaling of the CIN topology. The end-to-end Service Orchestration component is responsible for calculating network paths and ensuring application intent is met through the interconnection of physical and virtual/container functions across clusters.

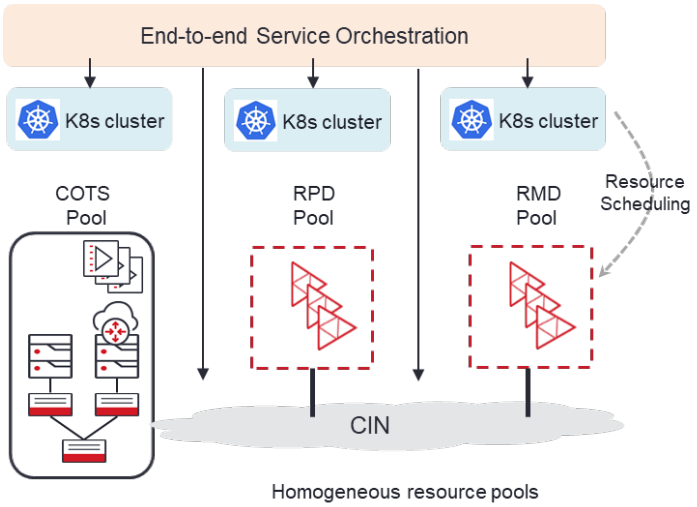

Figure 12 - Homogeneous Resource Scheduler

The second approach, as illustrated in Figure 13 - Heterogeneous Resource Scheduler, requires that Kubernetes be aware of network topology to make pod scheduling decisions. As discussed before, when Kubernetes is used for orchestrating the CNO across a heterogeneous network pool, its scheduler assigns pods to remote nodes based on CPU and memory resources only, potentially making sub-optimal decisions due to dynamic network latency and performance characteristics that it is not aware of.

To overcome this problem, the Kubernetes scheduler can be extended with capabilities to make network-centric decisions (Kubernetes Scheduler Extensions, 2020). This network-aware scheduler can use labeling mechanisms across the physical underlay to build a view of network topology, latency & BW utilization underpinning worker nodes. Additionally, it can label specialized resources such as FPGAs, GPUs and smart NICs to filter and select nodes given the CNF resource requirements.

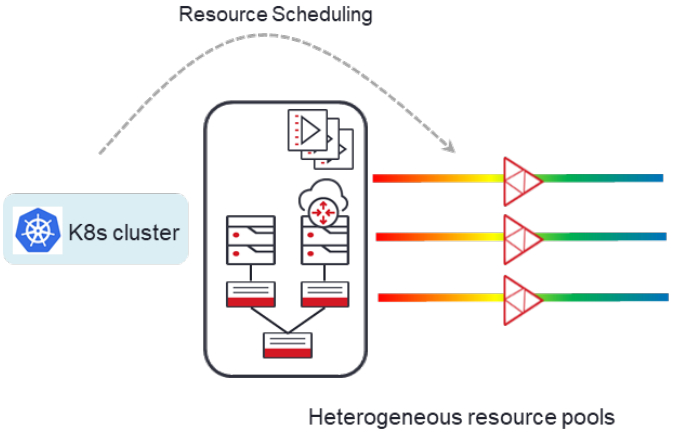

Figure 13 - Heterogeneous Resource Scheduler

This label-based, custom scheduler extension mechanism can be used to place CNFs to highly optimized nodes with run-time operating systems, available specialized software as Data Plane Development Kit (DPDK), Single-root input/output virtualization SR-IOV and various accelerators.

5.5.2. Undercloud Resources & Edge Optimized Runtime

The resource-constrained reality of the distributed edge cloud nodes means careful consideration must be given to CPU, memory, and storage requirements for Kubernetes. Since centralization of the control plane is possible, the remaining concern is with the runtime environment footprint and Kubernetes agents to coordinate with the centralized master.

We have discussed several options in this paper, including lightweight Kubernetes distributions such as k3s, MicroK8s and KubeEdge to address Kubernetes agents running on worker nodes. The other challenge is addressing the container runtime itself, as discussed in Virtual and Cloud Native Network Functions. This runtime package should be based on a real-time Linux distribution built using the Yocto Project (Yocto Project, 2020).

As mentioned above, highly optimized resource nodes with real-time operating systems and specialized software and hardware can be included into pod scheduling decisions using Kubernetes labels and custom scheduler extensions.

Within CableLabs, there is a new initiative, called Project Adrenaline, which is harnessing momentum to address the management of heterogeneous accelerators available at different locations in the cable access network. This project aims to promote technologies and architectures that enable a distributed & heterogeneous edge compute fabric to support dynamic placement of workloads (Levensalor & Stuart, 2020). The ability to orchestrate workloads and abstract the use of accelerator resources through an edge optimized Kubernetes runtime is extremely beneficial for the application developer community; this initiative will accelerate application design cycles and deployment of new features and bug fixes, independently of the underlying infrastructure allowing for concurrent innovation and cloud-style delivery.

5.6. Open-Source Building Blocks

In previous sections, we have described the use of certain open source software components to build edge cloud infrastructure. In this section, we will provide a high-level overview of relevant open source projects and pointers for additional information. Open source software greatly reduces the price for solutions and components when the same codebase is used by several businesses, and these businesses coordinate the development effort and prioritization of features. Several organizations provide a means for coordination of development of open source software, usually in the form of a membership. The Linux Foundation is one of the largest and best known open source organizations and it includes several “suborganizations” with focus on specific areas of technology, one of which is edge.

5.6.1. LF Edge

The Linux Foundation *Edge* was announced in Jan 2019 as an “umbrella organization to establish an open, interoperable framework for edge computing independent of hardware, silicon, cloud, or operating system” and includes over 30 “Premier” members from the operator, cloud and vendor community and over 40 “General” and “Associate” members (Linux foundation edge, 2020).

One of the most relevant for cable MSOs is the Akraino Edge Stack project.

5.6.1.1. Akraino Edge Stack

The Akraino Edge Stack intends to develop a fully integrated edge infrastructure solution for the Edge tier (LFEdge Akraino, 2020). This project consists of two main elements:

1. Blueprints as declarative configurations of entire software stacks to address specific use cases,
2. Software for common components declared in blueprints

The Akraino edge stack is an open-source software stack that improves the state of edge cloud infrastructure for operators, service providers and IoT networks. This edge stack can be viewed as the runtime and infrastructure layers for VNFs and CNFs. Akraino Blueprints are divided into two groups: approved and proposals, that are structured into several families. Each blueprint targets a very specific use case and a very specific deployment size, referred to as "Point of Delivery" (Akraino PODs, 2020).

Each blueprint uses the Akraino reference software stack illustrated in Figure 14 - Akraino Software Stack (The New Intelligent Edge - Akraino Edge Stack Overview, 2018).

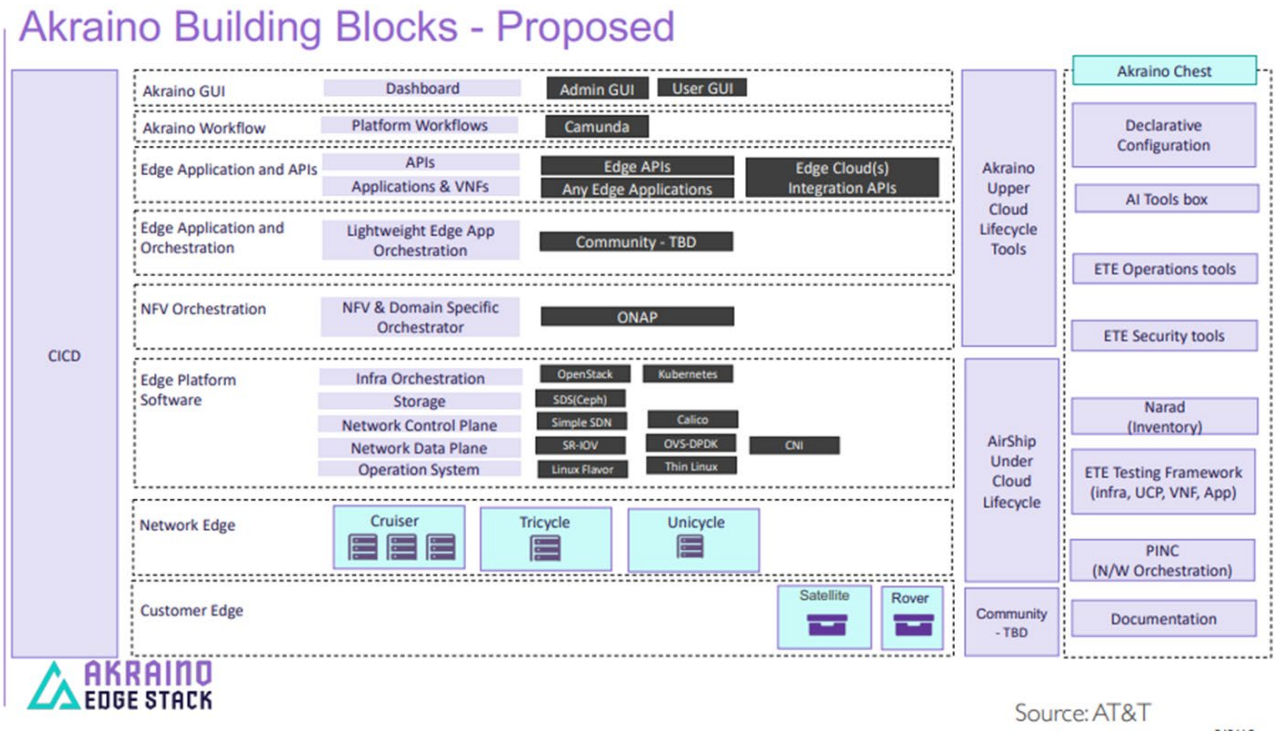

Figure 14 - Akraino Software Stack

Akraino uses an edge cloud architecture model with the full control plane located at the infrastructure edge. A blueprint represents a standard model of deployment for various operator sites: central regional and edge. While the blueprints cover a wide range of use cases with Multi-access Edge Computing (MEC) and 5G vRAN (Virtualized Radio Access Network) applicable to cable MSOs, there are no blueprints to address different control plane deployments.

A relevant blueprint for this paper is the Kubernetes-Native Infrastructure (KNI) Blueprint Family. The KNI is optimized for Kubernetes-native workloads and also allows hybrid deployments (CNF & VNF) using KubeVirt, a technology that allows VMs to run as a pod inside a Kubernetes cluster. There are currently two KNI blueprints in progress:

- Provider Access Edge (PAE) optimized for real-time and high performance vRAN and MEC workloads
- Industrial Edge (IE) optimized for small footprint and low latency for IoT, serverless and machine learning workloads

It is noteworthy that KNI uses a commercial Kubernetes distribution called "Red Hat OpenShift" (https://www.openshift.com/) and the Cluster API (https://cluster-api.sigs.k8s.io/) to deploy a Kubernetes cluster. The Cluster API is declarative and uses tooling to simplify provisioning, upgrading, and operating multiple Kubernetes clusters. This makes it much more suitable for the distributed edge computing deployments.

5.6.2. Cloud Native Computing Foundation

The Cloud Native Computing Foundation (CNCF) hosts open-source software components for cloud native applications (https://www.cncf.io/). CNCF hosts Kubernetes, however there are over 1,400 projects, product, or technologies under the CNCF umbrella, see (CNCF Landscape). It is recommended to get familiar with categories of technologies and products and to see how CNCF suggests combining them into a solution.

5.7. Commercial Cloud Platforms for Edge Computing

In its simplest form integration with a cloud platform is a matter of accessing specific endpoints on the internet, but most of cloud platform providers offer more tightly coupled software supporting computations at the Edge Tier.

When looking at this software it helps to understand that the Internet of Things (IoT) was the very first use case for edge computing that was offered by commercial and open-source cloud platforms. The IoT use case requires very specific cloud-based components (like a thing registry) and messaging protocol (MQTT), and Edge Tier nodes usually serve as IoT gateways. Also, this use case requires data processing as close as possible to the source. Such processing can be done as a dedicated process at the edge node, in the form of a standalone application, a container, a virtual machine or a serverless application. Serverless application frameworks are increasing in popularity because they do not require packaging, easily fit into event-driven design and use compute resources only when active.

5.7.1. AWS IoT Greengrass

The AWS IoT Greengrass consists of a binary that is installed on a node (a Linux server, for instance) at the Edge Tier. After installation and registering with the AWS IoT cloud service, it provides MQTT messaging service to IoT devices. Messages can be processed right at the node by a serverless application (integrated with AWS Lambda service) or forwarded to the cloud for consumption by other AWS services. AWS IoT Greengrass also integrates with artificial intelligence software, providing means to run models pre-trained at AWS or elsewhere.

5.7.2. AWS Outpost

AWS Outpost is a ready-to-use edge cloud infrastructure. It provides several compute, storage and networking services that are enough to run an Edge Tier based datacenter. Typically, an Outpost rack of server is deployed on the enterprise premise or the operator's network to peer directly with access networks such as 5G and perform real-time edge processing functions. Outpost is offered as a managed service and is considered an extension of AWS cloud regions, making it possible to seamlessly deploy applications across core and edge clouds.

5.7.3. Azure IoT Edge

Microsoft Azure IoT Edge is a binary that extend the Azure IoT Hub to the edge. As AWS IoT Greengrass, it can serve as IoT gateway, and can run containers and artificial intelligence software, all integrated with respective Azure services.

5.7.4. Azure Stack Hub

Microsoft Azure Stack Hub is another ready-to-use edge cloud offer, very similar to AWS Outpost, except that it is only a software stack that run on commodity servers. It provides several compute, storage and networking services that are enough to offer IaaS and PaaS services at edge locations or local zones.

5.7.5. Google Cloud Anthos

Google offers a different approach to enable edge and multi-cloud environments. Google Anthos is a control plane and run-time Kubernetes environment that unifies delivery of containerized applications across a wide variety of public (e.g. AWS) and private cloud (e.g. VMware) environments.

6. Conclusion

Cable operators' investments to modernize access networks and move towards DAA is opening up new opportunities to transform their operations models and differentiate their network services with support for edge-centric Enterprise applications. Embracing cloud-native principles and an applications-first mindset is critical to the success of this transformation while simultaneously creating new revenue streams for Edge Cloud applications. Adapting Kubernetes orchestration and containerization of network and application functions are foundational first steps, which when coupled with an Intent-Driven Undercloud as defined in this paper, creates an adaptive and application aware network built for dynamic scale, business agility, operational efficiency and service innovation. This strategy enables low-latency and high bandwidth on-ramps to edge cloud resources at Hubsites and DAA locations where dynamic application demands can be satisfied through intelligent placement of network & application topologies.

Abbreviations

5G	Fifth Generation cellular network technology
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AWS	Amazon Web Services
BSS	Business Support System

CIN	Converged Interconnect Network
CMTS	Cable Modem Termination System
CNCF	Cloud Native Computing Foundation
CNF	Cloud-native Network Function
CNI	Container Network Interface
CNO	Containerized Network Overlay
cRAN	Centralized Radio Access Network
CRI	Container Runtime Interface
DEC	Distributed Edge Computing
DPDK	Data Plane Development Kit
ENF	Edge native Network Function
GCP	Google Cloud Platform
gRPC	gRPC Remote Procedure Calls
HTTP	Hypertext Transfer Protocol
IT	Information technology
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
K8s	Kubernetes
KNI	Kubernetes-Native Infrastructure
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
LF	Linux Foundation
LLD	Low Latency DOCSIS
MAN	Metropolitan Ares Network
MEC	Multi-access Edge Computing
MQTT	Message Queuing Telemetry Transport
MSO	Multiple-System Operator
NIC	Network Interface Card
NFV	Network Function Virtualization
NSM	Network Service Mesh
OCI	Open Container Initiative
ONAP	Open Network Automation Platform
OPNFV	Open Platform for NFV
OVN	Open Virtual Network
OSS	Operations Support System
PIU	Physical Infrastructure Underlay
POD	Point of Delivery
REST	Representational State Transfer
RMD	Remote MACPHY Device
RPC	Remote Procedure Call
RPD	Remote PHY Device

VNF	Virtualized Network Function
vBNG	Virtual Broadband Network Gateway
vFW	Virtual Firewall
vRAN	Virtualized Radio Access Network
WAN	Wide Area Network
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

References

Akraino PODs. (2020). Retrieved from https://wiki.akraino.org/pages/viewpage.action?pageId=1147248

Alusha, D. (2019). *Cloud-native computing in 5G networks.* Oyster Bay, NY: ABI research for visionaries.

CNCF Cloud Native Landscape. (n.d.). Retrieved from CNCF: https://landscape.cncf.io/category=certified-kubernetes-distribution&format=card-mode&grouping=category

CNCF Landscape. (n.d.). Retrieved from Cloud Native Computing Foundation: https://landscape.cncf.io

coreos.com/rkt/. (n.d.). Retrieved from coreos.com/rkt/: https://coreos.com/rkt/

Evolution to Distributed Access Architectures. (n.d.). Retrieved from COMMSCOPE: https://www.commscope.com/solutions/fixed-access-networks/distributed-access-architecture/

github.com/containernetworking. (2020). Retrieved from https://github.com/containernetworking/cni

github.com/opencontainers/runc. (n.d.). Retrieved from https://github.com/opencontainers/runc

k3s.io. (2020). Retrieved from k3s.io: https://k3s.io/

Kubernest Best Praactices. (n.d.). Retrieved from Kubernest.io: https://kubernetes.io/docs/setup/best-practices/cluster-large/#size-of-master-and-master-components

Kubernetes Components. (n.d.). Retrieved from Kubernetes.io: https://kubernetes.io/docs/concepts/overview/components/

Kubernetes Overview. (n.d.). Retrieved from Kubernetes.io: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Kubernetes Scheduler Extensions. (2020). Retrieved from https://kubernetes.io/docs/concepts/extend-kubernetes/#scheduler-extensions

Kubernetes, automated container deployment, scaling, and management. (2020). Retrieved from Kubernetes, automated container deployment, scaling, and management: https://kubernetes.io/

Levensalor, R., & Stuart, C. (2020, July). *The Modular, Virtualized Edge for the Cable Access Network.* Retrieved from Adrenaline™ Project: https://openadrenaline.com/

LF Edge. (2020, June 20). *Open Glossary of Edge Computing 2.1.0.* Retrieved from LF Edge: https://github.com/State-of-the-Edge/glossary/blob/master/edge-glossary.md

LFEdge Akraino. (2020). Retrieved from lfedge.org: https://www.lfedge.org/projects/akraino

Linux foundation edge. (2020). Retrieved from Linux foundation edge: https://www.lfedge.org

MicroK8s. (2020). Retrieved from MicroK8s: https://microk8s.io/

Namiot, D., & Sneps-Sneppe, M. (2014). On -Micro-services Architecture. *International Journal of Open Information Technologies*, 24-27.

network service mesh. (2020). Retrieved from network service mesh: https://networkservicemesh.io/

The Converged Interconnect Network. (2020). Retrieved from https://www.ciena.com/insights/white-papers/the-converged-interconnect-network.html?aliId=eyJpIjoiMGFuUG9ZeTFwV2djdlR0TyIsInQiOiJsb3hSd05oZnZ4d2V5bGVZTXlnVG9BPT0ifQ%253D%253D

The New Intelligent Edge - Akraino Edge Stack Overview. (2018). Retrieved from https://object-storage-ca-ymq-1.vexxhost.net/swift/v1/6e4619c416ff4bd19e1c087f27a43eea/www-assets-prod/summits/24/presentations/21275/slides/Akranio-OverviewOpenStackv2.pdf

Whie, G., Sundaresan, K., & Briscoe, B. (2019). *Low Latency DOCSIS: Technology Overview.* Retrieved from https://www.cablelabs.com/technologies/low-latency-docsis

Yocto Project. (2020). Retrieved from https://www.yoctoproject.org/

