5G Network Evolution with AWS

Design scalable, secure, reliable, and cost-efficient cloud-native core and edge network on AWS

July 2020

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

5G is transforming the connectivity landscape, allowing lower latency and higher bandwidth across a larger scale of devices. Setting up and managing 5G mobile network functions on AWS allows for global scalability, cost reduction, elasticity, and hundreds of augmenting features (for example, AI, Analytics, IoT, DevOps). This paper highlights the best practices for designing and deploying 5G mobile network functions on AWS.

Introduction

The advent of 5G brings dramatic improvements to both the radio access network (RAN) and its core network. Besides with the most important and significant change in the radio technology, such as using millimeter wave spectrum (mmWave) for better throughput of and less latency in data transmission, network architectures of Core and RAN become to have an easier transition to the modern innovations of cloud-native software technologies such as microservices, containerized, service-based, and stateless architecture.

For the Core Network aspect, 3GPP[1] defines 5G standalone (SA) Core Network to have decomposed architecture with the introduction of a service-based interface (SBI) using HTTP/2 as a baseline communication protocol, and control plane and user plane separation (CUPS). This function decomposition, SBI, and CUPS of 5G network function software strongly favor cloud-native container-based implementation. Even though 5G network function can be built based on the legacy generation of architecture such as virtual machine (VM) based monolithic architecture, the real benefits in terms of agility, fast innovation, hyper scalability, elasticity, and simplified operations and lifecycle management (LCM) can be realized only when the 5G network functions (NFs) are designed and implemented on a cloud-native container-based microservices architecture.

In the case of RAN, decomposition of Central Unit (CU) and Distributed Unit (DU), the use of enhanced Common Public Radio Interface (eCPRI), and Open RAN (O-RAN) concept make traditional Baseband Unit (BBU) to be more easily transformed to a virtualized network function or containerized network function. This can bring the benefit of scalability and cost-efficiency in terms of both, operating expenditures and capital expenditure.

AWS and 5G

The transformation of both Core and RAN in the 5G era makes AWS an ideal platform for hosting them. This is due to the breadth and depth of AWS services and an API-driven approach to designing modern, cloud-native applications. Additionally, the promise of network slicing is expected to bloom in the 5G era, creating a private and enterprise-oriented network. As such, the 5G network would be best built on the hyper-scalable platform. Therefore, AWS becomes a natural choice for providing 5G network creation, not only for its breadth and depth of services but also because of the strongest and widest partner ecosystem across network equipment providers (NEP) and

communication service provider (CSP) in the telecom industry. Openness has been a long-standing requirement of CSPs and AWS Cloud provides exactly that – strong foundation for all NEPs to build and innovate on.

This whitepaper describes a reference architecture of 5G on AWS to help CSPs and NEPs build a carrier-grade 5G production network, in the wide spectrum of views such as design, deployment model, use cases, and AWS tools. It focuses on achieving operational excellence and agility needed by CSPs leveraging AWS products and services for networking, compute, storage, DevOps, and CI/CD pipeline. It also discusses how CSPs can monetize its network by building services using other AWS services like that for data lakes, IoT, and AI/ML.

The paper lays out the typical deployment journey of 5G evolution from using non-standalone (NSA) core, to a new standalone (SA) core network and RAN. Specifically, how relevant AWS services are used for helping cloud-native microservices and stateless architecture-based NFs will be introduced with providing a reference architecture for 5G SA core network that brings the benefit of hyper-scalability, reliability, unified programmable orchestration, and faster speed of innovation for a new service. Lastly, AWS has stretched its service coverage to the Edge cloud by using AWS Outposts, Local Zones, and Wavelength. This whitepaper highlights how these AWS services can contribute to building a seamless 5G Network with providing the best value, the most cost-saving, and the better monetization strategy to the service provider.

Building blocks of AWS services for 5G network implementation

5G has stringent requirements to meet user demands. However, due to a mature and burgeoning set of features, 5G workloads can be deployed on Amazon Web Services (AWS) despite stringent service-level requirements while also benefiting from scalability, elasticity, and high availability. Today, several customers are using AWS, its partners, and open source solutions to run workloads with reduced cost, faster agility, the ability to go global in minutes, and rich features that AWS services offer.

Customers use AWS features such as enhanced networking with an Elastic Network Adapter (ENA) and the latest generation of Amazon Elastic Compute Cloud (EC2) instances to get the following benefits:

- Data plane development kit (DPDK)

- Single root I/O virtualization (SR-IOV)
- Huge pages
- NVM Express (NVMe)
- Non-uniform memory access (NUMA) support
- Bare metal instances to meet RTC workload requirements

These instances offer network bandwidth of up to 100 Gbps and commensurate packets per second, delivering increased performance for network intensive applications.

- For scaling, Elastic Load Balancing offers Application Load Balancer, which offers WebSocket support and Network Load Balancer that can handle millions of requests per second.
- For network acceleration, AWS Global Accelerator provides static IP addresses that act as a fixed entry point to your application endpoints in AWS. It has support for static IP addresses for the load balancer.
- For reduced latency, cost, and increased bandwidth throughput, AWS Direct Connect establishes dedicated network connection from on-premises to AWS. Highly available managed SIP trunking is provided by Amazon Chime Voice Connector.

Because the 5G network aims to follow modern software architecture, it is expected to be built on microservice architecture. The most common design pattern in these days for microservice is using containers. AWS is the most preferred platform for developers to run their container-based microservice application. Today, 84% of Kubernetes workloads that run in the cloud are running on AWS.

This container-based microservice application can be empowered by many AWS services, including Amazon Elastic Kubernetes Service for managed Kubernetes service, App Mesh for service-mesh, and AWS Cloud Map for application resource discovery.

Cloud-native core network evolution with AWS

This section explains the 5G core network evolution, which is often mentioned as a cloud-native core network. As illustrated in Figure 1, a cloud-native core offers the benefit of hyper-scalability, high availability, fast innovation, programmable orchestration and automation, and new business opportunity using network slicing to CSP operators.

This section will cover how these key benefits can be ensured and implemented in AWS environment.

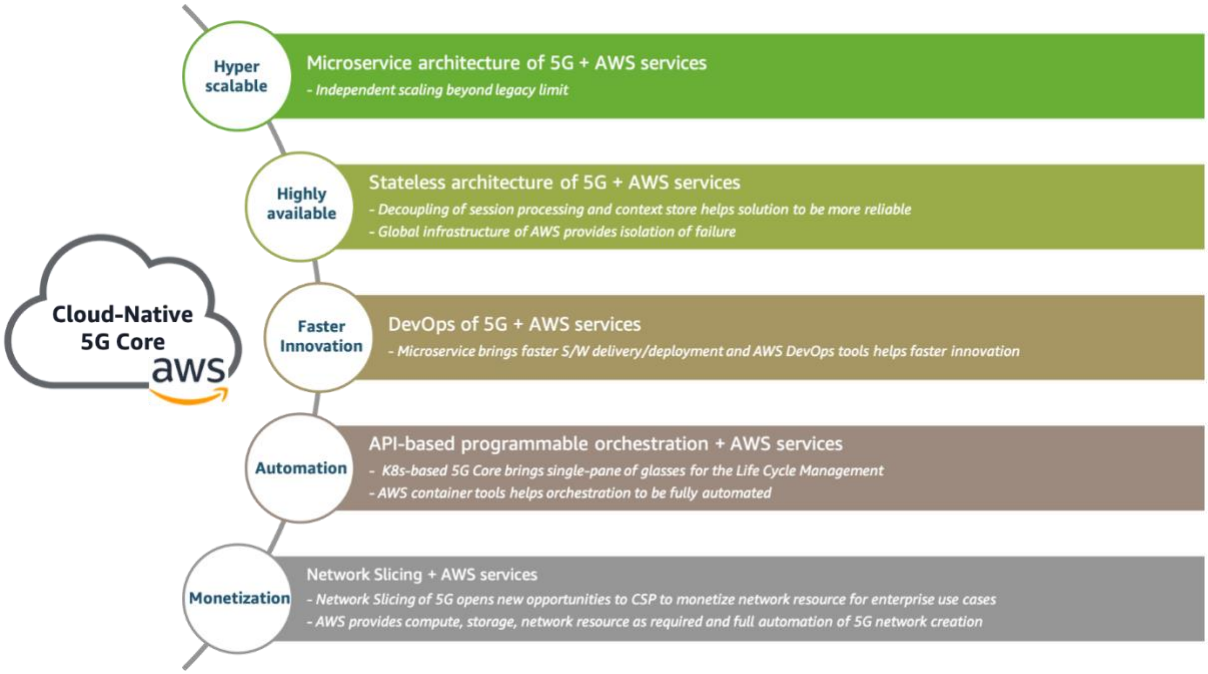


Figure 1 – Cloud-native 5G Core and AWS services

5G SA core characteristics and feature parity with AWS Services

This section describes key characteristics of 5G SA core before illustrating desirable architecture on AWS. Then, we will map the most suitable AWS service for each of those characteristics.

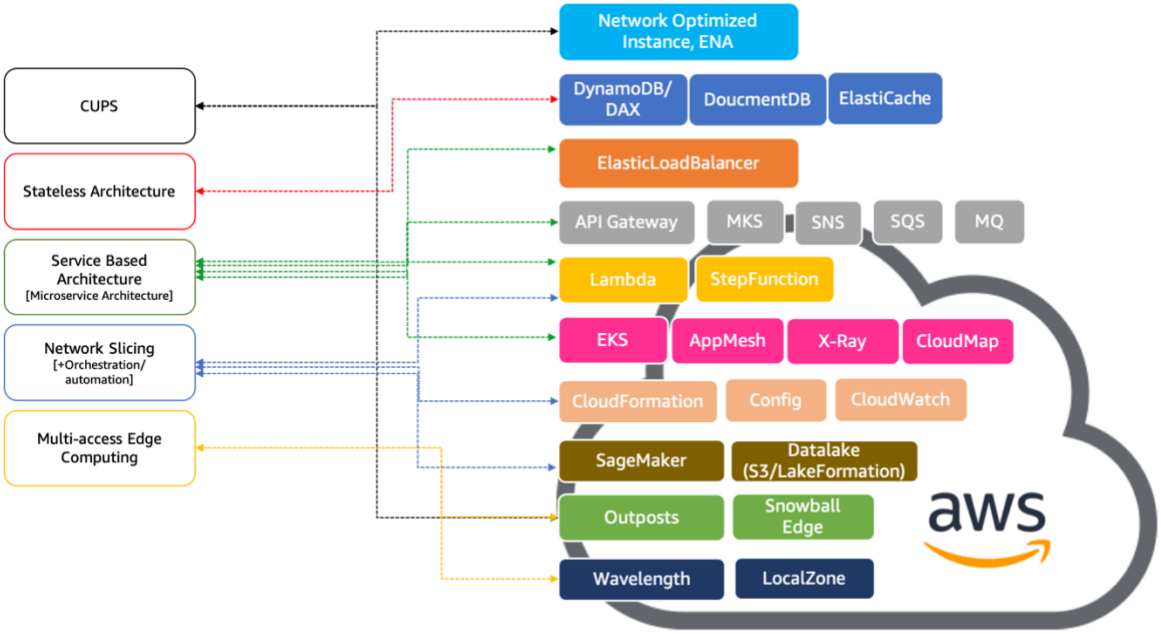


Figure 2 – 5G Characteristics and AWS feature parity

Control and User Plane Separation

5G SA Core Network inherits control and user plane separation (CUPS) architecture from 3GPP Release 14. In 4G, S/PGWs are de-composed to S/PGW-C and S/PGW-U to provide an efficiency for scaling of services independently. This decomposition continues to 5G and therefore User Plane Function (UPF) plays a role only on packet processing of user traffic, while all other singling processing is done by all other control functions such as Session Management Function (SMF). This CUPS architecture of 5G SA core can bring the benefit of cost-saving and scaling with making control plane to be centralized in the AWS Regions, while distributing UPF functions to the Edge data centers using AWS Outposts.

Stateless architecture

The 3GPP standard has introduced stateless architecture for network optimization and better reliability and resiliency. In general, a stateful network function/element must keep context information of subscriber information or transport layer association information. Hence this data must persist when the function encounters any sudden failure. On the other hand, stateless architecture is a prevalent design pattern in modern microservice-based software architecture, which necessarily has a decoupling of the application layer and the data storage layer using external data store. For this purpose,

3GPP defines Unstructured Data Storage Function (UDSF) in TS 21.195. AWS provides broad choice for data storage for supporting stateless architecture, such as in-memory database, Amazon ElastiCache that provide Redis and Memcached, and relational database like Amazon RDS and Aurora, as well as non-relational databases like Amazon DynamoDB or Amazon DocumentDB.

Service-based interface

The most outstanding change in the 5G Core Control plane is to introduce Service-based Interface (SBI or SBA: Service-based Architecture) from traditional Point-to-Point network architecture. With having this change, except for a few legacy interfaces such as N2 and N4, almost every interface is now defined to use unified interface, using HTTP/2 protocol. This change made communication among NFs to be like a service-mesh rather than serial chaining, which contributes to reduction in dependency among each interface and therefore helps with independent scaling of each function. As a result, the agility of having new features and services across network functions is increased. These SBI characteristics can have a feature parity with various AWS services which have helped a lot of AWS customers for building microservice-based applications, such as container services of Amazon ECS, Amazon EKS, and service discovery and mesh of App Mesh, AWS Cloud Map, and messaging services of Amazon MSK, Amazon MQ, API Gateway, SNS and SQS.

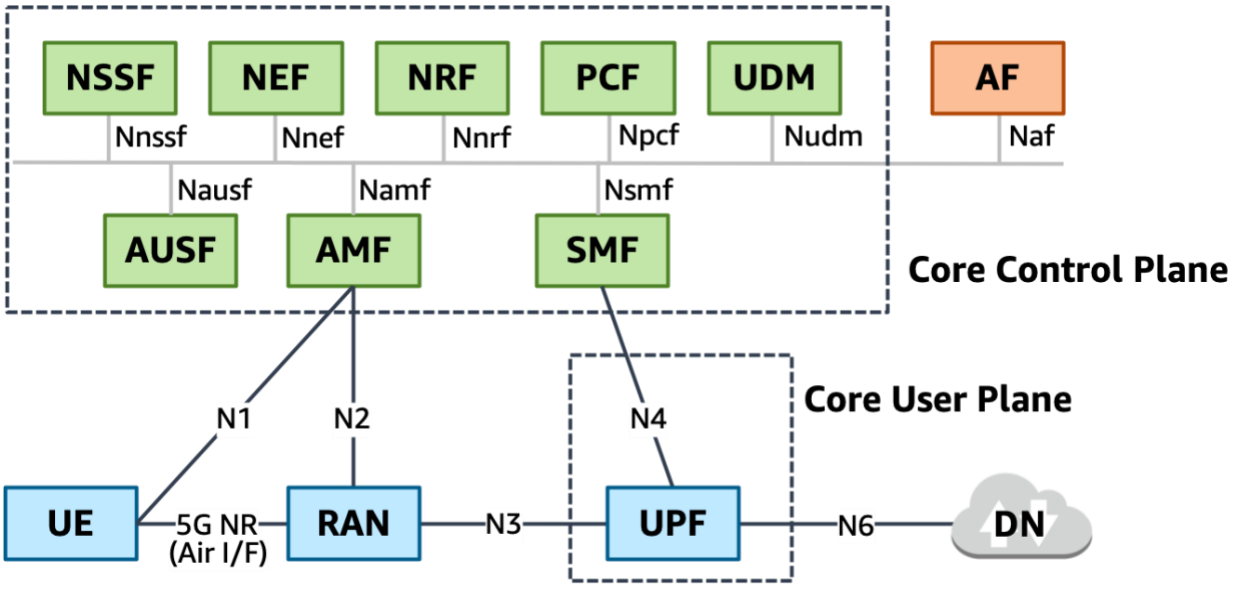


Figure 3 – Service-based Interfaces in 5G SA Core

Network slicing

Even though Dedicated Core (DECOR) and Evolved DECOR (eDECOR) have been already introduced in LTE/4G, Network Slicing is practically more burgeoning in the 5G because of cloud-native 5G Core Network architecture along with the maturity of network function virtualization (NFV) and Software Defined Network (SDN) technologies. In 5G, network slicing is expected to allow enterprise customers to leverage bespoke network which meets each requirement on customizable network capabilities such as speed, quality, latency, reliability, security, and services. [2-NetworkSlicing]. This will be a new opportunity for the monetization of network service to CSP. A variety of AWS programmable services related to orchestration and management, such as AWS Lambda, AWS Config, CloudFormation, Step Function, and CDK can help design and implement network slicing.

5G Non-standalone (NSA) core and standalone (SA) core

As defined in 3GPP and preferred by CSPs, 5G is expected to get deployed gradually, from NSA (option 3/4/7) to SA (option 2). As we will describe in the next section 3.4, AWS can host cloud-native 5G SA core with providing the benefit of agility, scalability, elasticity, and cost-saving. Moreover, as mentioned in the whitepaper for 4G EPC on AWS[2], various APN Partners already have implemented and demonstrated 4G virtualized Evolved Packet Core (vEPC) on AWS.

If CSPs choose to go in the path of Option3, they must investigate available resources of 4G core network. We know that CSPs will eventually move to 5G SA core. However, if a CSP doesn’t have enough 4G EPC resources, then AWS can be a good choice for hosting 4G Core to serve 5G NSA service for the intermediate purpose.

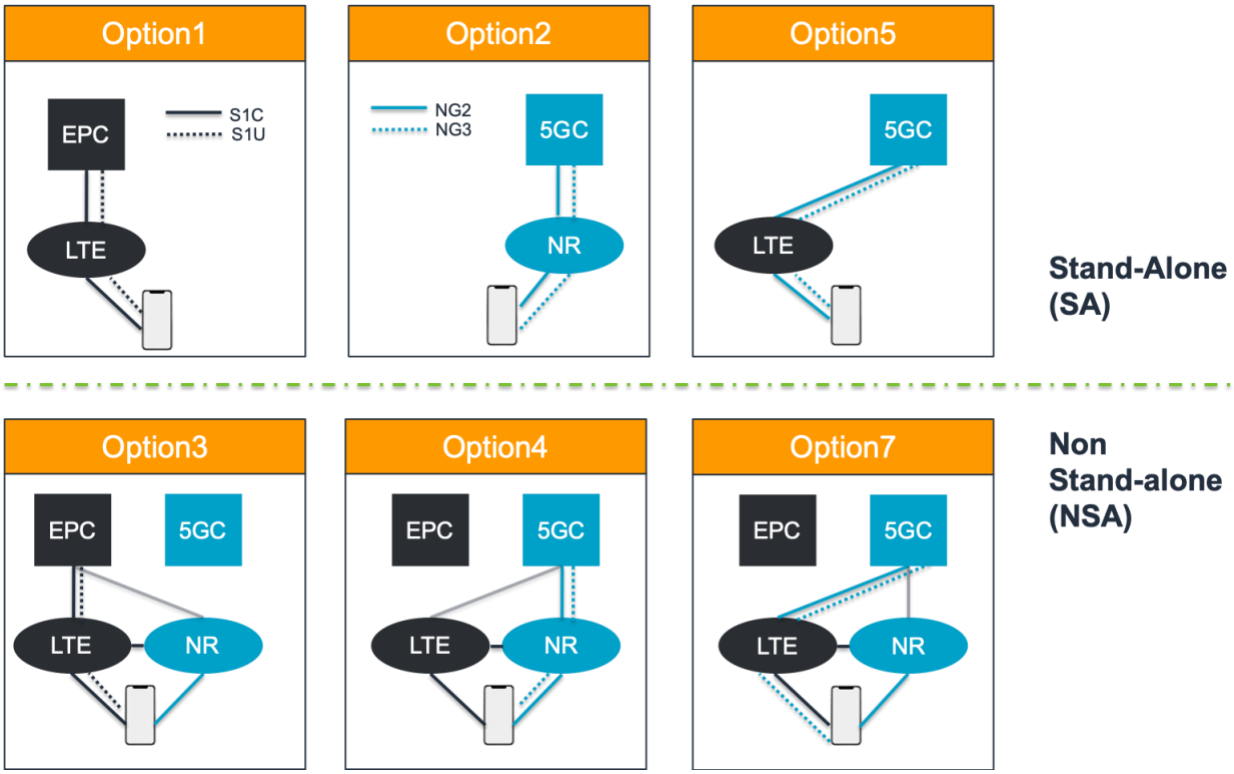


Figure 4 – 5G Deployment Options in 3GPP

Design considerations of cloud-native 5G SA core on AWS

Microservice architecture in AWS

Microservice architecture is a software architecture where the software is composed of small independent services that communicate over APIs[3]. This architecture makes it easier for applications to scale independently and faster to be developed and be updated, which then enables innovation and acceleration of time-to-market for new features.

In general, successful microservice architecture can be defined with the compliance of 12-factor apps[4]. From this general practice, we can also derive tenets for microservice-based and cloud-native network function (CNF) design:

- One code base for many deployments, and support for blue/green deployment (Factor I. Codebase, V. Build-release-run, and X. Dev/Prod Parity).

- Container-based could be preferred to control dependencies and scaling (Factor II. Dependencies, VIII. Concurrency)
- Stateless architecture with decoupling of processing and context-store functions (Factor IV. Backing services, VI. Process, VIII. Concurrency, IX. Disposability).
- Open API-based communication and use of service mesh and Kubernetes ingress service (Factor VII. Port binding).
- Independent logging, observability, and monitoring using tools (Factor XI Logs).
- Automation of management and orchestration (Factor XII Admin processes).

Various AWS tools have been helping developers to design their solution to have the best fit for container-based microservice with satisfying 12-factor rule. This best practice can be seamlessly applied to NFV design, such as cloud-native 5G core network functions.

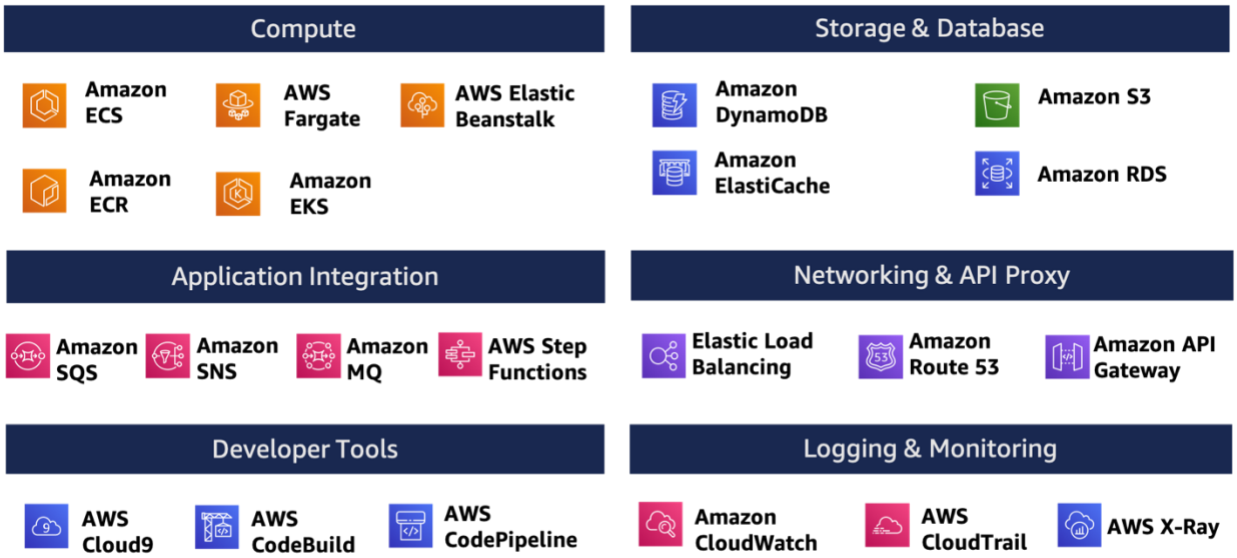


Figure 5 – Building-blocks for containerized-microservice

Design approaches for container-based cloud-native network function

Microservice architecture doesn't necessarily require a container-based implementation. However, a container-based implementation offers scalability and high availability more than virtual machine (VM) based architecture.

One of the benefits of using containers is modularization for each one-purpose function, without having a dependency on other environments, such as library and kernel version.

Once we can derive modularization of each micro-function, then we can decompose each service to have separate scalability and to minimize redundancies using reusable, small function blocks across multiple network functions. For example, container-based microservice can make exemplary architecture in Figure 6 easier than VM-based architecture. In this example, one NF (for example, AMF) can be logically defined with sharing common services with other NFs (for example, SMF), while providing independent and more granular scaling of signaling processing containers.

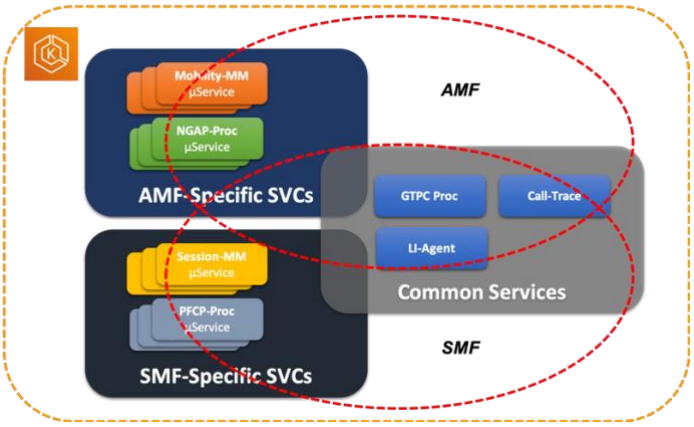


Figure 6 – Example for anatomy of container-based NF

One challenge of using microservices is that there is more management complexity than a typical VM-based implementation. Kubernetes[5] helped this orchestration challenge in many container-based applications. In the case of NFV, a VM-based solution (VNF) requires S-VNFM (vendor-specific VNF manager) and/or G-VNFM (generic VNF manager) for lifecycle management of VMs. A Kubernetes-based application generally doesn't require VNFM like middle layer management because Kubernetes itself manages the lifecycle of containers and NFs, with directly interfacing Domain-Orchestrator or Global Service Orchestrator.

Even though all those benefits are straightforward for the control plane functions (such as AMF/SMF/PCF/AUSF/UDM), it is less obvious for user-plane function (UPF) to go with a container-based approach. This is because unlike control function, UPF is likely to be distributed at the edge site. This deployment model can give a constraint for having scalability and cost-saving by resource sharing compared to control plane's centralized deployment model

> **Note**
>
> Because container-based UPF can provide unified Kubernetes orchestration with the control plane, it is design choice of NF developers and CSP operators.

You can orchestrate both container-based and VM-based implementation with the standard AWS API calls.

Kubernetes cluster design considerations in Amazon EKS

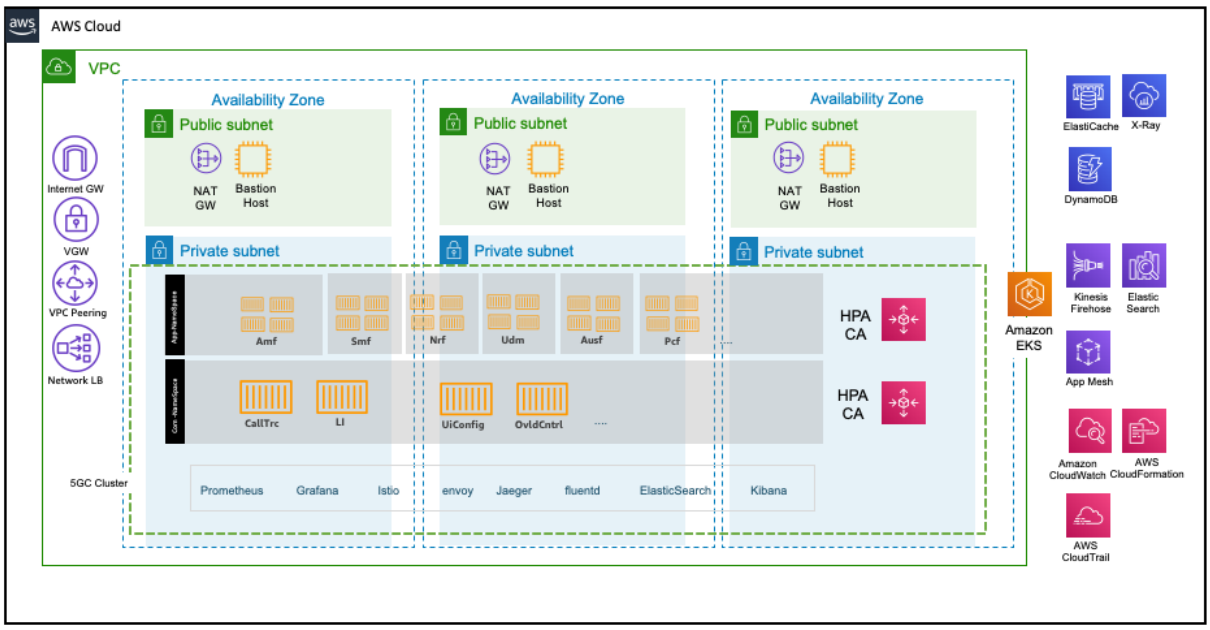


Figure 7 – EKS-based 5G Core implementation

Legacy protocol interfaces (SCTP and UDP)

Despite the 3GPP mandate for cloud-native architecture, it still has a legacy interface, with the exception of HTTP/2 such as SCTP (for N2 interface of 3GPP) and UDP (for N4 interface of 3GPP). For legacy interfaces that are not aligned with Web API style of communication, we can usually use NF vendors' home-grown load balancer pod. We can also introduce 3rd party load balancer from AWS Marketplace such as F5 BigIP.

Multi-homed pod (Multus meta CNI plugin)

In Kubernetes networking, service exposure to receive traffic from the external network must be done through Kubernetes Ingress usually with a load balancer. But in some cases, such as UPF or PGW, NF built in the cluster must receive all type of traffic destined to IP address of numerous mobiles, which makes hard to use load balancer. Besides, depending on the design of CNF, some CNF's home-grown load balancing pod can require a multi-homed direct interface to the external network for serving legacy protocols such as SCTP, as addressed in the previous bullet. In such environment, Multi-homed pod is often required for implementing CNF on the Kubernetes.

Multus CNI plugin[6] allows Pod to have these multiple interfaces in Kubernetes networking. In the AWS environment, Multi-homed Pod implementation currently can be supported in three approaches of the following diagram Figure 8.

1. Using host networking (or privileged mode). This applies to the DPDK interface.
2. Using Multus meta CNI plugin with IPvlan CNI.
3. Using Multus meta CNI plugin with host-device CNI (applicable to DPDK interface).

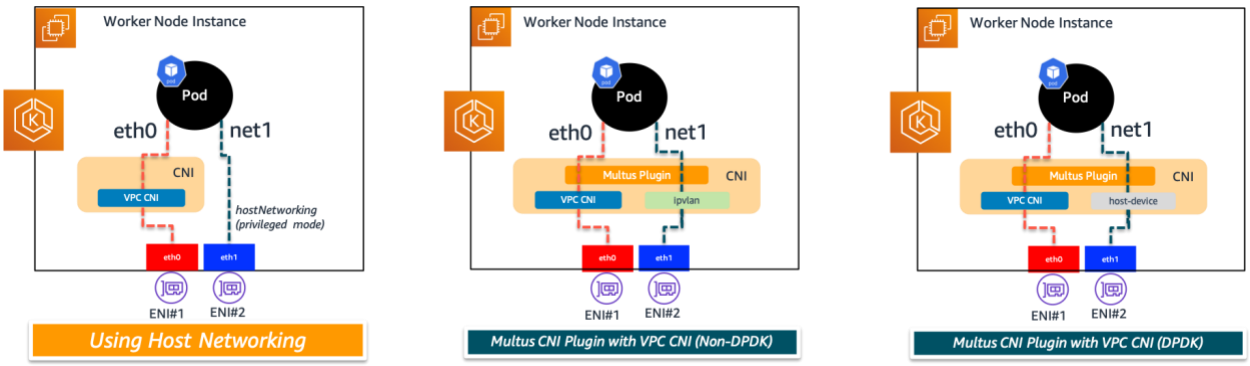


Figure 8 – Multi-homed Pod implementation on EKS

In case of UPF implementation at bare-metal systems (on COTS servers), the usual practice is using SR-IOV (Single Root I/O Virtualization) Device Plugin to collect SR-IOV VF resource pool to be available to Pod, to have higher I/O performance and lower CPU utilization for fast packet processing. But in the case of EC2 Nitro systems, since enhanced networking interface is a representation of SR-IOV VF (virtual function), all the elastic network interfaces are regarded as VF resources to the Pod, and therefore,

SR-IOV device plugin is not necessary. As shown in the preceding option 1 and option 3, once a pod gets connected to the host elastic network interface through hostNetworking or host-device CNI plugin, the pod can enable the DPDK PMD driver directly to accelerate the packet processing.

Then we can automate the process to attach additional interfaces to the worker node using CloudFormation, CloudWatch Event, Lifecycle Hook, Lambda function and Serverless Application Repository shown in the example of Figure 9 and some GitHub guides[7],[8],[9]. It is important to note, if we want to attach an additional interface to be used for the multus interface or hostNetworking interface, we must put the Kubernetes tag of "node.Kubernetes.amazonaws.com/no_manage=true' so that those interfaces are not controlled by VPC CNI Plugin. The `node.Kubernetes.amazonaws.com/no_manage` tag is read by the `aws-node daemonset` to determine whether an elastic network interface attached to the instance is under control of VPC CNI plugin.

> **Note**
>
> Attaching an elastic network interface with the `no_manage tag` results in an incorrect value for the kubelet's —max-pods configuration option. It also causes a reduced number of Pods to be available in the instance since the number of max Pods is tied up with the number of IP addresses that VPC CNI plugin can allocate from ENIs under its control.

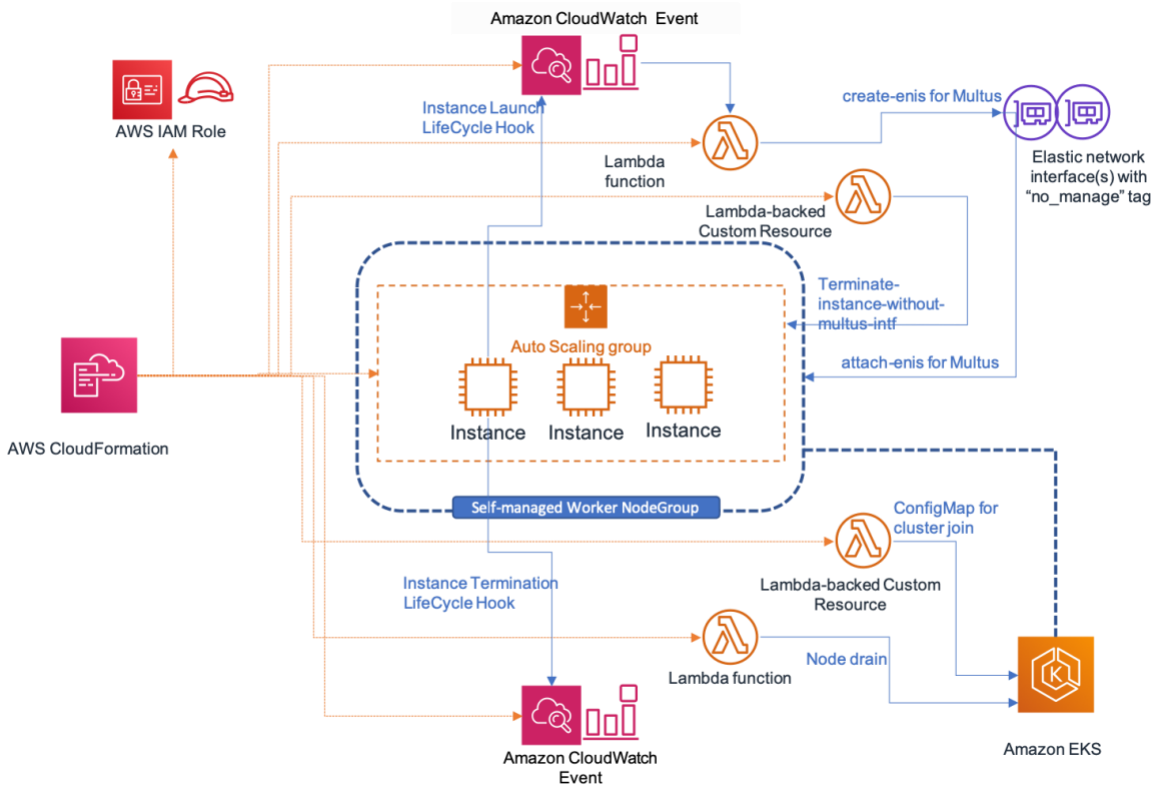


Figure 9 – Automation example for self-managed worker node group with Multiple interfaces

Cluster design

When we design a group of 5G core NFs on general Kubernetes and EKS, we must set the boundary among each network function or group of network functions. This boundary can be implemented as the cluster itself or namespace definition in Kubernetes. If we assume network function is defined as namespace (for the ease of helm chart management) as a minimum level, then we must consider the below aspects.

- **Blast radius** – The boundary of the cluster to contain multiple NFs should be set in the consideration of security and fault isolation aspect. In general, the larger the cluster coverage we have, the higher the risk of security and fault impact. Also, if each NF comes from different NFV providers, then it would be natural to set a cluster boundary for each provider since it gives more flexibility for managing Kubernetes configuration and version control.
- **Operational complexity and cost** – on the contrary of the above aspect, the more we break down NFs into multiple clusters, operational complexity, and the cost will increase, which can cause difficulty in service orchestration.

- **Network requirement** – Down to the worker node group level, we must consider the network interface requirement for each NF. For instance, UPF would be required to use the multus interfaces for N3 and N6 interface, and then we must attach specific interfaces to the worker node group for UPF.

Dev-Prod environment and canary update

One of the benefits for using AWS to host a 5G Core network is that we can create a new slice or new development, testing/staging, and production environment easily and very quickly as needed using AWS CodePipeline, CDK or CloudFormation. In AWS, we can create the same logical configuration of resources for the development environment with the production environment in the same VPC or completely different VPC (using the same IP address). If we use a completely different VPC, then we can have isolation between Dev and Prod environments from the networking layer. In the case of a software upgrade, the best practice for microservice is using Canary deployment. The purpose of a canary deployment is to reduce the risk of deploying a new version that impacts the workload. The method will incrementally deploy the new version, making it visible to new users in a slow fashion. As you gain confidence in the deployment, you will deploy it to replace the current version in its entirety[10]. In the Kubernetes environment, individual microservice can be upgraded in this manner using Elastic Load Balancing as ingress service of Kubernetes or using DNS update via Amazon Route 53. Amazon CloudWatch Synthetics allows you to monitor application endpoints more easily. With this new feature, CloudWatch now collects canary traffic, which can continually verify your customer experience even when you don't have any customer traffic on your applications, enabling you to discover issues before your customers do.

Service mesh and service discovery

One of the challenges with microservices is allowing each service to discover and communicate with each other. Service meshes are an additional layer for handling inter-service communication, which is responsible for monitoring and controlling traffic in microservice architectures. This allows tasks, like service discovery, to be completely handled by this layer. Typically, a service mesh is split into a data plane and a control plane. The data plane consists of a set of intelligent proxies that are deployed with the application code as a special sidecar proxy that intercepts all network communication between microservices. The control plane is responsible for communicating with the proxies. In an Amazon EKS based NF implementation, self-managed Istio/Envoy or AWS App Mesh can provide this service mesh function for the communication of east-west interface among microservices in one network function.

For the communication between multiple NFs in SBA, 3GPP Release 16 defines various options for using NRF and Service Communication Proxy (SCP). Those options are implementable in AWS using various AWS service discovery services such as Amazon Route 53 and AWS Cloud Map. For example, each NF must register its service to NRF at the initial launch and for that, NRF service should be accessible from each NF from the initial launch. Route 53 private hosted zone can be used to register an alias record of ELB ingress of NRF service so that other NFs can resolve the NRF IP address using a pre-known NRF URL. In other examples, Route 53 private hosted zone can be created to resolve the FQDN of each NF to find the service IP of each NF.

Hybrid network for the edge connectivity

The control plane of the core can be centralized such on the AWS Region cloud. However, to effectively distribute user traffic and to provide ultra-low latency to subscribers, RAN central unit control plane (CU-CP) and UPF like edge function can be deployed on the AWS Outposts. So, some interfaces such as the N4 interface between UPF in the edge and SMF in the Region cloud, and N2 interface between CU-CP in the edge and AMF in the Region cloud must be with the hybrid network connectivity between AWS Cloud and telecom operator's network.

As described in the user guide of Outposts[11], the AWS Outposts must have a service link connection back to the AWS Region through either AWS Direct Connect or VPN over the public internet options. In the case of 5G network implementation, because the service level agreement (SLA) requirement for the connection in terms of bandwidth, jitter, and latency, Direct Connect over a VPN is a recommended approach. Since the Outpost must be able to reach the public AWS ranges, AWS Direct Connect public virtual interface should be leveraged to set up a service link rather than other VIF types. After the service link is established, the Outpost is in service and is managed by AWS. The service link is used for AWS control traffic as well as signaling traffic between the Outpost and any associated VPCs such as N2 and N4 interfaces.

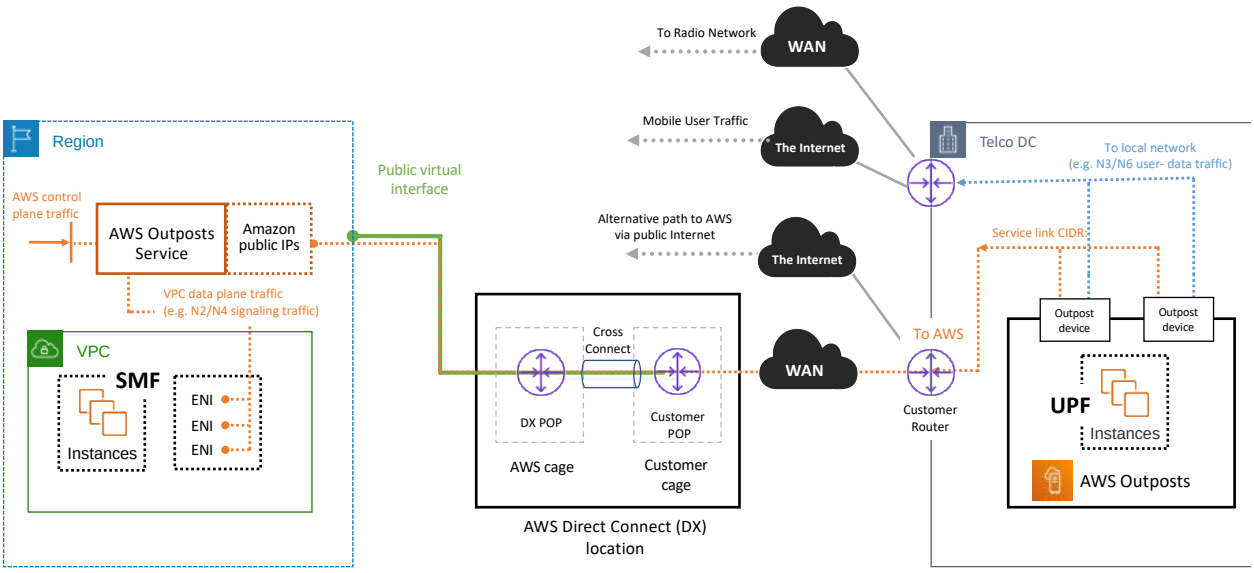


Figure 10 – Outposts connectivity through public VIF

Reference architecture of 5G Core based on AWS Well-Architected Framework

The 5G network deployment model is depicted in Figure 11 and Figure 12. In a vertical view, you can virtually implement the mobile switching office (MSO) or central data center in the AWS Region that hosts the OSS/BSS and the majority of control plane core function. Then you can implement the local central office (or distributed data center) with a fleet of AWS Outposts hosting mostly user-plane functions, such as UPF, RAN CU, and MEC.

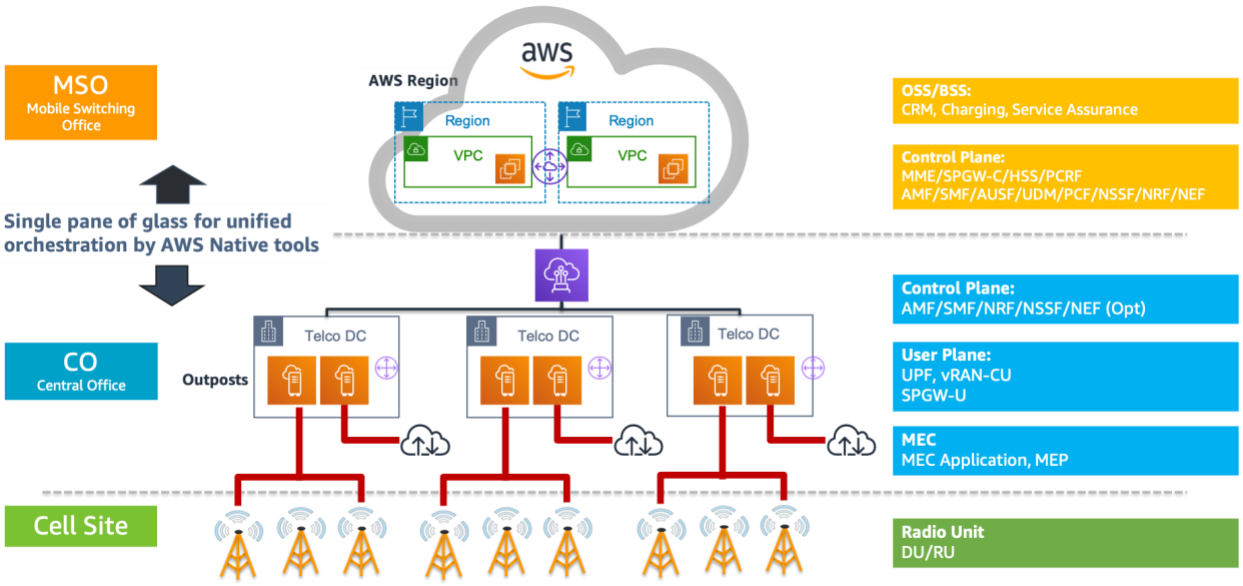


Figure 11 – 5G Network Deployment Hierarchy - vertical view

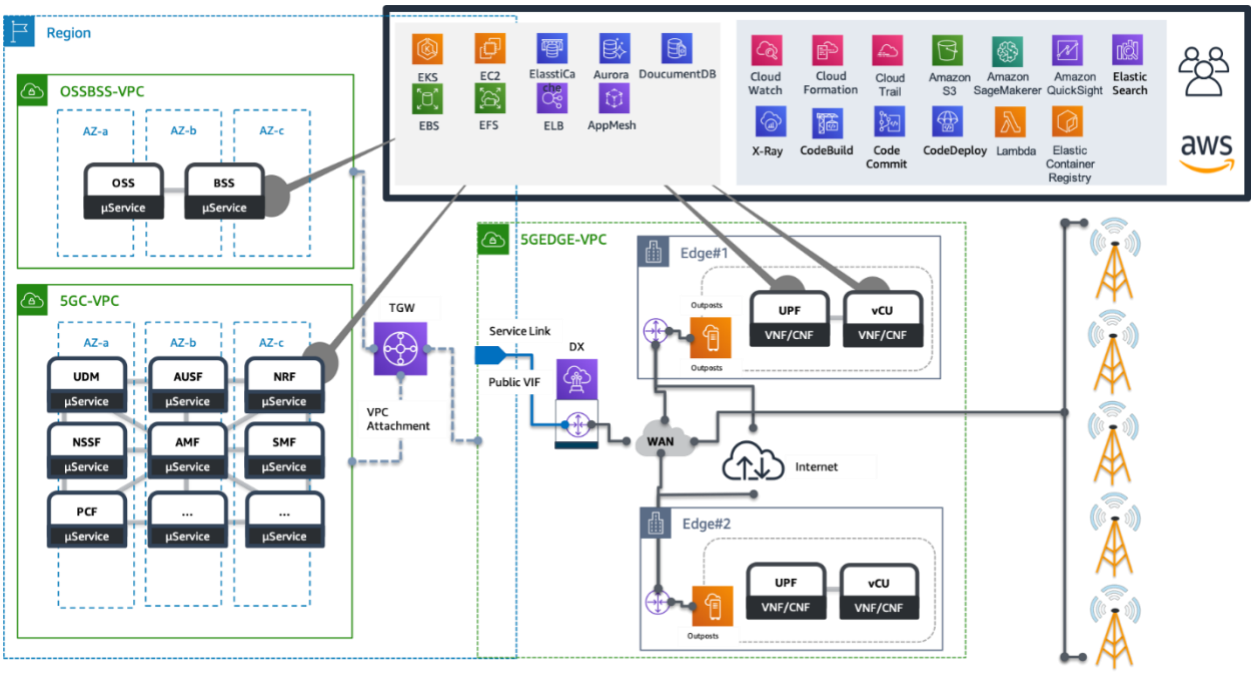


Figure 12 – Reference architecture of 5G network

This section explains the reference architecture of the 5G core network. The explanation assumes the use of the AWS Well-Architected Framework[12], as show in Figure 12. The Well-Architected Framework was developed to help cloud architects

build secure, high-performing, resilient, and efficient infrastructure for their applications. The framework is based on the following five pillars:

- Operational excellence
- Security
- Reliability
- Performance efficiency
- Cost optimization

The framework provides a consistent approach for customers and APN Partners to evaluate architectures and implement designs that will scale over time.

Maximized high availability using AWS global infrastructure

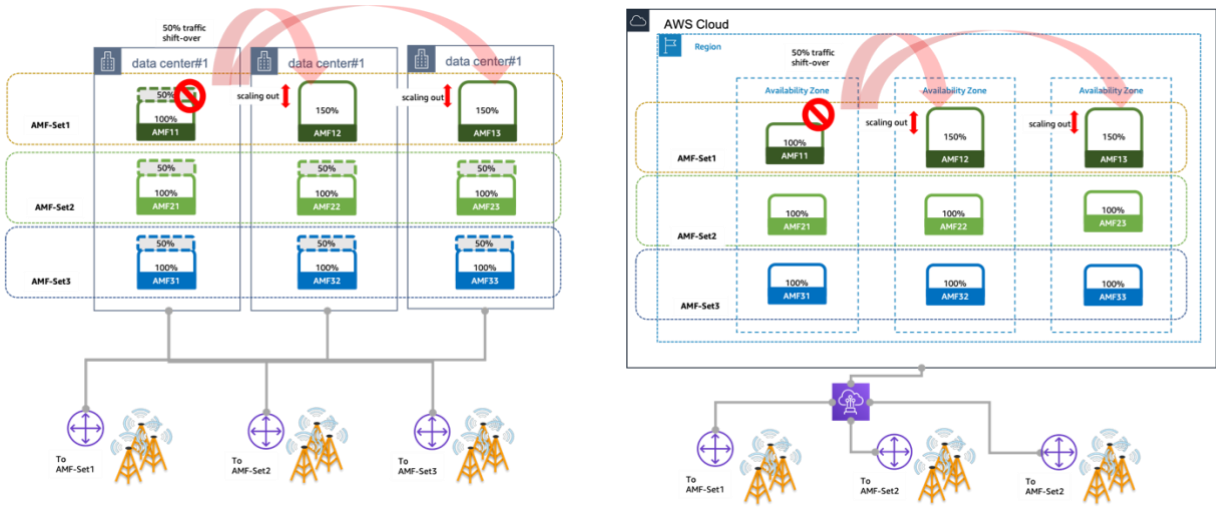
In the implementation of traditional core network design, it is always ensured to have fault-isolation across multiple data centers. For example, Figure13 (a) shows that network functions are created in separate data centers with being grouped as a logical group (such as MME pool, AMF-Set). This means that any network function can take over the traffic from the function which is located at the data center where outage happened.

We can apply the same principle of fault-isolation in AWS deployment. AWS delivers the high network availability. Each Region is fully isolated and comprised of multiple Availability Zones, which are fully isolated partitions of our infrastructure. An Availability Zone is one or more discrete data centers with redundant power, networking, and connectivity in an AWS Region. All Availability Zones in a Region are interconnected with high-bandwidth, low-latency networking, over fully redundant, dedicated metro fiber providing high-throughput, low-latency networking between Availability Zones.

Availability Zones make partitioning applications for high availability easy. If an application is partitioned across Availability Zones, companies are better isolated and protected from issues such as power outages, lightning strikes, tornadoes, earthquakes, and more. Availability Zones are physically separated by a meaningful distance, many kilometers, from any other Availability Zones, although all are within 100 km (60 miles) of each other.

To better isolate any issues and achieve high availability, we recommend that you build a solution on multiple Availability Zones in the same Region. This would be commonly

requested for a 5G core network implementation as shown in the example of Figure13 (b).



(a) Traditional DC design with network redundancy

(b) Example of AMF deployment in AWS using Multi-AZ

Figure 13 – Network Dimensioning

Using multiple Availability Zones to maximize high availability doesn't only mean the separate deployment of NF at each Availability Zone. Multi-AZ can be applied in many different dimensions based on the characteristics of each NF. For example, some representative design patterns can be listed out, but it won't be restricted only to these. Whichever pattern you chose, ensure that Multi-AZs maximize the high availability of the service needed for carrier-grade telecom workload, especially for the core network.

Multi-AZ design pattern 1: Application layer redundancy across Multi-AZs (Pooling, NF-Set, or Geo-Redundancy)

As show in Fig 13 (b) you can use 3GPP's redundancy mechanism such as pooling and NF-Set concept or generic geo-redundancy model after deploying each NF at each AZ. Communicating peers (for example, gNB to AMF) will re-route the traffic to new NF in another AZ against one NF or entire one AZ failure.

Multi-AZ design pattern 2: Spans worker node group across multi-AZ in EKS

In this pattern, you can create a worker node group in the EKS cluster to be spanning across multiple Availability Zones, so that one NF can be working in physically multiple data centers. For this pattern, generic SBA interfaces (HTTP/2) can use AWS Network

Load Balancer which can be created at each Availability Zone with providing local endpoint for each Availability Zone. In the case of the N2 interface, which is anchoring SCTP, we can use multiple Transport Network Layer Association (TNLA) features in the 3GPP 23.501. This means we can create multiple SCTP anchor points at each AZ so that NGAP can get switched over to other alive SCTP connections when one SCTP anchor point or AZ goes unavailable.

Multi-AZ design pattern 3: Sharing common context-store for stateless NF using AWS managed database solutions (DynamoDB)

If an NF is designed in a completely stateless architecture, then we can have multiple microservices at each AZ with making them accessible to a common database. In this case, AWS managed DB solutions especially DynamoDB can help to build up scalable and reliable DB. DynamoDB automatically spreads the data and traffic for your tables over enough servers to handle your throughput and storage requirements, while maintaining consistent and fast performance. All your data is stored on solid-state disks (SSDs) and is automatically replicated across multiple Availability Zones in an AWS Region, providing built-in high availability and data durability.

An example of the reference architecture for the highly-available 5G core network control plane can be illustrated like Figure 14. As shown in the diagram, multiple patterns of Multi-AZ can be used in the mix based on NF-specific requirement and dimensioning requirement, along with the consideration of the cost (which will be discussed in the next section).

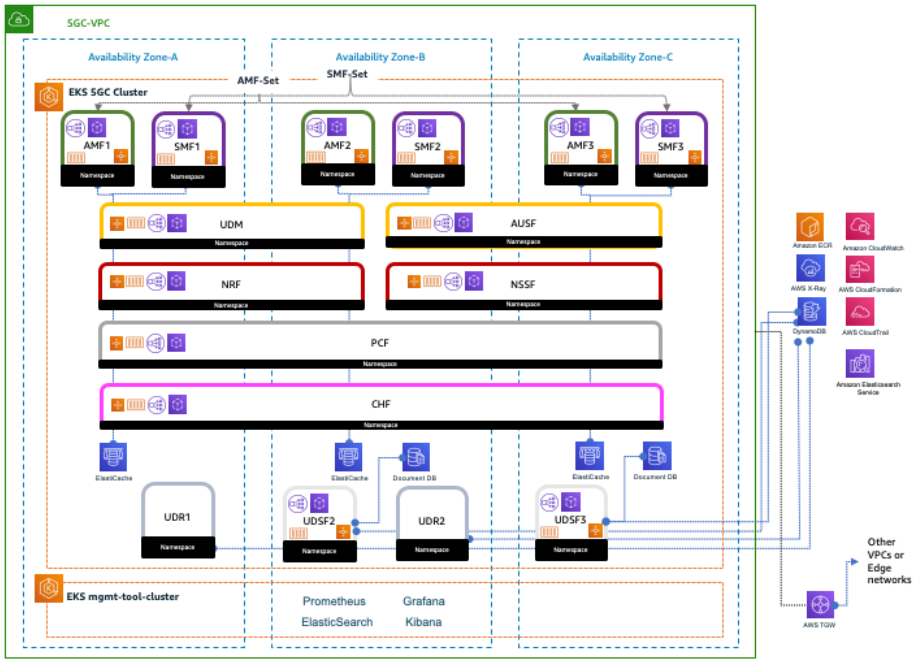


Figure 14 – Reference architecture for 5G core control plane with various Multi-AZ patterns

Optimize performance

In the case of a container-based 5G implementation, it is important to find the balance between scalability, resource utilization, operational complexity, and overall performance to select correct instance types for the worker node. For example, `c5.4xlarge` instance can provide 16 vCPU and 32 GB. Even four `c5.xlarge` instances can provide the same amount of CPU and memory resources in total, but the condition of resource utilization and scalability would be quite a different story. That is, while it gets a complexity of having more worker node instances to manage, scale-out even can be more granular which can give birth to better resource utilization. Therefore, this instance selection should be chosen with fully reflecting characteristics of microservice applications, such as how often scale in/out event happens and instantaneous computing power required.

In general, the smaller size of the instance has an advantage for the signaling processing and event handling process. This is because it requires granular elasticity on the busy hour. On the other hand, the bigger size of the instance or network optimized instances such as `c5n` and `m5n` would have the advantage of user packet processing because user traffic packet processing requires larger unit CPU resource consumption and constant high performance.

In terms of packet processing in user-plane, AWS enhanced networking uses single root I/O virtualization (SR-IOV) to provide high-performance networking capabilities on supported instance types. SR-IOV is a method of device virtualization that provides higher I/O performance and lower CPU utilization when compared to traditional virtualized network interfaces. Enhanced networking provides higher bandwidth, higher packet per second (PPS) performance, and consistently lower inter-instance latencies[13].

Additionally, CUPS deployment using the AWS Outposts can also be regarded as a performance optimization, as well as cost optimization. Using Outposts, we can offload user traffic processing to the edge network by providing low latency for the user's 5G service.

Optimize cost

As shown in Figure 13.a, typical deployment of core network in the on-premises data center requires redundant servers and network equipment to cope with sudden traffic takeover from another data center. These resources usually remain underused in normal operation. Therefore, they can be regarded as overbooking of network capacity.

In the example shown in Figure 13 (a), every data center must have 150% of network capacity to deal with single-point of data center failure. One of the benefits for using AWS to host core network is that you don't have to overprovision capacity, because physical resources are already prepared in the cloud. As shown in Figure 13 (b), in AWS, you only need to set up NF at each Availability Zone or Region with the size just required to cover logical or geographical area (for example, TA: Tracking Area), with having a well-engineered value of a maximum number of instances in the Auto Scaling Group (ASG) so that the worker node group can scale out to serve shift-over traffic against any sudden outage at another NF or Availability Zone. In the case of the database, if you choose to use Amazon DynamoDB, it scales out as required dynamically which also eliminates the traditional dimensioning challenge of overbooking.

AWS also provides many tools for optimizing EC2 instances, such as AWS Trusted Advisor, AWS Cost Explorer, and AWS Compute Optimizer. These tools help CSP operators and NFV providers to select optimal instance plan (for example, AWS Savings Plan and Amazon EC2 Reserved Instance) and type for cost-saving.

As part of networking cost optimization, think about user-plane offload at edge network using the AWS Outposts, which can bring down data-transfer-out cost compared to the region cloud especially for user-plane traffic. Note that data transfer to and from the

Outpost that is hosting UPF to the local network (going to gNB) or to the internet via the Local Gateway (LGW) incurs no charge.

Automate the process with AWS Orchestration and DevOps tools

For the operational excellence, well-architected application on AWS has to provide an automation through operations as code. AWS CloudFormation is fundamental building block to create version-controlled templates for your infrastructure. You should be able to set up Continuous Integration/Continuous Deployment (CI/CD) pipelines using the AWS Developer Tools (for example, AWS CodeCommit, AWS CodeBuild, AWS CodePipeline, AWS CodeDeploy, and AWS CodeStar). In the NFV domain, this AWS native automation can be integrated into MANO framework and CSP's service-orchestration framework, which include:

- **Infrastructure deployment** – CDK and CloudFormation initiate VPC creation, subnets creation, EFS creation, EKS cluster creation, and other required all compute, storage, networking layer creation. Most of these steps of deployment configuration will be one time and will not be touched.
- **CNF deployment** – This involves kubectl, Kubernetes operator, or Helm execution, which can be automated by Lambda, Step Function, and CodePipeline of AWS services.
- **Continuous changes and deployment** – These will be a sequence of steps that will be carried out iteratively to deploy changes coming as a part of container/configuration changes resulting in upgrades. Similar to the CNF deployment case, this can be automated by AWS services with the trigger from CodeCommit, ECR, or 3rd party source systems such as GitLab Webhook[14].

Further details about automation, orchestration, and DevOps implementation are introduced in the following Orchestration and Automation section.

Ensure security with shared responsibility model

A shared responsibility model of security between AWS and user is also applied to 5G network implementation on AWS. Especially in the EKS based 5G implementation, the shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. For Amazon EKS, AWS is responsible for the Kubernetes control plane, which includes the control plane nodes and `etcd` database. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs.
- **Security in the cloud** – Your responsibility includes the following areas.
    - The security configuration of the data plane, including the configuration of the security groups that allow traffic to pass from the Amazon EKS control plane into the customer VPC.
    - The configuration of the worker nodes and the containers themselves.
    - The worker node guest operating system (including updates and security patches).
    - Other associated application software:
        - Setting up and managing network controls, such as firewall rules
        - Managing platform-level identity and access management, either with or in addition to IAM
    - The sensitivity of your data, your company's requirements, and applicable laws and regulations

For more detail of security implementation on EKS environment can be referred from the EKS Security guide[15].

Radio Access Network evolution with AWS

In this section, we address key characteristics of 5G NG-RAN and introduce the best matching AWS services that fit to each of key characteristics.

Control and user plane separation (CUPS)

Along with Core Networks' CUPS architecture in the former section, 5G Next Generation Radio Access Network (NG-RAN) also supports CUPS architecture. While decomposition of CU and DU makes traditional BBU to be more easily transformed to a virtualized network function or containerized network function, CU is split into two parts as defined in 3GPP TS 38.401: CU-CP (Central Unit Control Plane) and CU-UP (Central Unit User Plane). The use of 5G NG-RAN Fronthaul (CPRI/eCPRI), Midhaul

(F1-C, F1-U interfaces), and Backhaul (NG interfaces) powered by AWS and the partners will allow CSP distribute CU and DU functions to Telco Edge data centers using AWS Outposts. As a result, it can bring the benefit of elasticity, scalability, and cost-efficiency in terms both of opex and capex.

Network slicing

To get end-to-end network slicing capability, RAN network slicing has to interwork with core network slicing. Since AWS provides a single pane of glass for unified management using AWS services and 3rd party orchestrator, where CU, DU, and UPF are deployed on AWS Outposts, and control plane of Core networks are in the Region. As shown in Figure 15, RAN network slicing creation is triggered by S-NSSAI (Single Network Slice Selection Assistance Information) and elastic DU, CU-UP, and UPF selection is possible on top of Outposts to meet service requirements such as high bandwidth or low latency. A variety of AWS programmable services related to orchestration and management such as AWS Lambda, AWS Config, CloudFormation, Step Function, as well as CDK, can help with network slicing.

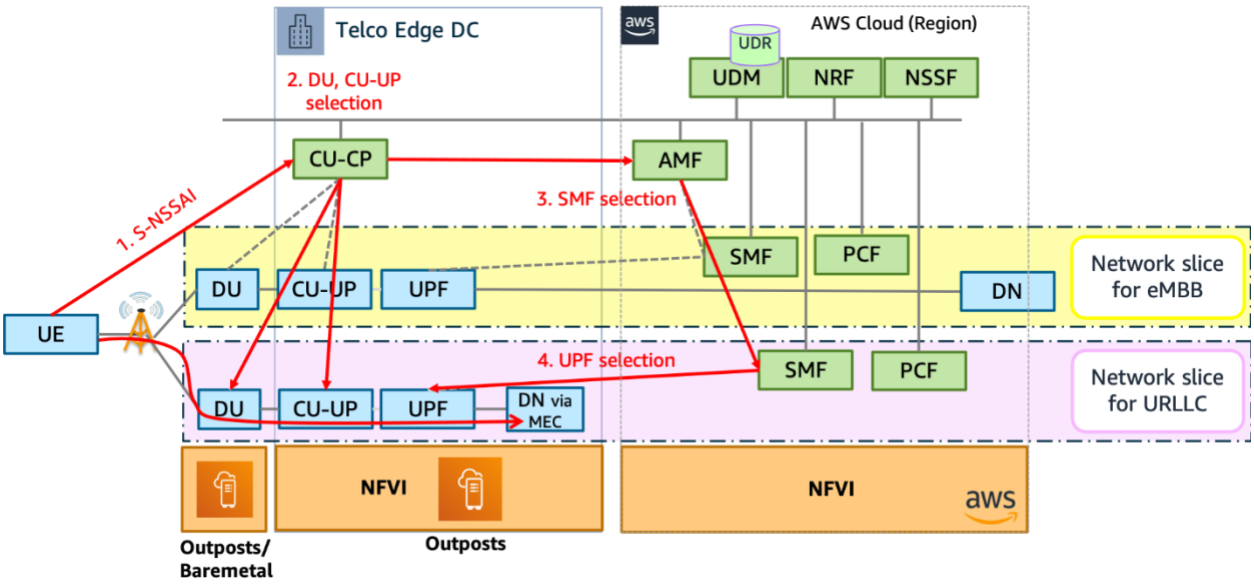


Figure 15 – Network Slice creation on AWS

Multi-access edge computing on AWS

Users of the multi-access edge computing (MEC) framework require computing capabilities at the edge of the network. Per ETSI, this environment is characterized by

ultra-low latency and high bandwidth as well as real-time access to radio network information that can be leveraged by applications[16].

AWS Partners with CSPs to provide AWS Wavelength, which enables developers to build applications that deliver single-digit millisecond latencies to mobile devices and end-users. This is ideal for developers wanting to benefit from using the AWS ecosystem directly. AWS also provides services for CSPs to build their own MEC capabilities and offer it to customers at edge locations, on-premises as well as in AWS regions.

Wavelength

AWS developers can deploy their applications to Wavelength Zones, AWS infrastructure deployments that embed AWS compute and storage services within the telecommunications providers' data centers at the edge of the 5G networks, and seamlessly access the breadth of AWS services in the Region. This enables developers to deliver applications that require single-digit millisecond latencies such as game and live video streaming, machine learning inference at the edge, and augmented and virtual reality (AR/VR).

AWS Wavelength brings AWS services to the edge of the 5G network, minimizing the latency to connect to an application from a mobile device. Application traffic can reach application servers running in Wavelength Zones without leaving the mobile provider's network. This reduces the extra network hops to the Internet that can result in latencies of more than 100 milliseconds, preventing customers from taking full advantage of the bandwidth and latency advancements of 5G.

Wavelength delivers a consistent developer experience across multiple 5G networks around the world, allowing you to build the next generation of ultra-low latency applications using familiar AWS services, APIs, and tools. You can simply extend your Amazon Virtual Private Cloud (VPC) to one or more Wavelength Zones and then use AWS resources like Amazon Elastic Compute Cloud (EC2) instances, Amazon Elastic Block Storage (EBS) volumes, AWS Elastic Container Service (ECS), Amazon Elastic Kubernetes Services (EKS), AWS Identity and Access Management (IAM), AWS CloudFormation, and AWS AutoScaling to build, run, secure, manage, and scale your applications. With just an AWS account, users can deploy 5G applications in Wavelength Zones and seamlessly connect to applications and services in AWS Regions. Wavelength offers the flexibility to scale up or scale down, and pay only for the resources that are used. Fig 16 shows an example of applications being deployed at the

edge with AWS Wavelength connecting with components in the AWS Region as well as other services like S3 or even the internet through the Carrier network.

MEC on AWS

Customers can deploy virtual machines, containers, managed databases and other infrastructure at edge locations, on-premises and in the AWS Regions. In the AWS Regions, EC2 provides secure, resizable compute capacity in the cloud. Amazon Elastic Kubernetes Service (Amazon EKS) is a fully managed Kubernetes service. You can choose to run your EKS clusters using AWS Fargate, which is serverless compute for containers. EKS integrates with AWS App Mesh and provides a Kubernetes native experience to consume service mesh features and bring rich observability, traffic controls and security features to applications. Additionally, EKS provides a scalable and highly-available control plane that runs across multiple Availability Zones to eliminate a single point of failure.

Amazon Virtual Private Cloud (Amazon VPC) lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways.

With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services or call it directly from any web or mobile app.

There are several methods to deploy EC2 instances, EKS pods, containers, and networks for Mobile edge compute use-cases, discussed in the Orchestration in AWS section below. Additionally, the AWS developer tools allow customers to host code, build, test, and deploy MEC applications. AWS CodeStar enables you to quickly develop, build, and deploy applications on AWS. AWS CodeStar provides a unified user interface, enabling you to easily manage your software development activities in one place. With AWS CodeStar, you can set up your entire continuous delivery toolchain in minutes, allowing you to start releasing code faster. AWS CodeStar makes it easy for your whole team to work together securely, allowing you to easily manage access and add owners, contributors, and viewers to your project at no charge. The continuous delivery chain can be extended to on-premises deployments and non-AWS Cloud based Kubernetes instances using AWS Systems Manager (SSM).

AWS Outposts is a fully managed service that extends AWS infrastructure, AWS services, APIs, and tools to virtually any data center, co-location space, or on-premises facility for a truly consistent hybrid experience. AWS Outposts is ideal for workloads that require low latency access to on-premises systems, local data processing, or local data storage. AWS Outposts offers you the same AWS hardware infrastructure, services, APIs, and tools to build and run your applications on premises and in the cloud for a truly consistent hybrid experience. AWS compute, storage, database, and other services run locally on Outposts, and you can access the full range of AWS services available in the Region to build, manage, and scale your on-premises applications using familiar AWS services and tools.

Outposts can be used to setup MEC architecture on-premises and setup applications at edge locations and data centers. Similarly, the same MEC infrastructure can be setup in AWS Regions, with the ability to burst or failover across environments to meet the requirements of 5G customers.

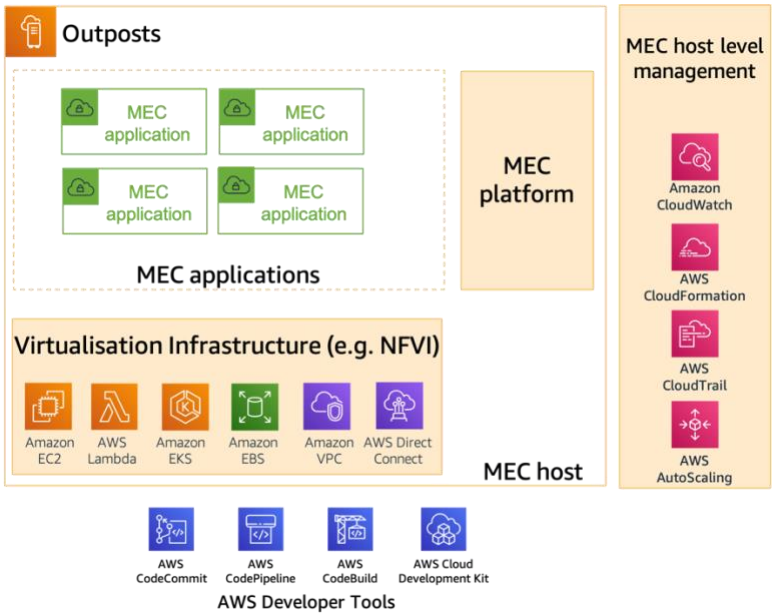


Figure 16 – Multi-access Edge Computing (MEC) framework on AWS (ref. ETSI GS MEC 003)

Based on ETSI GS MEC 003[17], EC2 or EKS could be used as the MEC host in the AWS Region or on-premises with AWS Outposts at the Edge as shown in Figure 16. These could also be deployed on the Edge with AWS Local Zones or AWS Wavelength.

AWS Local Zones are a new type of AWS infrastructure deployment that places AWS compute, storage, database, and other select services closer to large population, industry, and IT centers where no AWS Region exists today. With AWS Local Zones, you can easily run latency-sensitive portions of applications local to end-users and resources in a specific geography, delivering single-digit millisecond latency for use

cases such as media and entertainment content creation, real-time gaming, reservoir simulations, electronic design automation, and machine learning.

With Lambda serverless MEC applications can be deployed in the AWS Region or the one edge. Lambda can be run on many edge devices with AWS IoT Greengrass. AWS IoT Greengrass extends AWS to edge devices so they can act locally on the data they generate, while still using the cloud for management, analytics, and durable storage. With AWS IoT Greengrass, connected devices can run AWS Lambda functions, Docker containers, or both, execute predictions based on machine learning models, keep device data in sync, and communicate with other devices securely – even when not connected to the internet. Additionally, Lambda Edge is an extension of AWS Lambda, a compute service that lets you execute functions that customize the content that CloudFront delivers.

AWS provides multiple services covered in the in the Orchestration section below that address the MEC host level management functionality.

Orchestration and automation in AWS

Management and orchestration

One of the most common challenges of CSP in the era of NFV and 5G is management and orchestration across all networks and infrastructure resources as well as end-to-end service configuration. In case of 5G mobile network, network slicing will be leveraged for creating dedicated network for enterprise and specific service use cases. This means that orchestration will have to solve another layer of complexity. Throughout various industries, AWS platform has been already proven in providing various toolsets for efficient orchestration and management of highly scalable resources on AWS as well as workloads in the on-premises environment. This section describes the mapping of AWS services to the ETSI NFV framework which is predominantly used in the telecom industry. It also outlines implementation approaches for building telecom orchestrator on AWS with focusing on key benefits of using AWS based orchestration tools.

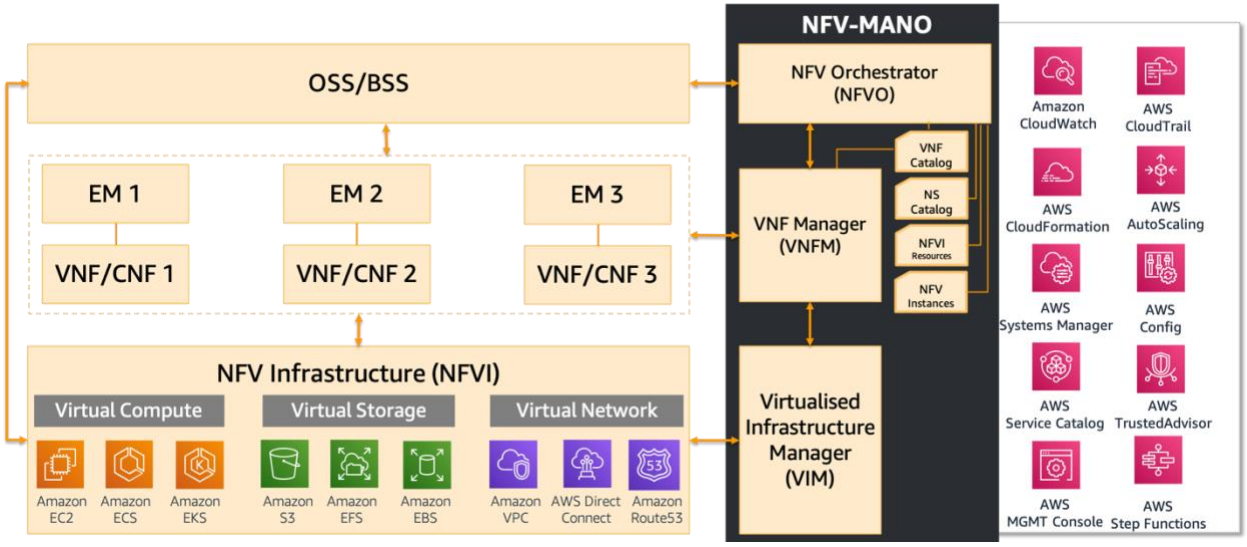


Figure 17 *– AWS Services and ETSI NFV framework mapping (ref. ETSI GS NFV-MAN 001)*

As shown in Figure 17, ETSI NFV defines the NFV-MANO (Management and Orchestration). In the MANO stack, there are components such as:

- VIM (Virtualized Infrastructure Manager),
- VNFM (Virtual Network Function (VNF) Manager),
- NFVO (NFV Orchestrator).

VIM provides NFV Infrastructure (NFVI) resource management function while VNFM manages the lifecycle and Day 0 configuration of VNFs. NFVO provides a mapping between virtual resources and physical resources as well as service chaining of multiple VNF functions when required. In the AWS Cloud, heavy lifting of underlying infrastructure management (i.e. physical resource management) is done by AWS, which means VIM and NFVI are provided by AWS as a service and virtual resource. Furthermore, the abstraction functions of NFVO and VNFM can be also provided by AWS as each logical service component. Given the large number of services available in AWS that could add value to NFV-MANO stack, we are providing logical rather than 1:1 mapping.

AWS native management and orchestration

AWS Management Tools provide CSP with capabilities of instantiation and provisioning, configuration management, monitoring, governance control, and resource optimization. All orchestration and management tools in AWS can be leveraged in the manner of mix-and-match approach along with service demand and requirement, which provides

remarkable flexibility and programmability of orchestration to the CSP. The most commonly used AWS management services are:

- Resource Provisioning: Cloud Formation, Auto Scaling, Service Catalog.
- Configuration Management: Systems Manager, Batch.
- Monitoring and Performance: CloudWatch
- Governance and Compliance: CloudTrail, Config
- Resource Optimization: Trusted Advisor
- Container Orchestration: ECS, EKS, Fargate

Extended Orchestration Capabilities using AWS tools

Along with basic management services in AWS, CSP can implement more advanced and sophisticated orchestration tools using other AWS building blocks such as serverless and AI/ML services.

- **Service Orchestration**: Using AWS Step Functions and Lambda, CSP can implement end-to-end service orchestration for VNFs/CNFs running on AWS as well as VNFs in the on-premises environment.
- **Closed-loop orchestration with analytics**: With using S3 as a data lake for KPIs/alarm/events from each VNF and CNF, CSP can build data analytics using AWS services such as Glue, QuickSight, Athena, and Kinesis. Further to this, data lake and analytics can then be provided as input to machine learning services such as QuickSight(Insight) or SageMaker so that proactive feedback can be provided to the orchestration layer using Lambda.

Application-based Orchestration by API interworking

In the spirit of always providing the choice to our CSP customers, an alternative to cloud native approach using AWS tools as described above, is to implement the MANO layer on AWS by porting 3rd party partner's NFVO and G/S-VNFM (Generic or Specific VNFM) into AWS platform through API integration. This integration can be considered in three different points:

- **NFVO-AWS interface**: if selected NFVO uses direct mode (direct interface from NFVO to VIM), then AWS EC2 and other service API should be called by NFVO directly. Also, the NFVO supplier can implement an adapter whose northbound APIs are ETSI NFV SOL003 (NFVO - S-VNFM) or SOL002 (EM, VNF/CNF - G-VNFM) compliant and the southbound interworks with AWS.
- **VNFM-AWS interface**: if given NFVO uses indirect mode, then VNFM should be able to interwork with AWS management using AWS API calls. In this case, AWS will be likely regarded as just another VIM and NFVI layer.
- **VNF/CNF-AWS interface**: if the EM (Element Management) module of VNF/CNF can interwork with AWS management through API calls (e.g. in case of scaling-in/out), then we may be able to eliminate or minimize the function of VNFM with having a direct interface between VNF/CNF and AWS. Kubernetes based application generally doesn't require "VNFM like" middle layer management since Kubernetes itself manages the lifecycle of containers and CNFs, with directly interfacing NFVO.

When 3rd party partners implement their own orchestrator on AWS, it will be more beneficial to leverage AWS services natively and develop using container services (ECS/EKS), service mesh, serverless (Lambda), and API Gateway, as well as CI/CD DevOps tools. This approach will result in creation of orchestrator that is fully built based on microservice-based flexible architecture. This is a critical aspect because the complexity of the 5G orchestration tool will continue to increase as new service cases get added. Microservice-based architecture will ensure that orchestrator can be further developed and enhanced in agile and flexible manner.

Automation and DevOps

Automation of CNF lifecycle management

Given that 5G networks would increase complexity in terms of heterogeneous network environment as well as microservice-based architecture, CNF lifecycle management should be fully automated to maximize the efficiency of operation in scale and minimize the cost. Figure 18 shows how to automate 5G CNF deployment and lifecycle management.

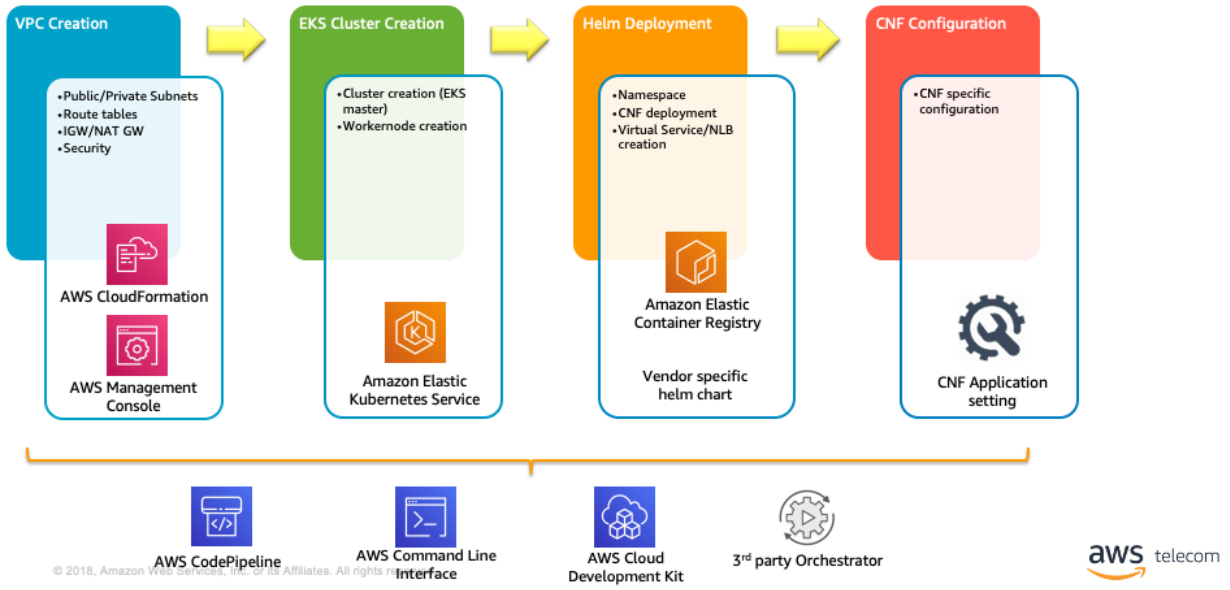


Figure 18 – General procedures for 5G deployment on AWS

CloudFormation, CodePipeline, CLI, or CDK enables to create VPC environment including public and private subnets, route table, IGW/NAT GW, and security group configuration. Amazon EKS helps managed or self-managed cluster creation and Lambda-backed custom resource in CloudFormation template triggered by CloudWatch event can assign EKS worker nodegroup with multiple interfaces. And, helm deployment configures the namespace, load balancer creation, and CNF onboarding using Amazon ECR or vendor-specific helm chart. CNF-specific configuration can be set by CodePipeline, AWS CLI, or CDK. In case EKS cluster worker node needs to get increased during the lifecycle management after instantiation, CloudWatch event triggers AutoScaling. 3rd party partner's orchestrator can control the entire procedure through AWS API interworking.

The important consideration is to ensure that the application, infrastructure, and pipelines are monitored continuously. The applications when deployed by the pipeline are tested during the deployment time, however, it is important to configure observability to track the runtime health of the application PODs using various metrics. Logs and metrics can be monitored with combination of FluentD, Prometheus, CloudWatch agent with Container Insights running on EKS Cluster and making the metrics available on the CloudWatch dashboard. Various alarms and event rules should be configured to trigger notification OR remediation action on the event of any Service getting impacted on

runtime. The remediation action can be either done via Pipelines or from the EKS orchestration tool.

Automation of DevOps pipeline

The Cloud-based Continuous Deployment Pipelines enable the CSP to gain a competitive edge with agility in rolling out new services and functionalities as Virtualized Network functions on AWS Infrastructure. What used to take multiple weeks for truck-rolls to deploy a service at CSP can now be orchestrated in minutes using Deployment Pipelines. In this section, we will discuss various considerations of building the pipelines and achieve an end-end automation.

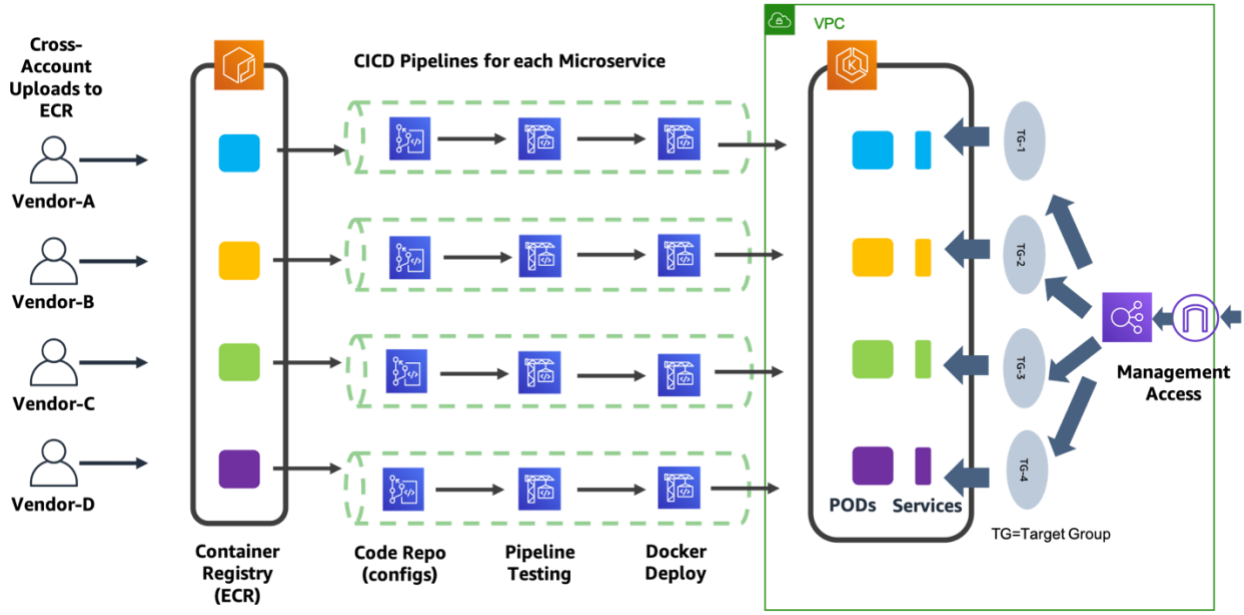


Figure 19 – Example of deployment pipelines for multi-vendor environment

The following sections detail the design criteria for continuous deployment pipelines.

Automation

In case of a CSP, the network would be built out of multiple instances of CNF microservices, supplied from various vendors as binary docker images. These images will regularly undergo updates from individual vendors to release new features and thus would need to be deployed as updates into the network environment. Figure 19 shows a typical deployment for a CSP to automate the CNF deployment updates. Multiple CD pipelines are deployed per individual ISV, that are kicked-off with the upload of updated Docker images or configuration (helm charts, YAML files). Each respective pipeline will run through various stages for vetting the updates by first deploying it on test

environments for focused feature testing, later in the staging environment for system-level testing and eventually in production using blue/green canary-based deployments.

Above Figure 19. shows AWS CodePipelines configured with the Source stage comprising of AWS CodeCommit as configuration repository and Amazon ECR for container registry whereas Test and Deployment stages using AWS CodeBuild service.

Security

Security checks are kicked off at various stages of the pipeline to ensure the newly uploaded image is secure and complies with the desired compliance checks. The container registry would scan for any open CVE vulnerabilities, the configuration checked against leaking out any sensitive information (known PII pattern), test stage triggering compliance check rules (for example, unexpected open TCP/UDP ports, DOS vulnerabilities) and eventually verifying backward and forward compatibility for graceful upgrade/rollback safety. Apart from the application, it is critical to provision pipeline security by ensuring encrypted transfer of artifacts across stages, whether at rest or transit. Additionally, the resource permissions should be restrictive enough to ensure that no accidental or unintended updates occur. For example, the test stage of the pipeline should be limited to test environment resources.

It is important to provision for an audit mechanism to facilitate investigation of past events. AWS CloudTrail offers this visibility, recording every API calls across services. AWS Config offers capability for compliance validation.

Observability

For a CSP, monitoring is one of the critical elements of a solution, especially managing network at large scale. It is recommended to setup monitoring via metrics, logs, events for the application, infrastructure and pipelines, using various services like Amazon CloudWatch, AWS X-Ray, AWS CloudTrail.

Infrastructure as Code

It is important to maintain the blueprint of the infrastructure in the form of code. This allows for deterministic reproduction of the infrastructure with the same expected behavior, when needed. The code is maintained in the code repository and a pipeline setup to orchestrate updates to the deployed stacks (for example, AWS Cloud Development Kit (CDK) and AWS CloudFormation). Because multiple pipelines will be provisioned for individual vendors (ISVs), it's important to build templates of IaC for agile onboarding of ISV functions. These can be split into:

- **day0 template** – base cluster infrastructure

- **day1 template** – application-specific deployment and configuration
- **day2 template** – pipelines to carry out iterative deployment steps

AWS CloudFormation service enables you to describe and provision all the infrastructure resources in your cloud environment in JSON or YAML templates. Not only for AWS resources, AWS CloudFormation provides a powerful extension mechanism through AWS Lambda-backed custom resources, you can write your own resources to extend AWS CloudFormation beyond AWS resources and provision the required resource. For example, you can integrate a third-party software as a service (SaaS) product, or you can even provision on-premises resources in hybrid environments. Another great use case is to extend AWS CloudFormation by providing utility resources that perform tasks to transform or process properties of your infrastructure[18]. AWS CDK offers developers the ability to build out code using higher level familiar programming languages like Python, TypeScript, JavaScript, Java, C#, and then compile the code into lower level CloudFormation JSON format that can then be deployed.

Deployment

We recommend that you configure for blue/green and canary-based deployments in test as well as in production environments[19]. Blue/green deployments allow you to test new application version in a contained environment. They also provide an easy and graceful method to switchover production traffic. Canary-based deployments extend this concept by enabling the non-production green environment to be tested with a small proportion of production traffic to uncover any issues caused due to the nature of production traffic. Thus, the new application version is tested against internal simulated test traffic as well as small proportion of production traffic, which offers more confidence to the users before switching over the production traffic.

This idea is shown in Figure 20. Automation can be achieved by configuring AWS CodePipeline with the blue/green and canary-based deployment stages. The approval stage shown above may be manually driven initially during provisioning, however later should be fully automated. In the test environments, it is a good practice to always test with a rollback action to validate forward and backward compatibility, before deploying into production. The blue/green deployment on clusters with service mesh would depend upon the support provided by the end-application as well as the routing gateway for the service mesh for a graceful transition. Stateless applications are supported for blue/green canary-based deployments, however, stateful applications may need to integrate added functionality to support it. One other factor to consider in deployments is the number of deployment targets. For a CSP, deploying to 100’s of EKS clusters, the

pipeline should provide a capability to carry out parallel deployments (fanning out) to multiple targets.

Workflow

As shown in Figure 21, CNF deployment procedure starts in after a CNF provider drops an image at CSP's ECR (from their DevOps pipeline). The flow of the actions within a pipeline is depicted in the below figure.

By using AWS infrastructure for core and edge network, the CSP can have a single-pane of glass for orchestration which is fully programmable and can be fully automated with a wide range of AWS services and tools.

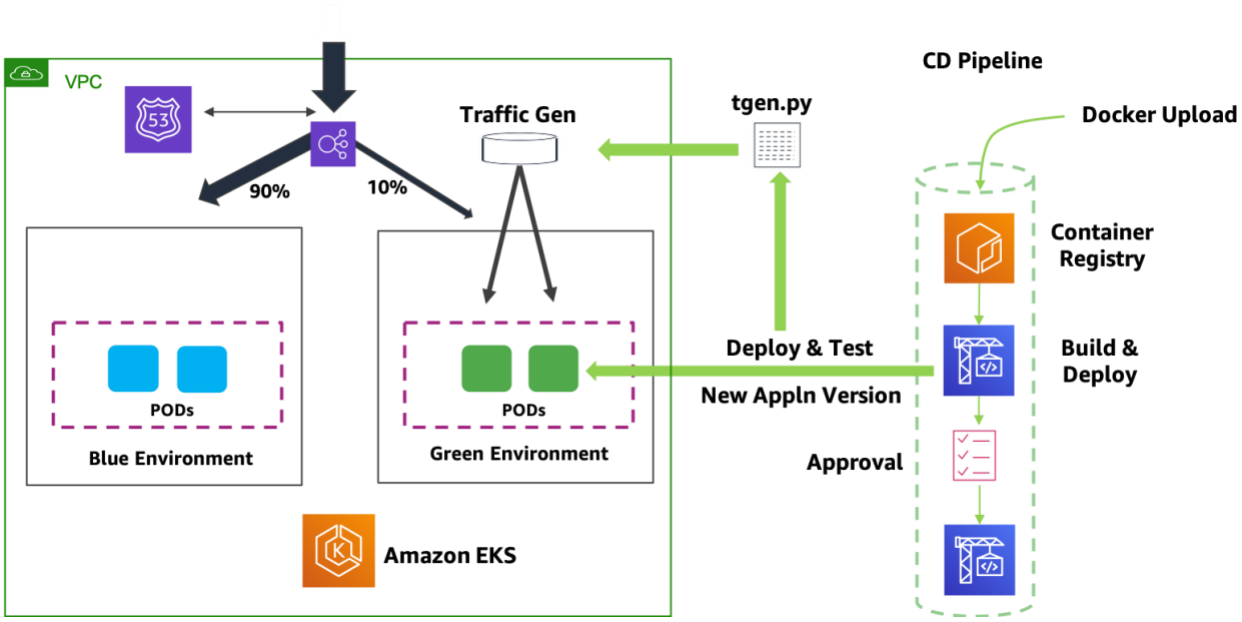


Figure 20 – Blue/Green Canary Deployments

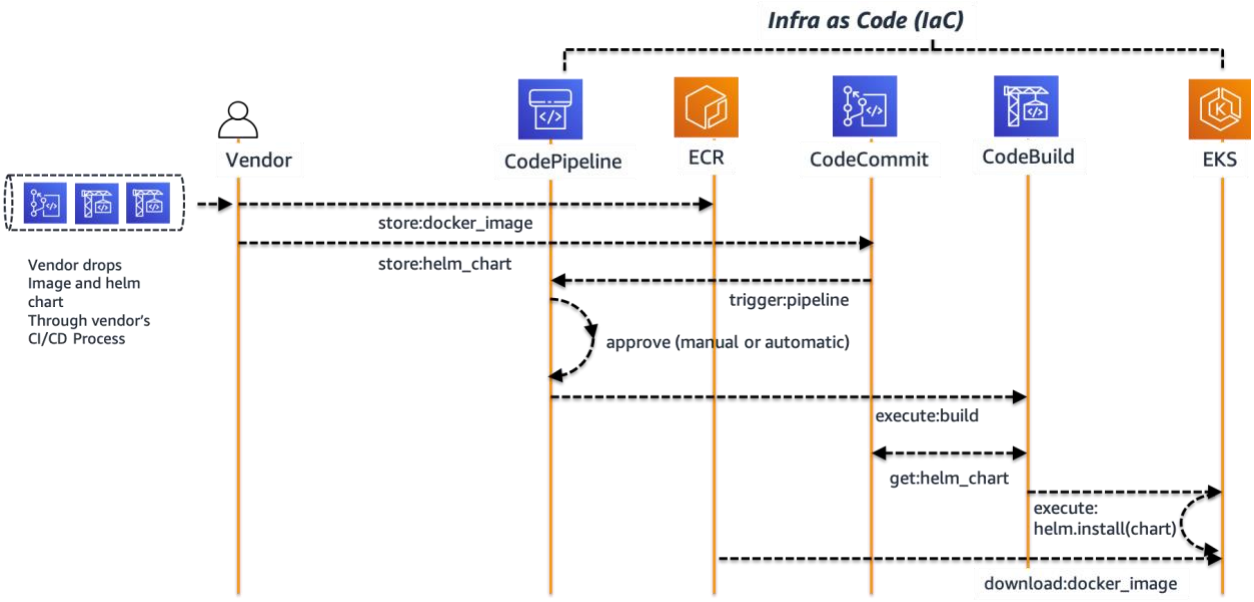
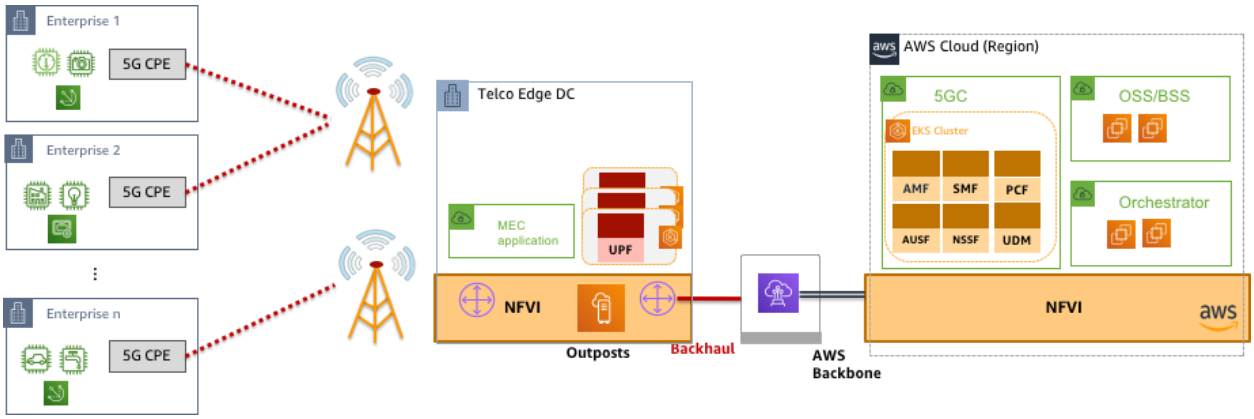


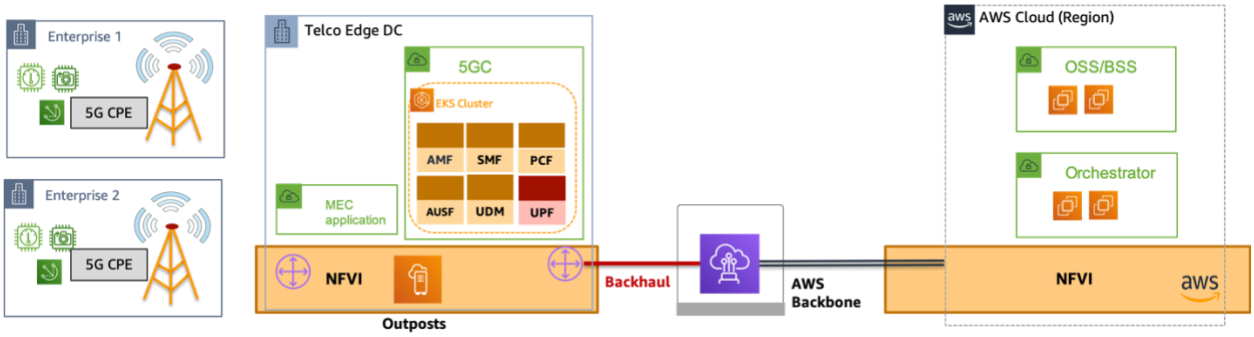
Figure 21 – Example of CI/CD pipeline with using AWS DevOps tools

Private/Local 5G network use case

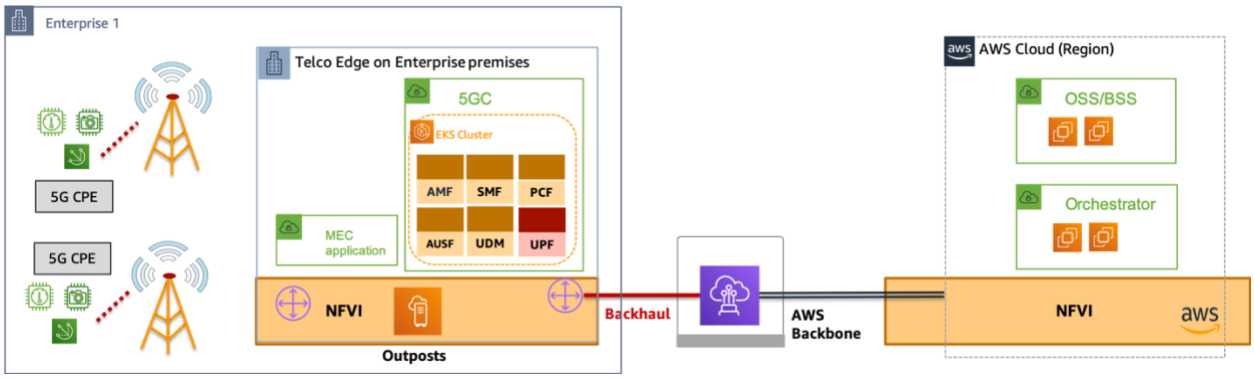
One of potential 5G use cases is private/local 5G networks, mainly for Greenfield B2B business. For private/local 5G networks, the following three deployment models can be considered:



(a) Hybrid deployment model



(b) Local deployment model for multiple sites



(c) Local deployment model for single enterprise, single site

Figure 22 – Private 5G Network Deployment Model

Figure 22 (a) shows hybrid deployment model, where AWS Outposts is deployed in Telco Edge DC, on which UP (User Plane) serving multiple Enterprises or sites, possibly with network slicing for separation of SLA, security requirements etc. Also, MEC application(s) are deployed on the Outposts. 5G Core control plane (UDM, AMF, SMF, etc.), Orchestrator, and OSS/BSS run in an AWS Region.

Figure 22 (b) shows local deployment model for multiple Enterprises or sites where Telco Edge DC deployment contains complete 5GC on Outposts serving single or multiple Enterprises, single or multiple sites. Also, MEC application(s) are deployed on the Outposts. Orchestrator is handled an AWS Region.

Figure 22 (c) shows local deployment model for single enterprise and single site, where Telco Edge on Enterprise premises deployment contains complete 5GC on Outposts serving single Enterprise, single site. Also, MEC application(s) are deployed on the Outposts. Orchestrator is handled an AWS Region.

For hybrid deployment model, there are two approaches to realize network resource isolation:

- Dedicated UP geared by CUPS and isolation by Access Point name (APN) separation
- Network slicing by S-NSSAI. For a local deployment model, resource isolation can be by either the entire 5G core network dedication using public land mobile network (PLMN) IDs or UP dedication by APN separation.

Conclusion

AWS provides capabilities for you to deploy and leverage 5G infrastructure globally to attain scalability, elasticity as well as high availability, all while meeting key requirements. Several customers are using AWS, APN Partners, and open-source solutions to host mobile workloads on AWS. This has resulted in reduced cost, greater agility, and a reduced global footprint. Especially for partner solutions, AWS has the broadest and strongest partners in the ecosystem available through AWS Marketplace and the APN Partner Central for each part of the stack presented in this paper.

The reference architectures and best practices provided in this whitepaper can help you successfully set up 5G workloads on AWS and optimize the solutions to meet end user requirements, all while optimizing for the cloud. AWS extends its cloud beyond Regions to the distributed edge. This provides CSPs with a choice between AWS Outposts (to implement cloud native user plane) or Outposts and Wavelength to host MEC applications and latency sensitive workloads. Additionally, management and orchestration, as well as network slicing, can be deployed cost effectively, following cloud native architectures and with an easy path to use AI/ML capabilities to create predictive and self-healing networks.

Contributors

Contributors to this document include:

- Young Jung, Ph.D., Senior Partner Solutions Architect, WW Telco Partner, Amazon Web Services
- Tetsuya Nakamura, Senior Partner Solutions Architect, WW Telco Partner, Amazon Web Services
- Tipu Qureshi, Principal Engineer, AWS Premium Support, Amazon Web Services
- Nikunj Vaidya, Senior Solution Architect, WW Specialist SA (DevOps), Amazon Web Services

- Rada Stanic, Principal Solutions Architect, WWCS Geo, Amazon Web Services
- Vebhhav Singh, Principal Solutions Architect, WWCS Sales, Amazon Web Services
- Mahesh Goenka, Principal Partner Development Manager (NFV Portfolio lead), WW Telco Partner, Amazon Web Services
- Kaixiang Hu, Principal Software Development Engineer, EC2 Connectivity Services, Amazon Web Services
- Shane Hall, Senior Software Development Engineer, 5G, Amazon Web Services

Document Revisions

Date	Description
July 2020	First publication.

Glossary

- **AF** – Application Function
- **AMF** – Access & Mobility Management Function
- **AUSF** – Authentication Server Function
- **CHF** – Charging Function
- **CNF** – Cloud-native or Containerized Network Function
- **CU** – RAN Central Unit
- **CU-CP** – CU Control Plane
- **CU-UP** – CU User Plane
- **CUPS** – Control and User Plane Separation
- **DN** – Data Network
- **DU** – RAN Distributed Unit
- **EPC** – Evolved Packet Core

- **MANO** – Management and Orchestration
- **MEC** – Multi-Access Edge Computing
- **NEF** – Network Exposure Function
- **NFV** – Network Function Virtualization
- **NFVI** – Network Function Virtualization Infrastructure
- **NFVO** – Network Function Virtualization Orchestrator
- **NRF** – Network Repository Function
- **NSA** – Non-Stand Alone 5G
- **NSSF** – Network Slice Selection Function
- **PCF** – Policy Control Function
- **RAN** – Radio Access Network
- **RU** – RAN Radio Unit
- **SA** – Stand Alone 5G
- **SBI** – Service Based Interface
- **SCTP** – Stream Control Transport Protocol
- **SMF** – Session Management Function
- **UDM** – Unified Data Management
- **UE** – User Equipment
- **UPF** – User Plane Function
- **VIM** – Virtualized Infrastructure Manager
- **VNF** – Virtual Network Function
- **VNFM** – Virtual Network Function Manager

Notes

[1] 3GPP. Technical Specification 23.501 – System Architecture for the 5G System (5GS)

[2] https://d1.awsstatic.com/whitepapers/carrier-grade-mobile-packet-core-network-on-aws.pdf

[3] https://d1.awsstatic.com/whitepapers/microservices-on-aws.pdf

[4] https://12factor.net/

[5] https://kubernetes.io/

[6] Multus CNI plugin, https://github.com/intel/multus-cni

[7] https://github.com/aws-samples/amazon-eks-refarch-cloudformation

[8] https://aws.amazon.com/premiumsupport/knowledge-center/attach-second-eni-auto-scaling/

[9] https://github.com/aws-samples/cfn-nodegroup-for-multus-cni

[10] https://wa.aws.amazon.com/wat.concept.canary-deployment.en.html

[11] https://docs.aws.amazon.com/outposts/latest/userguide/region-connectivity.html

[12] https://d1.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf

[13] https://d1.awsstatic.com/whitepapers/amazon-ec2-networking-for-telecom.pdf?did=wp_card&trk=wp_card

[14] https://docs.gitlab.com/ee/user/project/integrations/webhooks.html

[15] https://docs.aws.amazon.com/eks/latest/userguide/security.html

[16] https://www.etsi.org/technologies/multi-access-edge-computing

[17] https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf

[18] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-custom-resources.html

[19] https://github.com/aws-samples/amazon-eks-cdk-blue-green-cicd