vCloud NFV Cloud Native Reference Architecture 3.3

VMware vCloud NFV OpenStack Edition 3.3
You can find the most up-to-date technical documentation on the VMware website at:

https://docs.vmware.com/

If you have comments about this documentation, submit your feedback to
docfeedback@vmware.com
About vCloud NFV Cloud Native Reference Architecture

The VMware vCloud NFV Cloud Native Reference Architecture provides guidance for designing and creating the vCloud NFV Cloud Native platform by using Essential PKS.

This vCloud NFV Cloud Native Reference Architecture guide describes the high-level design principles and considerations for implementing the vCloud NFV Cloud Native environment.

Intended Audience

This guide is intended for telecommunications and solution architects, sales engineers, field consultants, advanced services specialists, and customers who are responsible for Virtual Network Functions (VNFs), Cloud-Native Network Function (CNF), and the NFV environment on which VNFs and CNFs run.

This guide leverages the vCloud NFV OpenStack Edition architecture, so you must be familiar with the latest revision of this reference architecture.
Introduction to vCloud NFV Cloud Native Architecture

The vCloud NFV Cloud Native architecture combines a carrier grade NFV infrastructure with VMware® Integrated OpenStack and Essential PKS.

This chapter includes the following topics:

- Acronyms and Definitions

Acronyms and Definitions

vCloud NFV uses a specific set of acronyms that apply to the NFV technology and the Telco industry.

Table 2-1. General Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFD</td>
<td>Bidirectional Forwarding Detection for failure detection on the transport links</td>
</tr>
<tr>
<td>DPDK</td>
<td>Data Plane Development Kit, an Intel-led packet processing acceleration technology</td>
</tr>
<tr>
<td>MTTR</td>
<td>Mean Time to Repair</td>
</tr>
<tr>
<td>MTTU</td>
<td>Mean Time to Understand</td>
</tr>
</tbody>
</table>

Table 2-2. NFV Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCP</td>
<td>Centralized Control Plane in the NSX-T Data Center architecture</td>
</tr>
<tr>
<td>CNI</td>
<td>Container Network Interface</td>
</tr>
<tr>
<td>CNF</td>
<td>Cloud-Native Network Function executing within a Kubernetes environment</td>
</tr>
<tr>
<td>LCP</td>
<td>Local Control Plane in the NSX-T Data Center architecture</td>
</tr>
<tr>
<td>MANO</td>
<td>Management and Orchestration components, a term originated from the ETSI NFV architecture framework</td>
</tr>
<tr>
<td>NCP</td>
<td>NSX Container Plug-in</td>
</tr>
<tr>
<td>NFVI</td>
<td>NFV Infrastructure</td>
</tr>
<tr>
<td>NFV-OI</td>
<td>NFV Operational Intelligence</td>
</tr>
<tr>
<td>N-VDS (E)</td>
<td>Enhanced mode of the N-VDS switch using DPDK and vertical NUMA alignment to accelerate workloads</td>
</tr>
</tbody>
</table>
Table 2-2. NFV Acronyms (continued)

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-VDS (S)</td>
<td>Standard mode of the N-VDS switch</td>
</tr>
<tr>
<td>PSP</td>
<td>Pod Security Policy</td>
</tr>
<tr>
<td>VIM</td>
<td>Virtualized Infrastructure Manager</td>
</tr>
<tr>
<td>VNF</td>
<td>Virtual Network Function executing in a virtual machine</td>
</tr>
</tbody>
</table>

Table 2-3. Telco Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSS</td>
<td>Home Subscriber Server</td>
</tr>
<tr>
<td>K8s</td>
<td>Kubernetes</td>
</tr>
<tr>
<td>MVNO</td>
<td>Mobile Virtual Network Operator</td>
</tr>
<tr>
<td>PCRF</td>
<td>Policy and Charging Rules Function</td>
</tr>
<tr>
<td>PCF</td>
<td>Policy Control Function</td>
</tr>
<tr>
<td>PGW</td>
<td>Packet Gateway in the mobile evolved packet core 4G architecture</td>
</tr>
<tr>
<td>SGW</td>
<td>Service Gateway in the mobile evolved packet core 4G architecture</td>
</tr>
<tr>
<td>SBC</td>
<td>Session Border Controller used in the voice telephone for control and data plane communications between clients</td>
</tr>
<tr>
<td>SMF</td>
<td>Session Management Function</td>
</tr>
<tr>
<td>UDM</td>
<td>Unified Data Management</td>
</tr>
<tr>
<td>UPF</td>
<td>User Plane Function</td>
</tr>
</tbody>
</table>
5G services require a combination of low-latency and high throughput with high user densities and concurrences. The distribution of functional components in the network requires a more sophisticated service delivery model.

The network is transforming into a combination of highly distributed and centralized functions. This way, the network is moving away from the typical centralized models in service delivery. The third-party IaaS, PaaS, and SaaS offerings from public cloud providers are making an emerging paradigm shift in the network.

Figure 3-1. Reference Environment

<table>
<thead>
<tr>
<th></th>
<th>Far Edge</th>
<th>Hub</th>
<th>Near Edge</th>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Edge</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Slices</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Central Office</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Central &amp; Regional</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Millions</td>
<td>1,000's</td>
<td>100's</td>
<td>10's</td>
<td></td>
</tr>
<tr>
<td>40 - 80 µs</td>
<td>2 - 5 ms</td>
<td>&lt; 10 ms</td>
<td>&lt; 20 - 50 ms</td>
<td></td>
</tr>
<tr>
<td>1-3 servers</td>
<td>1 - 5 servers</td>
<td>1 - 5 racks</td>
<td>Multiple racks</td>
<td></td>
</tr>
</tbody>
</table>

The highly distributed topology of the network supports the next-generation service characteristics in distribution and composition. The topology requires a new way of managing the network and infrastructure resources. The number of services that are spanning the industry verticals is expanding exponentially. Today’s endpoints ranging from fixed to mobile grow into billions with IoT connections. The highly distributed edge sites are projected to be in the tens of thousands, regional sites in the 100’s, core sites in the 10’s, and a large variety of public cloud provider sites.

NFV and Software Defined Networking (SDN) transformations introduce complex interconnections and interactions between end-points, such as branches, small offices, connected cars, and IoT gateways, to private data centers and public cloud providers. The reference environment and its transformation are not only a technical challenge but also impacts the business and operating processes.

This chapter includes the following topics:
Key Customer Objectives

The goal of network modernization is to drive greater classes of service innovation and timely enablement. The following key objectives are considered for CSPs as they transform their networks and design for new business and operational models:

Cloud-Native Environments

Cloud-Native approaches are dictating a new NFV paradigm and microservices VNF architectures. Container technology is the new light-weight execution environment for such microservices and delivery. While the fine-grained abstraction of Telco applications might be a good fit for control plane functions in the next-generation architecture, the user plane functions are expected to run as native VM functions. Hence, the cloud infrastructure environment must be heterogeneous enabling hybrid execution environments for native VM and containerized applications.
Cloud Native Data Center Evolution

This section covers a set of solutions and usecase scenarios to modernize the CSP cloud infrastructure environment with vCloud NFV OpenStack Edition.

This chapter includes the following topics:

- Key Stakeholders
- Conceptual Architecture
- Cloud Native Workload
- Cloud Native Logical Architecture
- Kubernetes Security Image and Monitoring

Key Stakeholders

The reference architecture considers the following key stakeholders that are involved in the end-to-end service management, lifecycle management, and operations:

**Cloud Administrator**

Cloud Administrators handle deployment, configuration, and management of the VIM layer. Primary responsibilities include:

- Install, provision, and manage SDDC (vSphere, NSX, vSAN, vRealize Suite).
- Install, provision, and manage VMware Integrated OpenStack Carrier Edition.
- Set up the 2-pod or 3-pod architecture and ensure that appropriate components are deployed in the pods.
- Create VMware Integrated OpenStack Carrier Edition projects and Kubernetes clusters with the required amount of resources (CPU, memory, and storage)
- Create tenant-specific users and RBAC in both VMware Integrated OpenStack Carrier Edition and Essential PKS.
- Container image management including CNF image onboarding, vulnerability detection, access, and replication policy for the container image registry

- Provide and maintain tooling required to bootstrap, operate, and fully manage the lifecycle of Essential PKS clusters.

Cloud administrators are expected to have the background knowledge of OpenStack and Kubernetes administration and strong background knowledge of vSphere. Cloud administrators onboard tenants and manage permissions through users, groups, and project definitions. They are also responsible for OpenStack health monitoring and logging.

**Tenant Administrator**

Tenant Administrators are members of one or more OpenStack projects. They manage Kubernetes clusters and work with cloud administrators to ensure sufficient resources allocation and classes of service for workloads. For more details, see the Multi Tenancy section in Design Objectives.

Tenant administrators create and expand Kubernetes clusters, set up the Kubernetes cluster user access, and deploy CNFs. As part of the Kubernetes cluster management, tenant administrators are also responsible for CNF monitoring and logging.

**Kubernetes User**

Kubernetes Users deploy CNFs using Helm, kubectl command, Kubernetes API, or Kubernetes dashboard.

Kubernetes users do not have OpenStack access and are not aware of how a cluster is constructed. For example, a Kubernetes user can request as part of a container definition for a specific type of device (such as a volume or network interface) without knowing the underlying hardware implementation. The Kubernetes scheduler then places the container to a node that has the hardware type requested by the user. Kubernetes Users are also responsible for CNF monitoring and logging.
Figure 4-1. Key Kubernetes Persona

<table>
<thead>
<tr>
<th>Cloud Admin</th>
<th>Tenant 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Maintains VIM</td>
<td></td>
</tr>
<tr>
<td>• Adding Users to VIM</td>
<td></td>
</tr>
<tr>
<td>• Create Tenants</td>
<td></td>
</tr>
<tr>
<td>• Create Tenant Objects</td>
<td></td>
</tr>
<tr>
<td>• (Compute/Network/Storage and Quota)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tenant Admin</th>
<th>Tenant User</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Kubernetes Cluster CRUD and Management</td>
<td></td>
</tr>
<tr>
<td>• Create Tenant Objects (Compute/Network/Storage)</td>
<td></td>
</tr>
<tr>
<td>• Authorize user access</td>
<td></td>
</tr>
<tr>
<td>• Deploys CNF</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tenant Admin</th>
<th>Tenant User</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Kubernetes Cluster CRUD and Management</td>
<td></td>
</tr>
<tr>
<td>• Create Tenant Objects (Compute/Network/Storage)</td>
<td></td>
</tr>
<tr>
<td>• Authorize user access</td>
<td></td>
</tr>
<tr>
<td>• Deploys CNF</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tenant Admin</th>
<th>Tenant User</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Kubernetes Cluster CRUD and Management</td>
<td></td>
</tr>
<tr>
<td>• Create Tenant Objects (Compute/Network/Storage)</td>
<td></td>
</tr>
<tr>
<td>• Authorize user access</td>
<td></td>
</tr>
<tr>
<td>• Deploys CNF</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tenant User</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Deploy CNF (CRUD Operations)</td>
</tr>
</tbody>
</table>

Conceptual Architecture

Cloud infrastructure-based computing is the next-generation standard in modernizing the CSP networks as they evolve to 5G architectures, services, and agile delivery.

Containers gained immense popularity as a portable and lightweight virtualization solution for 5G Service-Based Architecture (SBA). Kubernetes substrate is only one of the many components to consider when delivering Carrier-Grade Container as a Service (CaaS). A Carrier-Grade CaaS platform requires a complex ecosystem of solutions and functions delivering to a pre-set business and operating model. The cloud infrastructure modernization changes not only the business model in service agility and metered revenue models, but also challenges the silo operating model. The following figure shows the conceptual view of the various domains, capabilities, and their interactions that need consideration in the modernization of networks and business and operational models.
Cloud Native Management

The container management is responsible for the management of the container infrastructure and container lifecycle. Container management consists of cloud native networking, persistent storage, Kubernetes deployment, Kubernetes cluster security, container image registry, and operation management for cloud native workloads.

Cloud Native Workload

This section describes the seamless orchestration and management of cloud native workload.

Scope

VMware vCloud NFV Container platform leverages IaaS design from the latest vCloud NFV reference architecture. By aligning the container platform with the VIM layer best practices, the vCloud NFV infrastructure delivers a myriad of Telco use cases and facilitates next generation service-oriented solution for Telco CNF workloads (such as 5G) in addition to traditional VNF workloads. The vCloud NFV Container platform solution is integrated with holistic operations management and service assurances capabilities, empowering the operator to rapidly deliver services while ensuring quality.

Design Objectives

The vCloud NFV Container platform components can be used in various ways to construct a comprehensive, end-to-end solution that meets the business goal of CSPs. This guide discusses how components can be used to create the vCloud NFV Container reference architecture.

Cloud Native Architecture

Containers gained popularity as a portable and lightweight virtualization solution for 5G Service-Based Architecture (SBA). Kubernetes substrate is only one of the many components to consider when delivering Carrier-Grade Container as a Service (CaaS). The Carrier-Grade CaaS platform requires a complex ecosystem of solutions and functions that offer a set of business and operating models.
The following figure shows the conceptual view of various domains and capabilities:

**Figure 4-3. Cloud Native Architecture Overview**

[Diagram of cloud native architecture]

**Cloud Native Management Platform**

The container management domain is responsible for the management of the container infrastructure and container lifecycle. This domain consists of the following key components.

**Infrastructure Domains**

The infrastructure domain consists of both physical and virtual infrastructure. The CaaS offering is as robust as the infrastructure domain it operates. The infrastructure layer must be highly available and elastic, with a pool of resources that can be allocated and released on demand. The infrastructure domain must accommodate workloads of all types, low latency, stateless, stateful, flexible vNIC, multiple network interfaces, and IPv6/IPv4.

Telco workloads consist of VNFs and containerized VNFs. The infrastructure domain must offer the same level of abstraction across all workload types. SLAs are tenant-specific. The infrastructure domain must provide a consistent tenant abstraction across all workloads.

**Cloud Native Networking**

Container networking is usually the most challenging component to deploy and maintain in a Kubernetes cluster. Without robust networking, components cannot communicate within a Kubernetes cluster and to resources and users outside the cluster. Container networking also serves as crucial functions for maintaining security, reliability, and service quality.

Container networking must be dynamic. When containers are created and destroyed, container security and QoS policies must be re-applied to the newly created container instance. Container networking must align with Kubernetes multi-tenancy and allow the use of troubleshooting tools to ease the adoption of containers.
Container networking must also be efficient, cloud and infrastructure agnostic, and able to leverage the capabilities supported by the underlying infrastructure to provide a low latency and high throughput.

**Cloud Persistence Storage**

Telco workloads consist of both stateful and stateless components. 5G stateful components such as UDM and PCF require data persistence. If a UDM or PCF Pod is stopped and brought back online, the subscriber data associated with the new instance must not be lost. The underlying storage provider supporting data persistence must be open and flexible, and also support access modes required by the Telco applications.

**Kubernetes Substrate**

A standard Kubernetes deployment consists of a control plane and data plane. The Kubernetes control plane consists of kube-apiserver, kube-controller-manager, kube-scheduler, and etc. The Kubernetes data plane consists of one or more worker nodes. Worker node components include kubelet, kube-proxy, and container runtime.

The Kubernetes substrate must be open-source compatible along with an open customization model that allows the use of new Kubernetes tools and resources (such as Knative, Istio, and Operators) without the need for customization. From the Telco Cloud perspective, the Kubernetes substrate can be leveraged through vCloud NFV.

**Cluster Security**

Container security is categorized into three phases: Build, Deploy, and Run. The build and deploy phases involve container image security. Image security involves a tight control of how images are stored and the detection and reporting of known vulnerabilities on the production image. The run phase involves runtime protection of the Kubernetes cluster. Runtime protection involves handling of privileged container escalations, Pod admission control, role-based access control, and federated authentication.

**Container Image Registry**

The container image registry is a repository in which container images are stored. The image registry allows users to pull container images and publish new container images. A private container registry is the best way to distribute and share container images in a secure environment. Use cases for running a private container registry include:

- Tightly control how images are stored
- Distribute and share container images in an isolated environment.
- Image auditing and security vulnerability reporting
Operations and Management Domain

The Operations and Management Domain consists of monitoring and logging. As monolithic VNFs are refactored into microservices and orchestrated with Kubernetes, the requirements for monitoring Telco applications are changing. The application data must be captured at the container level, at scale, and across thousands of endpoints. Because Kubernetes workloads are ephemeral by default and can start or stop at any time, application monitoring must also be dynamic and aware of Kubernetes labels and namespaces. A consistent set of rules or alerts must be applied to all new and old pods.

Centralized logging is an essential part of any Carrier-Grade Kubernetes deployment. Configuring and maintaining a real-time high-performance central repository for log collection can ease the day-to-day operations of tracking what went wrong and its impact. The effective centralized logging helps development teams quickly observe application logs to characterize the application performance. Security compliance and auditing often require an organization to maintain digital trails of who did what and when. In most cases, a robust logging solution is the most efficient way to meet these requirements.

Multi Tenancy

Kubernetes allows you to isolate containers by using namespaces in a single cluster or by deploying containers across multiple clusters. From the Telco standpoint, a tenant is a logical entity that has dedicated RBAC, policies (such as resource quotas), dedicated networks with routing policies, and delegated administration. Multi-tenancy ensures the following:

- Resources such as CPU, memory, network, and storage for each tenant are isolated from other tenants. No workload of a tenant inadvertently impacts other tenant’s workloads.
- Each tenant has sufficient resources available when its workloads are scaled up.
- The networking stack of workloads for a tenant is isolated from other tenants. No workload from two tenants exchange routes with each other. Cross-tenant communication must go through an external network.
- Each tenant has an administrator who performs CRUD operations for their tenancy domain. Access to another tenant’s information is strictly prohibited.
- Separate statistics (for operations) and measurements (for billing) are maintained for each tenant.

Essential PKS Architecture

The VMware NFV vCloud container platform is a layer on top of vCloud NFV, leveraging design recommendations from the latest reference architecture.

The following diagram shows different hosts and components of the Essential PKS architecture:
Essential PKS Architecture

Kubernetes Control Plane runs as pods on the Kubernetes master node.

The following diagram shows the components of the Kubernetes master node:

Kube-API server

The Kubernetes API server is the central management entity that receives all API requests for the management of Kubernetes objects and resources. The API server serves as the frontend to the cluster and is the only cluster component that communicates with the etcd key-value store.
To access the API server, use a dedicated NSX-T Data Center load balancer in front of the control plane nodes. The load balancer performs health checks to ensure that the external clients such as kubectl connect to a healthy API server even during the cluster degradation.

**Kube-Controller-Manager**

The Kubernetes controller manager is a daemon that embeds the core control loops shipped with Kubernetes. In robotics and automation applications, a control loop is a non-terminating loop that regulates the state of the system. In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the API server and attempts to move the current state towards the desired state. The controllers that are shipped with Kubernetes are the replication controller, endpoints controller, namespace controller, and service accounts controller.

**Kube-Scheduler**

Kubernetes schedulers know the total resources available in a Kubernetes cluster and the workload allocated on each worker node in the cluster. The API server invokes the scheduler every time there is a need to make modifications to a Kubernetes pod. Based on the operational service requirements, the scheduler assigns the workload on a node that best fits the resources requirements.

**ETCD**

Etcd is a simple, distributed key-value store used to store the Kubernetes cluster configuration, data, API objects, and service discovery details. For security reasons, etcd must be accessible only from the Kubernetes API server. Etcd can run as a service as part of the master node or as a standalone node. For the deployment environments where Kubernetes resources are constantly created and destroyed, consider dedicated standalone VM worker nodes on hosts with SSD-based datastores. Typical clusters or environments with less churn can integrate etcd with the rest of control plane within the master node.

**Essential PKS Data Plane**

Worker nodes are where all the container workloads run. A worker node requires the container runtime kube-proxy and kubelet daemon to participate as a member of the Kubernetes cluster. Worker nodes run as VMs. Depending on the type of Telco workloads, worker nodes may require advanced network features, such as multiple network interfaces, Container Networking Interface (CNI), IPv6, SR-IOV, exclusive CPU core assignment, and NUMA pinning.

**Logical Building Blocks**

The VMware vCloud NFV container platform is a layer on top of vCloud NFV 3.3 OpenStack Edition, leveraging design recommendations from the latest reference architecture.
Kubernetes clusters are deployed on VMs provisioned by the Virtual Infrastructure Manager leveraging the following capabilities:

- Multi-tenancy and quota management
- Networking: NSX-T Data Center is the networking fabric used to provide networking.
- Tenant Virtual Data Center (Tenant VDC): A tenant VDC allows the creation of VDCs for tenants that offer specific SLA levels for each Telco workload.

**Figure 4-7. Logical Building Blocks**

**Multi-Tenancy and Quota Management**

In VMware Integrated OpenStack, the OpenStack project is the unit of multi-tenancy. In the context of the container reference architecture, a Kubernetes tenant maps to an OpenStack project. Within each project, a tenant can have one or more users and a set of virtual resources based on the project quota. Project quotas are the operational limits that configure the number of system resources available per OpenStack project. A Kubernetes cluster is composed of one or more VMs. The quota system that is used to limit the VM consumption can also be used to limit the Kubernetes cluster consumption.

The OpenStack project quota sets limits on the amount of consumption; however, it does not guarantee the resource availability through reservation. Tenant VDC is a resource pool at its core and provides guaranteed resource availability to tenants. A tenant can have one or more tenant VDCs, each mapping to a different resource availability policy. Tenant VDCs are elastic; more resources can be added to a tenant VDC as its capacity grows.

The tenant VDC can map to a single vSphere cluster or stretch across multiple vSphere clusters. As with VM instances, a tenant can provision Kubernetes clusters into any authorized project. The amount of resources guaranteed to a Kubernetes cluster is based on the capacity of the tenant VDC.

**Container Network**

Kubernetes networking is managed with the Container Networking Interface (CNI) plugin. The NSX Container Plugin (NCP) is a CNCF-compliant container plugin that integrates with NSX-T Data Center to build Kubernetes networking and security objects.
NCP provides integration between NSX-T Data Center and Kubernetes. The main component of NCP runs in a container on each worker node and communicates with the NSX Manager and the Kubernetes control plane. NCP also monitors changes to containers and other resources and manages networking resources (such as logical ports, segments, gateways, and security groups) for containers by the NSX API. Additional NCP functionalities are as follows:

- Automatically creates an NSX-T Data Center logical topology for a Kubernetes cluster and creates a separate segment for each Kubernetes namespace.
- Connects Kubernetes pods to a segment and allocates IP addresses.
- Supports network address translation (NAT) and allocates a separate Source NAT (SNAT) IP for each Kubernetes namespace.
- Works with the NSX node agent to connect Kubernetes pods to a segment. The NSX node agent runs on each Kubernetes node as a DaemonSet.

**Container Networking Architecture**

After the Kubernetes cluster is deployed, the NCP plug-in automatically creates an NSX-T Data Center segment and Tier-1 gateway for the configured namespaces. Each Kubernetes cluster is associated with a Tier-0 provider gateway instance. The Tier-0 gateways are provisioned and maintained by the cloud admin. The UUID of the Tier-0 gateway is provided to the tenant admin and programmed into the NCP during the time of Kubernetes cluster creation.

NCP is responsible for attaching the Tier-1 gateway to the Tier-0 provider gateway instance. The traffic between Kubernetes namespaces and the external physical domain is routed across the Tier-0 gateway. A single Tier-0 gateway can support one or more Kubernetes clusters. The following diagram shows how NSX-T Data Center integrates with Essential PKS.
Cloud Persistent Storage

For stateful applications, Kubernetes uses volume plugins to link containers with an external storage to provide storage persistence. Traditionally, the volume plugins were built, compiled, and bundled with the Kubernetes code base, also known as the in-tree model. The in-tree model is restrictive; the addition of new storage platform capabilities requires changes in the Kubernetes codebase. With the introduction of Container Storage Interface (CSI) and flex volume, the volume plugins can be bundled outside of the Kubernetes codebase, also known as the out-tree model. The out-of-tree approach enables faster feature velocity and is recommended by VMware and Kubernetes community. The vCloud NFV container reference architecture leverages the OpenStack Cinder plugin as the persistent storage implementation.

Cluster Security

Cluster security is categorized into Build, Deploy, and Run. Container images must be scanned for known security vulnerabilities during the container build and integrated with your Continuous Integration and Continuous Delivery (CICD) pipeline. For Telco applications, most containers images are distributed by VNF vendors where the scanning for vulnerabilities during the build phase is not applicable. Therefore, it is critical to scan images during the ingest phase. For the maintenance recommendations about container images, see Container Image Management.
The CICD pipeline for Telco Applications is about adopting the best practices of the software development life cycle, so that there is always a single source of truth to maintain versioning, CNF configuration, and operational traceability. Similar to how software codes are maintained, the CNF configuration must be maintained centrally, with RBAC and versioning. Any changes to the deployments can be traced and rolled back as needed. The CICD pipeline design is beyond the scope of the current reference architecture.

Kubernetes uses Role-Based Access Control (RBAC) to ensure that the Kubernetes cluster resources are exposed only to authorized users. User roles and role bindings are used to grant the user and service account access to Kubernetes cluster resources. Cluster admins must avoid the Kubernetes cluster-wide permission assignment in favor of namespace-specific permissions. Cluster admins must leverage Kubernetes service accounts to provide access to CNF applications that require the Kubernetes API access and must limit the authorization to the smallest set of required resources.

The Kubernetes Pod Security Policies (PSP) provide a policy-driven mechanism for requiring applications in the cluster to use containers in an approved way. Use pod security policies to prevent privilege escalation and containers running as root where appropriate.

To simplify the security management, separate the workloads across Kubernetes clusters instead of namespaces. Separating workloads across different Kubernetes clusters simplifies the RBAC policy management and reduces the risk of sensitive applications being accessed through less secure applications that share a container host or runtime.

Ensure that the network policies are always enabled.

**Container Image Management**

Containers are distributed by VNF vendors to the Telco operators as Docker Images. Containerized VNFs are released and updated at a higher frequency than the VM-based VNFs deployed currently in many networks. Therefore, tracking and managing containers can be challenging for Telco operators. For security reasons, Telco operators must have a tight control over the production CNF images. Image control is categorized into image notary, image access, and vulnerability control.

- **Image access:** Assign different levels of access to Kubernetes users based on their job function. Within each access level, users can only perform actions explicitly allowed by the cloud admin. For example, a guest user can only pull images, and a developer can push and pull images to and from registry. As part of the access control, the cloud admin can block certain images from being accessed. The admin can also prevent the usage of unstable or latest versions of the software until the validation completion.

- **Image notary:** Image notary or signing is about publishing and managing trust between image publishers and consumers. The image publisher must digitally sign all images stored in the container registry, and thereby prevent running unsigned images in a production deployment.
Vulnerability control: Docker image security scanning finds security vulnerabilities in the Docker image files. Image scanning parses through packages and dependencies that are defined in a container image, and checks for any known vulnerabilities in those packages or dependencies. Container image vulnerability scanning is required as part of the image ingest and periodically against production images. If the scan results are above a risk threshold, the container registry must mark the container image as non-compliant. The cloud admin must set retrieval policies to prevent a new workload instantiation of non-compliant container images.

Helm is widely leveraged by CNF vendors to simplify container packaging. With Helm charts, dependencies between CNFs are handled in formats agreed upon by the upstream community; allowing Telco operators to consume CNF packages in a declarative and easy to operate manner. With a proper version management, Helm charts also simplify workload updates. VMware bundles Harbor with essential PKS for secure container registry and Helm Chart management. Harbor is an open-source cloud-native registry that solves common container image challenges by delivering trust, compliance, performance, and interoperability. It fills a gap for organizations and applications that cannot use a public or cloud-based registry or want a consistent experience across clouds.

Cluster User Authentication

Kubernetes supports a few methods of authentication ranging from X509 client Certs, static token, password, service accounts, anonymous access, and OpenID Connect. Static token or password files are insecure and revoking user access can be challenging; therefore neither of them must be considered for production. Service accounts are reserved for in-cluster communication by pods and must not be carried out of the cluster. Authentication option that offers the most flexibility and security is based on OpenID Connect (OIDC). OIDC is more secure as the token exchange mechanism has a tiny attack surface. Also, even if a token is compromised, additional information is required to use the compromised token. Kubernetes lets you provide authentication with any public OpenID Connect-compliant provider, for example GitHub and Google.

Custom integration is also possible for user authentication. Most enterprises and service providers rely on Active Directory or LDAP as the central repository to manage users and groups. While Kubernetes does not support direct Active directory integration, several open-source and commercial products can integrate the active directory as the backend connector to OIDC.

Cloud Native Deployment Design

The Essential PKS design leverages the IaaS design recommendations of the vCloud NFV reference architecture. Two-Pod and Three-Pod deployment models are fully supported.
Figure 4-9. Cloud Native Deployment Design
Note To avoid the conflict of terminology, Pod is hereafter called as domain.

Management Domain

Architecture or hardware changes are not required in the vCloud NFV Management domain to support containers. The management domain comprises the Virtual Infrastructure Management (VIM), vCenter Server, and NSX Manager. The management domain in the core data center also contains the operation management components required for monitoring.

Kubernetes users can use only the approved version of the Essential PKS release. An Essential PKS repository server must be added to the management domain. The Essential PKS repository server mirrors the VMware signed Kubernetes packages; outbound Internet access is required on this server.

After the repository server is online, use aptly or yum to set up the mirror for VMware Essential PKS binaries and NCP dependencies.

Note The VM that you use to serve the Essential PKS repository must run the same distro and version as your cluster nodes.

To enforce the use of approved Kubernetes versions for new Kubernetes clusters, limit the Kubernetes cluster creation from the Infra node only. The infra node is a critical component in the service plane. For infra node details, see Resource Domain.

VMware Essential PKS signed binaries also include Docker container images. To host the Essential PKS signed binaries, the Management Workload domain must have a private container image registry. In addition to the Essential PKS signed binaries, this registry also hosts CNF images or helm charts that are required to deploy Telco applications. The registry must be multi-tenant aware and must support both image notary and vulnerability scanning.

Edge Domain

The Edge Pod provides North-South connectivity to the external provider networks. Architecture and hardware requirements remain the same in the edge pod to support containers. Network services, such as NAT, DHCP, and network metadata services, are always instantiated in the Edge pod. The services (such as service load balancer and L7 ingress controller) required for the container run time are also part of the Edge pod. Multiple configurations can be used for performance, scale, and Edge services.

Resource Domain

Architecture and hardware requirements remain the same in the resource pod to support containers. The vCloud NFV container platform consumes resources from the vCloud NFV Resource Workload Domain. Each tenant can have the following:

- A dedicated container service plane
- An OpenStack project that leverages the underlying SDDC resources to create Essential PKS clusters.
- Kubernetes clusters from a dedicated Tenant VDC to provide isolation.
Cloud Native Logical Architecture

Cloud Native logical architecture contains both service plane and data plane to support containers. The service plane hosts resources required for a tenant to bootstrap and operate Essential PKS clusters. The data plane consists of Kubernetes master and worker VMs.

Service and Workload Planes

The Service and Workload Planes are the core components of the Essential PKS architecture. The service plane provides monitoring and logging capabilities, while the workload plane hosts Kubernetes master and worker nodes.

Figure 4-10. Service and Workload Plane
Service Plane

To simplify the overall tenant consumption of the vCloud NFV container platform, the service plane is a new administrative domain for the container reference architecture. The design objectives of the service plane are:

- Segregate the control plane operations from the workload plane
- Provide a tenant-level isolation for Kubernetes operations
- Provide tenants day-2 visibility into the Kubernetes cloud infrastructure

The service plane hosts resources required to bootstrap, operate, and manage the lifecycle of Essential PKS clusters.

Core components of the service plane include a tenant-specific infra node. The design objectives of the infra node are:

- Enforce repeatability and consistency of the Kubernetes deployment by bundling pre-approved Kubernetes cluster blueprints
- Provide essential tooling for the tenant admin to interact with and manage the Kubernetes cluster
- Provide an interface for policy enforcement and auditing

On the infra node, the cloud admin can pre-stage the Essential PKS Heat templates, build tools, and Kubernetes CLI (kubectl), and provide access to VMware Essential PKS signed local package mirror repository.

The service plane is also an ideal place to host the tenant-specific log aggregator and monitoring.

The service plane deployment is designed to align with the vCloud NFV tenancy model. It is deployed as a separate OpenStack project, one per tenant.

Figure 4-11. Service Plane per Tenant

A tenant can have multiple Kubernetes clusters. A single instance of the service plane deployment can support multiple Kubernetes clusters.
If a Telco application spans across multiple OpenStack projects, one instance of the service plane can service requirements for all projects belonging to the tenant.

To ensure the scalability and overall design simplicity of the container architecture, the service plane OpenStack project is controlled and managed by the cloud admin. The cloud admin deploys and maintains all management applications residing in the service plane. The tenant admin has the API or SSH access to the services exposed by the service plane components but does not have the OpenStack project access.
Figure 4-14. Container Service Plane Persona

<table>
<thead>
<tr>
<th>Cloud Admin</th>
<th>Tenant 1</th>
<th>Tenant 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Create Service Plane Projects</td>
<td>• Access to Infra note to create K8s cluster</td>
<td>• SSH access to Infra node to create K8s cluster</td>
</tr>
<tr>
<td>• Create Service Plane Objects (Compute/Network/Storage and Quota)</td>
<td>• Integrages K8s Cluster logging and monitoring</td>
<td>• Integrages K8s Cluster logging and monitoring</td>
</tr>
<tr>
<td>• Deploy and Manage Monitoring and Logging tools</td>
<td>• Access K8s logging and monitoring</td>
<td>• Access K8s logging and monitoring</td>
</tr>
<tr>
<td>• Deploy and Manage Infra nodes</td>
<td>• Delegate monitoring and logging access to Tenant Users</td>
<td>• Delegate monitoring and logging access to Tenant Users</td>
</tr>
<tr>
<td>• Deploy and Manage other infrastructure tools</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tenant Admin

Tenant User

Tenant User

Tenant User

**Workload Plane Tenant Cluster**

The workload plane of the vCloud NFV container platform consists of Kubernetes master and worker nodes.

Master node: The Kubernetes control plane must run in redundant mode to avoid a single point of failure. To improve the API availability, an NSX-T Data Center load balancer is placed in front of the master nodes. The load balancer must perform health checks to ensure the API server availability. The following table lists the HA characteristics of the master node components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Server</td>
<td>Active/Active</td>
</tr>
<tr>
<td>Kube-controller-manager</td>
<td>Active/Passive</td>
</tr>
<tr>
<td>Kube-scheduler</td>
<td>Active/Passive</td>
</tr>
</tbody>
</table>

**Note** Do not place workloads on the control plane.
You must run etcd in cluster mode. etcd requires an odd number of cluster members to establish a quorum. A 3-node cluster tolerates the loss of a single member, while a 5-node cluster tolerates the loss of 2 members. The number of etcd instances is decided based on the high availability requirements. In a stacked mode deployment, etcd availability determines the number of master nodes.

Worker Node: It is crucial to place CNFs on Kubernetes worker nodes based on performance criteria. The ETSI NFV Performance & Portability Best Practices (GS NFV-PER 001) classifies NFV workloads into different classes. The characteristics distinguishing the workload classes are as follows:

<table>
<thead>
<tr>
<th>Workload Classes</th>
<th>Workload Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data plane workloads</td>
<td>Data plane workloads cover all tasks related to packet handling in an end-to-end communication between edge applications. These tasks are intensive in I/O operations and memory R/W operations.</td>
</tr>
<tr>
<td>Control plane workloads</td>
<td>Control plane workloads cover any other communication between Network Functions that is not directly related to the end-to-end data communication between edge applications. This category of communication includes session management, routing, and authentication. Compared to data plane workloads, control plane workloads are less intensive in terms of transactions per second, while the complexity of the transactions might be higher.</td>
</tr>
<tr>
<td>Signal processing workloads</td>
<td>Signal processing workloads cover all tasks related to digital processing, such as the FFT decoding and encoding in a cellular base station. These tasks are intensive in CPU processing capacity and are highly latency sensitive.</td>
</tr>
<tr>
<td>Storage workloads</td>
<td>Storage workloads cover all tasks related to disk storage.</td>
</tr>
</tbody>
</table>

For 5G services, data plane workloads are further categorized into the following profiles:

<table>
<thead>
<tr>
<th>Profile</th>
<th>Workload Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile 1</td>
<td>Low data rate with a large number of endpoints, best effort bandwidth, jitter, or latency. Massive Machine-type Communication (MMC) is an example application for profile 1.</td>
</tr>
<tr>
<td>Profile 2</td>
<td>High data rate with bandwidth guarantee only, no jitter or latency. Example application include enhancement Mobile Broadband (eMBB).</td>
</tr>
<tr>
<td>Profile 3</td>
<td>High data rate with bandwidth, latency, and jitter guarantee. Example application include factory automation, virtual and augmented reality.</td>
</tr>
</tbody>
</table>

**Workload Profile**

Each type of workload places different requirements on the Kubernetes cluster or worker node.

For example, worker nodes for profile 3 may require Kernel module updates such as huge pages, SCTP and SRIOV, or a DPDK-capable NIC with complete vertical NUMA alignment. Profiles 2 and 3 may not require specialized hardware but have requirements for kernel module updates. When you size Kubernetes worker nodes to meet the 5G SLA requirements, size the worker node based on workload characteristics. To ensure the resource availability, limit the number of pods per Kubernetes node based on the hardware configuration.
When you design a cluster for high availability, consider the following:

- Number of failed nodes to be tolerated at once
- Number of nodes available after a failure
- Remaining capacity to reschedule pods of failed node
- Remaining Pod density after rescheduling

Mapping Workloads to Kubernetes Deployments

Kubernetes deployments must be built based on the Service-Level Objectives of the Telco workload. For the maximum isolation and ease of performance management, it is recommended that Kubernetes tenancy is at the cluster level. Run multiple Kubernetes clusters, and distribute the Kubernetes clusters across multiple vSphere hosts or vSphere clusters for maximum availability.

If a single Kubernetes cluster must be used to host workloads with different operating characteristics, use Kubernetes namespaces to segregate the workloads within the Kubernetes cluster. If different hardware or worker node configurations are needed to support a workload, use tolerations and taints to ensure workload placement. Taints and tolerations work together to ensure that pods do not schedule onto inappropriate nodes. When taints are applied to a set of worker nodes, sharing a common hardware profile, the scheduler will not assign any pods that cannot tolerate the taint. For implementation details, see the Kubernetes documentation.

When using Kubernetes namespaces to separate the workloads, tenant admins must apply the correct RBAC policy at the namespace level so that only authorized users can access resources within a namespace.

Cloud Native Cluster Networking

Telco container architecture lets you provision, manage, and secure Kubernetes cluster deployments on isolated tenant networks. As shown in the following diagram, the data traffic between the tenants of each service provider is isolated. In addition, the tenants can bring in their own IP address ranges to assign
them to Kubernetes pods, services, and SNATs. Each tenant has a dedicated peering point with the physical domain.

The Shared Tier-0 gateway handles traffic between the PKS management network and the vSphere standard network where vCenter, NSX Manager, and VMware Integrated OpenStack are deployed. The tenant Tier-0 gateway connects to the shared Tier-0 over an NSX-T Data Center segment using a VLAN or an overlay transport zone.

**Figure 4-16. Cloud Native Cluster Networking**

![Diagram of Cloud Native Cluster Networking](image)

**Kubernetes Management Network**

The Kubernetes management network can support monitoring, logging, and image registry. The following diagram shows the per Cluster view of the container network topology. Each Kubernetes cluster comes with multiple networks. The Kubernetes management network is used for Kubernetes cluster node VMs. The API network allows special pods such as CoreDNS to register with the Kubernetes API server. The Pod network is managed by NCP and used for Pod to Pod communication.
**Figure 4-17. Kubernetes Management Network**

The NSX-T Data Center Tier-0 gateway connects to upstream physical layer 3 device, such as top of rack switches or a pair of redundant routers. Interfaces that connect upstream are Layer-2 adjacent with the physical devices. Interfaces can use BGP or static routing. For now, only the floating address pool must be advertised to the physical devices. Kubernetes Pod and Cluster networks are located behind NAT. Tier-0 gateways operate in Active-Active mode for the maximum north-south throughput. Assignment of a T0 gateway to a tenant occurs both at the OpenStack project and also as part of the NCP configuration.

**Tier-0 Topology**

The NSX-T Data Center Tier-0 gateway connects to upstream physical layer 3 device, such as top of rack switches or a pair of redundant routers. Interfaces that connect upstream are Layer-2 adjacent with the physical devices. Interfaces can use BGP or static routing. For now, only the floating address pool must be advertised to the physical devices. Kubernetes Pod and Cluster networks are located behind NAT. Tier-0 gateways operate in Active-Active mode for the maximum north-south throughput. Assignment of a T0 gateway to a tenant occurs both at the OpenStack project and also as part of the NCP configuration.
The Border Gateway Protocol (BGP) is used for route redistribution and filtering across all Tier-0 gateways. BGP allows the Shared Tier-0 gateway to dynamically discover the location of Kubernetes clusters deployed on each Tenant Tier-0 gateway.

**Figure 4-19. Shared Tier-0 topology**

**Tier-1 Gateway per Kubernetes Namespace**

Each namespace provisioned in Kubernetes maps to a dedicated Tier-1 gateway. Network services such as NAT are enabled on the Tier-1 gateway where appropriate. Tier-1 gateway operates in Active-Standby mode.

**Figure 4-20. Kubernetes Namespace**
Kubernetes Cluster Architecture

kubeadm is a Kubernetes bootstrapping tool that builds Kubernetes clusters with production-ready defaults and fully compliant with Kubernetes Conformance Program.

As stated in the VMware Essential PKS support matrix, kubeadm is the only bootstrapping tool supported through VMware Essential PKS.

The Telco cloud tenants have the option to integrate kubeadm with OpenStack Heat or upstream tools such as Kubespray. A reference implementation of the Heat stack orchestration engine can be found in the VMware github page.

In addition to API access to the VIM controller, the management pod is responsible for hosting image binaries that are required to deploy a production-grade Kubernetes cluster. During the cluster deployment, a tenant admin can specify the number of control plane and worker nodes to implement. The orchestration engine provisions the required VMs in the resource Pod, based on nova-scheduler affinity filters. After the VM is provisioned, the number of worker nodes in a Kubernetes cluster can scale in or scale out based on utilization.

For path isolation, each tenant can map to a dedicated Tier-0 gateway. A minimum of two NSX Edge nodes is required per tenant in a redundant design. The NSX Edge nodes reside in the Edge Pod and can be deployed as either VMs or bare metal.

External Service Access

By default Kubernetes services and pods have IPs routable only within the cluster network. To expose a workload outside of the cluster, services such as Load Balancer or Ingress can be used.

- Load Balancer: Deploying a service of type Load Balancer triggers NCP to reserve a Virtual IP (VIP) from the external IP pool and associates pods based on Kubernetes labels. You can use the VIP address to forward all traffic to your service. Since Load Balancers operates at Layer 4, both HTTP and HTTPS services can be exposed.
Ingress: Ingress is an API object that describes a collection of rules to allow external access to cluster services. Ingress is a critical component in Kubernetes as it is the aggregation point for all user traffic entering and leaving the Kubernetes cluster.
An ingress controller can be configured to provide externally reachable URLs, load balance the traffic, terminate SSL, and offer name-based virtual hosting. vCloud NFV recommends the following ingress controllers:

- **NCP**: NCP watches for ingress events in the Kubernetes API request. It automatically provisions layer-7 virtual servers for Kubernetes ingress, one for HTTP and another for HTTPS. The NSX-T Data Center load balancer is integrated with NCP and does not require additional configuration. For more information, see Supported Load Balancer Features in the NSX-T Data Center documentation.

- **Contour**: Contour is a VMware open-source Ingress controller that leverages Envoy as the data plane. Contour is optimal for large shared Kubernetes clusters that have a high ingress rule change frequency. It uses the concept of delegation to ensure coordination across rule changes. Only authorized changes reflect in the forwarding state of the controller. For every ingress rule, the tenant admin gives authority for a path or domain to Kubernetes users assigned to a particular namespace. Changes from an authorized namespace are accepted. Changes from unauthorized namespaces are marked and not programmed into the data plane. In addition to rule delegation, Contour also ensures that the dynamic updates to Ingress configuration do not impact the established TCP sessions across the Envoy proxy.

### Network Policy

Kubernetes network policy is a namespace property through which firewall rules can be defined. When using network policy, you assign rules to allow traffic between pods or into and out of a namespace. Use policyType ingress to protect traffic into a namespace and policyType Egress to control traffic out of a namespace. When defining network policies, exclude management components (such as the NSX Manager, API server, general monitoring tools) from the distributed firewall policy to avoid lockout or false alerts.

### Affinity Setting

Every Kubernetes cluster has a redundant set of master and worker nodes for added availability. To ensure that a single failure does not take down the entire Kubernetes cluster:

- Use VMware DRS Anti-Affinity or OpenStack hostgroups to ensure that the Kubernetes control VMs are not running on the same host.

- In a vSphere data center with multiple compute clusters, designate each compute cluster as a separate failure domain (Availability Zone). Use the Availability Zone information to deploy master and worker nodes evenly across Zones for the maximum availability.
At the Kubernetes layer, assign each worker node to a failure-domain using Kubernetes labels. In addition to failure-domain, use PodAntiAffinity and PodAffinity to control the CNF placement and ensure the maximum performance and availability.

**Workload Performance Considerations**

Telco workloads are classified based on their performance. The control plane workload performance must be supported adequately using the standard configurations on Kubernetes. The data plane workload performance can benefit from further tuning, as detailed in the following sections:

- NUMA Topology
- CPU Core Affinity
- Huge Pages
- Multiple Network Interface Design
IPv4 and IPv6 Requirements

NUMA Topology

In Non-Uniform Memory Access (NUMA), access to the memory zone local to the CPU is much faster than the memory associated with a remote CPU. Higher packet throughput can be sustained when sending and receiving data across vNICs within the same NUMA zone than across different NUMA zones.

When deploying Kubernetes worker nodes hosting a high data bandwidth application, ensure that the processor, memory, and vNIC are vertically aligned and remain within a single NUMA boundary. In addition to the worker node alignment, Topology Manager is also introduced in Kubernetes 1.16. The topology manager is a new component in the Kubelet. At the pod admission time, the topology manager figures out the best locality of resources by pulling topology hints from the Device Manager and the CPU manager. With the topology manager, the scheduler places pods only on worker nodes that meet the resource alignment.

**Figure 4-24. NUMA Alignment**

CPU Core Affinity

In addition to the vertical NUMA node alignment to provide the low-latency local memory access, it is also critical to minimize the CPU context switching and process the scheduling delay. Under normal circumstances, the kernel scheduler treats all CPU cycles as available for scheduling and preempts the executing processes to give the CPU time to other applications. Preempting one workload and assigning the CPU time to another workload is known as context switching. Context switching is necessary for CPU sharing during an idle workload. However, at the time of peak data throughput, context switching can lead to performance degradation. To optimize the CNF performance, dedicate and affinitize CPU cores to the worker node that hosts the data plane intensive workload. The CNF workload is then affinitized to the CPU core assigned to the worker node, providing exclusive access to the CPU resource. CPU pinning reduces context switching and maximizes the CPU cache utilization leading to deterministic behavior of CNF workloads. It is recommended to leverage the CPU Core affinity only for worker nodes that are meant to host data plane intensive workloads.
CPU pinning for containers is still in the early stages for Kubernetes. The CPU manager was introduced as an alpha feature in Kubernetes v1.8 and is currently in beta status. The CPU manager comes with the following two policies:

- **None**: Default policy. The kubelet uses CFS quota to enforce pod CPU limits. The workload can move between different CPU cores depending on the load on the Pod and the available capacity on the worker node.

- **Static**: With the static policy enabled, the CPU request results in the container getting allocated the whole CPU. It is also guaranteed that no other container can schedule on that CPU.

For data plane intensive workloads, the CPU manager policy must be set to static to guarantee an exclusive CPU core on the worker node.

### Huge Pages

Translation Lookaside Buffer (TLB) is a small hardware cache used to map virtual pages to physical hardware memory pages. For workloads that are sensitive to memory access latency or require a large amount of memory, the TLB miss must be minimized as much as possible. To reduce the chance of a TLB miss, enable huge pages on the Kubernetes worker node.

For Telco workloads, the default huge page size must be 1 Gbyte. For the worker node to report its huge page capacity, huge pages must be pre-allocated. Pre-allocated huge pages reduce the amount of available memory on a worker node. A node can only pre-allocate huge pages for the default size. The Transport Huge Pages must be disabled.

Container workloads requiring huge pages will use resource hugepages-<hugepagesize> in the Pod specification. Only a single hugepage per pod specification is supported. Hugepages allocation occurs at a pod level.

**Note** If a Pod has multiple containers each requesting hugepages, Kubernetes cannot prevent one container from consuming hugepages reserved for another container.

### Multiple Network Interface Design

CNF workloads such as the SMF or UPF may require multiple network interfaces to support tenant isolation or service routing. Multus is an industry terminology that refers to the container network interface (CNI) plugin for Kubernetes. It enables attaching multiple network interfaces to pods.

With the NCP plugin, NCP creates a default network for every pod. Tenant admins can define a pool of available interfaces through Kubernetes for selective workloads. Kubernetes users can request additional interfaces through pod specification using annotations. Additional interfaces can be both SR-IOV or ENS interfaces. IPAM instance assigned to the secondary interface is independent of the default network. The initial NCP implementation leverages the static IPAM model, and a Telco operator can bind either IPv4 or IPv6 static IPAM to a Multus interface. Dynamic IPAM is in the future consideration.

**Note** This feature is TechnologyPreview only.
IPv4 and IPv6 Support

Kubernetes supports IPv6-only clusters with a single IP stack for Telco workloads. Although the NSX NCP plugin is dual-stack capable, it must operate within the Kubernetes 'single-Pod-IP' limitations:

- Some CNI network plugins are capable of assigning dual-stack addresses on a pod, but Kubernetes is aware of only one address per pod.
- Kubernetes system pods (api server, controller manager, and so on.) can have only one IP address per pod. The system pod addresses are either all IPv4 or all IPv6.
- Endpoints for services are either all IPv4 or all IPv6 within a cluster.
- Service IPs are either all IPv4 or all IPv6 within a cluster.

Without the dual stack Pod support, applications that require a legacy IPv4-only client or services requires NAT64 and NAT46 to be implemented at the IPv6 and IPv4 boundary. This boundary can reside in the physical domain, or within NSX-T Data Center.

Additional interfaces configured by NCP can be assigned to a different address family. The following figure shows an example of SR-IOV interface in an IPv6 only cluster configured with IPv4.

**Note**

- Dual stack Pod support is added recently in Kubernetes 1.16.
- IPv6 feature is Technology Preview only.
Kubernetes Security Image and Monitoring

Cloud native provides image management using the inbuilt application. For monitoring, it uses logging tools to report the application-level faults or issues. Security model uses an integrated Identity provider that supports the OIDC protocol.

Kubernetes Security

The authentication architecture based on OIDC consists of a user backend that is integrated with an identity provider supporting OIDC protocol. After successful OIDC authentication, a user is issued with tokens that must be used in every client API request to the API server.
Figure 4-27. Security Architecture

Authentication

The key components of the authentication architecture are:

- **User database**: The user database must support the ability to add, modify, and remove users. This database is controlled by the cloud admin. The database can be standalone or a part of the identity provider. If the database is in standalone mode, the identity provider and the user database must align on the supported schema and protocol.

- **Identity provider**: Identity providers can be public providers such as GitHub or locally managed providers. If your organization has multiple identity providers or backends, an identity broker can be leveraged. An identity broker redirects authentication to the specific backends or other established identity providers. Opensource dex supports this type of integration.

- **Client Auth**: A Kubernetes user authenticates with the identity provider to retrieve an id_token for the Kubernetes API access. Tools are available to assist a user with the OIDC authentication process and configuration of the kubectl client. Project Gangway is one such tool recommended by VMware. Gangway is an Opensource project and can run as a client of an upstream Identity Service that speaks OIDC. The Kubernetes users point their browser to Gangway and complete the authenticating flow with the upstream Identity Service using Gangway. For the implementation details, see the Gangway GitHub page.

Authorization

When a cluster is deployed using kubeadm, RBAC is enabled by default. Kubernetes authorizes API requests using the API server. It evaluates all the request attributes against all policies and allows or denies the request. The default policy is to deny all. To allow the API request, it must be explicitly allowed by the policy to proceed.
To provide access to authenticated users, tenant admins can grant the user or group access to a specific cluster or namespace using RBAC role or cluster role binding. To get the most benefit from RBAC, VMware CRE team recommends the following:

- Run each component with the most restrictive permissions that still allow for expected functionality. Most applications in a cluster need restricted or no access to the Kubernetes API. System components such as an ingress controller or monitoring system may need more access but can often be limited to read-only access or access within a particular namespace.

- Ensure that the trusted components do not act as pivots that allow less privileged users to escalate privileges. The Kubernetes Dashboard and Helm tiller daemon are examples that deserve special attention. Isolate these components with application-level authentication/authorization or network access controls to prevent unauthorized access.

**Admission Controller**

Admission controllers provide security, governance, and configuration management. The admission control process in two phases. In the first phase, mutating admission controllers are run. In the second phase, validating admission controllers are run. Mutating admission controllers may modify the objects they admit and can be used to ensure consistency and governance. For example, the mutation admission controllers ensure that each container object is tagged using labels that reflect a project or customer id. Validating admission controllers prevent misconfiguration and reject a container deployment if a non-approved version of the image is used.

**PodSecurityPolicy (PSP)** is another type of admission controller. It controls the running of privileged containers, usage of host networking, file system, and so on.

**Figure 4-28. Admission Controller**

For details about admission policies, see the [Kubernetes documentation](https://kubernetes.io/docs/concepts/overview/working-with-objects/admission-control/).

**Pod Security**

PodSecurityPolicy is a built-in admission controller that provides Tenant admins a granular control over what kinds of workloads can be run in a cluster.
Pod Security Policy objects define a set of conditions that a pod must run with in order for the pod to be accepted into the system. It allows you to control:

- Usage of the host file system
- Ability to run privileged containers and privilege escalation.
- Usage of volume types
- Few other aspects of security-focused attributes including sysctl, secomp, and host networking.

When PodSecurityPolicy is enabled, users cannot onboard new pods unless authorized policies are in place. RBAC links the authorization policies to users or groups. In a standard deployment, define at least two pod security policies. The default policy must be restrictive and must map all Kubernetes users. Cluster admin and custom application providers that provide mission-critical services at a cluster level can map to a more permissive policy. When applying policies at a group level, a user belonging to multiple groups may map to multiple policies. If numerous policies match for the user, only the first policy will be assigned alphabetically. For additional customization details, see the Kubernetes documentation.

The following table describes the sample Default and Admin profiles:

<table>
<thead>
<tr>
<th>Type of workload</th>
<th>Default</th>
<th>Custom App/Cluster Admin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privileged</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Run as Root</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Type of workload</td>
<td>Default</td>
<td>Custom App/Cluster Admin</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>HostNetwork</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ConfigMap</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Secret</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PVC</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>HostPath</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>downwardAPI</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>emptyDir</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Dynamic Admission Controllers**

Use dynamic admission to inject custom business logic into the Kubernetes admission control pipeline. Both dynamic admission controllers and pod security policy objects are enabled in the API server as inputs to the admission plugin, leveraging admin-defined webhooks. The webhook is an independent service maintained by the cloud admin and can run as a VM, Kubernetes Pod in the service plane, or serverless. For example, the cloud admin can use ValidatingAdmissionWebHook along with ImagePolicyWebhook to reject all attempts to deploy a CNF using a public registry. For a sample webhook implementation, see the Github page.

The API server calls on the webhooks to evaluate, if the request falls within the intended objective or best practice. The admission controller can either deny the pod admission or alter the declaration to meet the intended objectives selectively.

**CNF Image Management**

Harbor solves common challenges by delivering trust, compliance, performance, and interoperability.

Telco applications consist of CNFs in the form of container images or Helm charts from traditional network vendors and homegrown extensions to BSS for provisioning CNFs. Public or cloud-based registries lack critical security compliance features to operate and maintain deployments.
The image registry allows users to pull container images and the cloud admin to publish new container images. Some of the critical security requirements for a cloud-native container registry are role-based access control and integration with the federated identity provider.

Role-Based Access Control: Different categories of images are required to support Telco Applications. Some CNF images are needed by all tenants, while others are tenant-specific. Cloud admins can determine a set of golden CNF images that all tenants can consume. Tenant admin may also need the images not offered by the cloud admin and will upload private CNF images or charts. To support Telco applications, it is a critical requirement to organize CNF images or helm charts into projects and assign different access permission for CNF images or Helm charts under a project.

Image Scanning and Signing

Container image scanning is an important part of maintaining and building container images. Irrespective of whether the image is built from source or from VNF vendors, it is important to discover any known vulnerabilities and mitigate them before cluster deployment.

Image signing establishes the image trust, ensuring that the image you run in your cluster is the image you intended to run. Notary digitally signs images using keys that allow service providers to securely publish and verify content.

Helm Support

Helm became the de facto package manager of Kubernetes, making it easy to deploy a vast array of Telco applications. Helm chart repository is a must-have system to help the build Telco container infrastructure. Helm charts must work seamlessly together with container images. A single registry capable of supporting both container image and Helm chart management is desirable.
Similar to imaging signing to establish trust, Helm has provenance tools to help chart users verify the integrity and origin of the package. Helm can generate signature files using standard tools based on PKS, GNUpG, and so on at the packaging time. As in the case of image signing, the unsigned helm charts must be blocked from the production.

**Registry Integration with Admission Controller**

Cloud admins may want to reject running container images from public docker registries to ensure image integrity. The easiest way to support this is through the ImagePolicyWebhook.

**Image maintenance**

It is critical to ensure that the CNF images are consistent across multiple data centers. It is recommended to define an image replication policy so that only production ready CNF images are replicated between sites. Images that are not required for production must be bounded to a retention policy, so that obsoleted CNF images do not remain in the registry. To avoid one tenant consuming all available storage resources, resource quota per tenant project is also critical.

**Cloud Persistent Storage**

Persistent storage in Kubernetes can be offered through static provisioning through a Tenant administrator or dynamically consumed with StorageClass. Different storage classes can map to Quality of Service levels. Each StorageClass points to a Storage provisioner. The out of tree OpenStack Cinder provisioner on top of VMware vSAN is recommended. The vCloud NFV container reference architecture also supports certified third-party storage solutions listed in the VMware Compatibility Guide.

**Kubernetes Logging Architecture**

Centralized logging and monitoring are essential part of any Kubernetes deployment. Configuring and maintaining a real-time high-performance central repository for log collection can ease the day-to-day operations of tracking what went wrong and its impact. Effective central logging and monitoring also helps when characterizing overall application performance.
Security compliance and auditing often require maintaining digital trails of who did what and when. In most cases, a robust logging and monitoring solution is the most efficient way to meet these requirements.

**Container Logging**

By default, container engines such as Docker capture the standard output or error and leverage the JSON-file driver on each node to write messages to files. Docker maintains a separate log file for each container and stores it in the `/var/log/containers` directory of the Docker host. Containers write application logs to this file as long as the container is running. The log outputs will be lost if a container is evicted or terminated.

**Cluster logging**

The following components are considered for the Kubernetes Control plane logs:

- Kubelet
- API Server
- Controller Manager
- Scheduler
- **Etcd**

  Some of these components run in a container, and some run as the systemd service. The systemd services write to journald, and components running in a container logs to `/var/log`.

**Kubernetes Logging Tools**

Fluentd: Fluentd is an open-source log processor and forwarder that allows you to collect logs from different sources, unify them, and send them to monitoring destinations. Fluentd is Kubernetes-native and integrates seamlessly with Kubernetes deployments. Fluentd can run as a systemd service on a node, or as a daemonset.

EFK Stack: The EFK Stack (Elasticsearch, FluentD, and Kibana) is an open-source tool for logging Kubernetes. It consists of the following components:

- **Elasticsearch**: Log store. Elasticsearch provides a scalable search and analytics engine for stored logs.
- **Kibana**: Visualization layer. With a user interface, you can query and visualize the logs stored in Elasticsearch.
- **FluentD** – See the above description for Fluentd.

EFK stack can be deployed in the Service Plane for Tenant log aggregation.

vRealize Log Insight: vRealize Log Insight is a log collection and analytics appliance that enables administrators to collect, view, manage, and analyze log data. vRealize Log Insight delivers heterogeneous and highly scalable log management with intuitive, actionable dashboards, sophisticated analytics, and broad third-party extensibility. It provides a deep operational visibility and faster troubleshooting across physical, virtual, and cloud environments.

VMware recommends using Fluentd as a log collector and forwarder for both container and cluster nodes. FluentD sends outputs to vRealize Log Insight for the cloud admin and a second output to EFK stack for the tenant access.

**Monitoring**

As monolithic VNFs are refactored into microservices and orchestrated with Kubernetes, the requirements for monitoring those applications are changing. The application data must be captured at the container level, at scale, and across thousands of endpoints. Because Kubernetes workloads are ephemeral by default and can start or stop at any time, application monitoring must also be dynamic and aware of Kubernetes labels and namespaces. A consistent set of rules or alerts must be applied to all new and old pods.

Maintaining a common layer of baseline metrics that applies to all CNF and infrastructure while also incorporating custom metrics is desirable. No new metric should trigger a major replumb of the monitoring stack.
Monitoring Architecture

cAdvisor: When performing various orchestration tasks (such as scheduling, managing container resource requests and limits) cluster components must collect and analyze cluster metrics such as CPU and RAM consumed by various applications and components of the cluster. cAdvisor is a built-in monitoring solution that provides such metrics. This tool is a running daemon that collects, processes, and exports information about containers. cAdvisor keeps information about container’s historical resource usage, network statistics, and so on. cAdvisor is a part of the Kubelet binary. Kubelet pulls the container resource usage metrics directly from cAdvisor and exposes the aggregated pod resource usage statistics using a REST API.

Metrics API: Kubernetes specification for metrics. The specification describes the following types of metrics:

- **Resource metrics**: Resource usage metrics for pods and nodes, for example CPU and Memory.
- **Custom metrics**: Arbitrary metrics for Kubernetes objects, for example, packet-per-second for a service. The metrics presented by this API may be a superset of those present in the resource metrics API.
- **External metrics**: Metrics foreign to Kubernetes, for example Infrastructure metrics.

Metrics Server: Metrics Server is a cluster-wide aggregator of resource metrics. All the metrics are kept in memory and can be consumed using the kubectl top command.

Prometheus: Prometheus is an open source, metrics-based monitoring system. Prometheus consists of multiple components:

- The Prometheus server scrapes and stores time-series data
- Client libraries instrument application code
- The Push Gateway routes metrics from jobs that cannot be scraped
- Exporters support third-party systems or short-lived jobs
- The Alertmanager generates alerts and notifications

In addition, Grafana, which is a popular open platform for analytics, provides the data visualization layer.
Architectural Realization Using
Cloud Native

The business objective of 5G is to enable applications that consume a large amount of data (for example, live streaming), provide an immersive experience (for example, Augmented Reality and Virtual Reality), and enable proliferation of network connectivity to billions of devices for industry control, connected cars, or machine to machine communication in IoT with a low latency and throughput.

**Figure 5-1. Differentiated Services Profile in 5G**

The 5G network must be dynamic and programmable to meet defined business objectives. Network operators need the ability to provision virtual network slices on-demand with QoS to honor SLA, flexible functions to augment capacity using industry-standard APIs, and re-route traffic during congestion predictively and securely.

**Reference Environment for 5G**

To handle the massive amount of data traffic, 5G is designed to separate the user plane from the control plane and distribute the user plane as close to the device as possible. As the user traffic increases, an operator can add more user plane services without changing the control plane capacity. This distributed architecture can be realized by constructing the data center and network infrastructure based on hierarchical layers. The following figure is an example of a hierarchical 5G design.
Applications such as sensors and IoT smart devices can reside in the device edge. The far edge is the aggregation point for the geographically distributed radio sites hosting RAN and IP routing aggregators. The far edge may also host selective mobile edge computing software, access, mobility, and user plane termination functions of the 5G core. The number of applications that can be hosted in the far edge sites is limited by available power and space.

The near edge is the aggregation point for far edge sites. It hosts many of the services as the far edge. The near edge also serves as a peering point for access to Internet or other infrastructure-related cloud services. If the near edge is not feasible due to power, space or geographic realities (for example in a small country) or if sufficient fiber is available at the far edge, the far edge can connect directly to the core data center bypassing the near edge.

The core layer hosts infrastructure components such as VIM, DNS, DHCP, OSS, monitoring, control plane functions of the 5G core, subscriber database, Kubernetes image repository, and so on.

This chapter includes the following topics:

- Realization of 5G Core Using Cloud Native

## Realization of 5G Core Using Cloud Native

The conceptual design principles discussed in this guide are applied in this section to transform the next-generation 5G-ready cloud infrastructure with a highly distributed control-user plane architecture in a hub-spoke topology. The following diagram depicts the architecture of 5G core.
3GPP 5G Service-Based Architecture

The 3GPP defines a Service-Based Architecture (SBA) for the 5G core, where the control and data plane functionalities are delivered using a set of Cloud-native Network Functions (CNFs), each with authorization to access other’s services. Each CNF is self-contained, portable, and scalable independently.

Figure 5-3. 3GPP 5G Service-Based Architecture

Services are exposed through a Service-Based Interface (SBI) using HTTP/2 REST. Brief descriptions of SBA components are as follows:

- **Network Function (NF) Repository Function (NRF):** Network functions and services produced by CNFs are registered centrally in the NRF. NRF maintains a list of available services and assists other network functions in service discovery. After the service is discovered, a CNF can interact directly with the CNF that produces the service. The Unified Data Repository (UDR) stores user subscription data and provides the data repository service to CNFs, such as the UDM, NRF, and NEF. The UDR also has an authorization management mechanism to guarantee the safety of data access and the information in the repository.

- **Unified Data Management (UDM):** UDM is the stateful message store of subscriber information. The UDM can also be stateless when deployed with the UDR.

- **Policy Control Function (PCF):** PCF uses the subscriber information from the UDM and provides policy rules to other control plane functions for enforcement.

- **Network Expose Function (NEF):** NEF securely exposes services and features of the 5G core and also authorizes all access requests originating from outside of the system. Based on the 3GPP specification, external exposure can be categorized as follows:
  - **Monitoring capability:** Monitors specific events for the UE in 5G System and makes the monitoring events information available for external exposure through the NEF.
  - **Provisioning capability:** Allows an external party to provision information that can be used for the UE in 5G System.
- **Policy/Charging capability:** Handles QoS and charging policy for the UE based on the request from an external party.

- **Access and Mobility Management Function (AMF):** AMF performs registration, reachability, connection, and mobility management between the User Equipment (UE), gNodeB (network equipment that receives the 5G wireless communication between the UE and Mobile network), and 5G core. It also serves as termination points for the Radio Access Network (RAN) control-plane interface (N2). The session management messages between the UE and SMF are proxied by the AMF. AMF discovers the best SMF instance by querying the NRF. AMF forwards subsequent session management messages to the SMF over the SBI interface.

- **Authentication Server Function (AUSF):** AUSF accepts new session requests from the AMF and interacts with the UDM for authentication.

- **Network Slice Selection Function (NSSF):** NSSF helps with the selection of suitable network slice instances for users. One of the key concepts in 5G is network slicing that allows the allocation of the required features and resources from the available network functions to different services or to tenants that are using the services.

- **Session Management Function (SMF):** SMF receives session management messages from the AMF. During the session establishment, SMF queries the PCF and the UDM for QoS and charging rules and also programs the UPF over the N4 interface. A single SMF can program multiple UPFs.

- **User Plane Function (UPF):** UPF is a distributed and configurable data plane that connects the UD to the data network (GTP-U – GRPS Tunneling Protocol for User plane). UPF sessions are programmed from the SMF over the N4 interface. Multiple UPFs can be chained together using the N9 interface. UPF connects to gNodeB using N3 interfaces and to external data network using the N6 interface.

- **Application Function (AF):** AF interacts with the 3GPP core network to provide additional network services to support following:
  - Application influence on traffic routing
  - Accessing NEF
  - Interacting with the policy framework for policy control

Based on the deployment, AF can be either within or outside the operator trust domain. If AF is outside of the trust domain, it must talk to the NEF to interact with the relevant Network function.

### Layered Service-Based Architecture

A layered view of the Service-Based Architecture is shown in the following diagram. At a high level, the UE and NG-RAN send access requests to the AMF. AMF authenticates against the AUSF and queries the NRF to select the SMF instance and passes the session management messages to the SMF. SMF receives the session management messages from the AMF, asks the UDM and PCF for subscription information and programs the UPF based on the class of service. The AF running on top of the 5G core can interact with the NEF to influence provisioning and forwarding decisions. The UDR stores user subscription data and provides the data repository service to CNFs, such as the UDM, NRF, and NEF.
Figure 5-4. Layered Service-Based Architecture

vCloud NFV Cloud Native Architecture

The cloud native orchestration at both the core and edge data centers is based on the Essential PKS upstream Kubernetes. The following diagram illustrates how essential components map to the vCloud NFV cloud native architecture.
A centralized container registry and a Git-based configuration management server are added to the management pod to support native cloud workloads. By centralizing the container image and configuration, it allows the Telco operator to have a complete control over the container image ingestion and access policy, image security, and image replication policies.
The vCloud NFV resource pod hosts both the cloud-native container-based and VM-based workloads. Each tenant gets a pool of resources and deploys VNF or CNF in the assigned resource pool. The VIM layer provides tenant-level isolation. vSphere HA and DRS provide enhanced redundancy for both types of workloads.

The tenant-specific service plane simplifies the onboarding experience. Tools required to deploy, run, and maintain the Essential PKS Kubernetes cluster are pre-populated using the Infra node. Tenant-specific Kubernetes monitoring and logging are also centralized in the service plane. The service plane is elastic and can grow along with CNF workloads. As 5G requirements expand beyond the core data center to the edge, the same service plane can be used to manage the remote edge Kubernetes clusters.

The edge cluster is the aggregation point for geographically distributed radio sites and selective mobile edge applications. Due to the physical constraints of the remote locations, it is essential to provide the same level of visibility and control without having to replicate the core data center.

As a design principle, the Kubernetes cluster in the edge data center must be self-contained and must survive extended outages that impact connectivity to the core data center. Kubernetes components at the edge data center must be implemented with the edge limitations in mind; however, they must fully align with upstream. The private branches are costly to maintain, introduce SLA challenges, and cause future adoption barriers. If the bandwidth is lacking at the edge, container image caching can be deployed to reduce WAN dependency. But it is a design trade-off as added caching tier takes away available compute and storage resources for other Telco mobile business applications.

A data center is as secure as its weakest link, security requirements at the edge is as critical as the core. In addition to the core data center, the edge data center must also leverage the same container runtime protection, RBAC, and Kubernetes admission controllers.

**Conceptual Deployment of 5G Components**

The following figure shows an example of a conceptual deployment of 5G control and data plane components onto the vCloud NFV Cloud Native Infrastructure.
In this example, the 5G SBA control plane components (such as AMF, UDM, SMF, NRF) are deployed in the core data center for centralized control and management. User data plane functions (UPF) are distributed to edge sites to provide a low latency and lower cost of front and backhaul communications. Network Expose Functions (NEF) can be deployed at both the core and edge to expose local services of the Network Functions to external entities. The Application Function (AF) can query the exposed services for provisioning and billing in the core data center.

The AF or NEF instances at the edge can be used to expose Radio Access Network (RAN) capabilities, such as signal quality, to assist the MEC platform in finding the most efficient path with the lowest transmission latency.

Core transport between the edge and core data center can be based on DWDM, Metro Ethernet, or MPLS. Operators must ensure that the sufficient bandwidth is available between the edge site and core site to support session management functions between the UPF and the SMF.

**Core Data Center Cloud Native Design**

Cloud-native workloads using containers are lightweight, simple to update, and declarative. When 5G workloads are built and deployed using cloud-native principles, it is portable and infrastructure agnostic, programmable, and dramatically improves the deployment frequency and time to market.

Design decisions at the core data center around multi-tenancy, cluster sizing, and placement affinity to achieve the maximum SBA control plane availability. For example, the 5G deployment uses one namespace for all SBA control plane CNFs, but it is possible to split SBA control plane CNFs into different namespaces based on CNF roles, for example UDR can reside on a dedicated namespace.
Master nodes at the core data center are deployed in stack mode. A stack mode deployment bundles etcd with all the relevant Kubernetes control plane components. A minimum three master nodes per cluster is required for redundancy.

The following three factors influence the Kubernetes cluster sizing (number of worker nodes) at the core data center:

- Maximum number of CNF per node based on the size of CNF
- Cluster failure tolerance (available capacity after failure to reschedule workloads)
- Capacity for future growth

To ensure that a single failure does not take down the entire Kubernetes cluster or the SBA control plane, master and worker nodes are evenly spread across the ESXi compute resource pool based on the physical data center layout. The same model applies to public cloud-based deployments, where the resource pool maps to an availability zone. At the Kubernetes level, failure-domain labels are also applied to each Kubernetes cluster node. Failure domain labels provide the Kubernetes scheduler with domain awareness. The labels can be assigned by the Kubernetes cloud provider or manually using the Kubernetes CLI.

The network requirements for the core data center are reasonably straightforward. Service-to-service communication between the 5G SBA control plane CNFs primarily uses Service-Based Interfaces (SBI) and do not require multiple NICs or line-rate throughput. Typically, a single vNIC per CNF is required. NSX T1 gateways deployed in Active/Standby mode act as the default gateway for SBA control plane CNFs. Since line-rate performance is not required, advanced placement techniques such as NUMA alignment, CPU pinning, and HugePages tuning are also not needed for the Kubernetes cluster at the core data center. The tenant-specific service plane manages the life cycle of the Kubernetes cluster.

In the deployment scenarios where additional links are required for traffic isolation, NCP can be used to provision multiple container interfaces.

**Realization of Edge using vCloud NFV**

While the core 5G control plane SBA (Service-Based Architecture) is deployed at the core data centers, the UPF is distributed to the Edges. Unlike the core data center, the edge data center is often constrained by compute resources due to limited space and power. Based on those constraints, it is essential that only required components are deployed at the edge data center.

The two main approaches in the Kubernetes community in addressing the edge architecture are:

- Remote edge with full Kubernetes components
- Centralized Kubernetes control plane at core, with custom controllers or Custom Resource Definition (CRD) to deploy and manage the edge.

Both approaches have sets of advantages and disadvantages. Remote edge with full Kubernetes components is simple to operate but potentially introduces overhead in compute and storage. Centralized Kubernetes control plane with custom controllers is lightweight, but still in early stages of implementation and often requires an operator to stitch their application into their framework. VMware is working actively...
with the upstream community to evolve the edge architecture. Based on the current 5G requirements, the remote edge with full Kubernetes is much easier to operationalize and offers a faster time to market. The remote edge data center with full upstream Kubernetes is sized based on edge workloads and is used for the example conceptual deployment of 5G.

Some SBA control plane CNFs and data plane CNFs require multiple network interfaces for traffic isolation. For example, the data plane UPF requires traffic from the UE to be routed to the local data network with the highest throughput, lowest latency, and jitter. The following figure shows network connections in the edge data center supporting the conceptual deployment of 5G.

**Figure 5-7. Realization of Edge using vCloud NFV**

To provide the highest throughput, worker node VM processor, memory, and vNIC need vertical alignment and remain within a single NUMA boundary. The data plane UPF pins to the CPU core assigned to the worker node using the Kubernetes CPU manager and provides an exclusive access to the CPU resources. The Kubernetes Topology manager is used at the admission time to figure out the optimal placement of UPF on worker nodes that best satisfy the resource alignment.
Each UPF has multiple network interfaces. The default network interface is an L3 overlay interface. This interface is used to connect to Kubernetes services (for example Kubelet), and the N4 interface is used to communicate with the SMF. NSX T1 routers deployed in the Active / Standby mode is the default gateway for the overlay interface. Additional interfaces can be SRIOV or Enhanced Data Path. Instead of L3 overlay, SR-IOV or Enhanced Data Path interfaces are deployed in VLAN mode and connect directly to the physical Top of the Rack (ToR) switch for the maximum throughput.

The edge data centers are resource constrained. To minimize the resource usage, metrics and logs are forwarded to the core data center for monitoring and correlation. Fluentd is the log forwarder in our example. All log messages are sent to the tenant-specific service plane log aggregator and a central aggregator maintained by cloud admin by the FluentD agent.

Container images are maintained centrally in the core data center, so there is a single source of truth and access policy definition for all deployment artifacts. The container image download from the core data center may not be extremely bandwidth-intensive, but it may deploy registries at each edge site to further reduce WAN bandwidth. When implementing distributed container registries, ingest container images only at the core data center and set up the image replication policy so that only required container images are replicated to the edge data center.

Similar to the core data center, Kubernetes pod security policy and admission controllers are used to ensure compliance.
Authors and Contributors

This section provides a brief description about the Authors and Contributors.

The following authors co-wrote this paper:

- Arunkumar Kodagi Ramachandra, Solutions Architect, NFV Solutions Engineering, VMware
- Indranil Bal, Solution Consultant, Telco Solutions, VMware
- Mike Brown, Sr. Staff Solution Architect, Telco Solutions, VMware
- Ram Pratap Singh, Senior Member of Technical Staff, Telco Solutions, VMware
- T. Sridhar, Principal Engineer & Chief Architect, Telco Solutions, VMware
- Xiao Gao, Lead Architect, Telco Solutions, VMware

Many thanks for contributions from:

- Andrea Li, Lead Solutions Test Architect, Telco Solutions Engineering, VMware
- Danny Lin, Sr. Director Product Management and Architecture, Telco Solutions, VMware
- Jambi Ganbar, Sr. Technical Solutions Manager, Telco Solutions Engineering, VMware
- Michelle Han, Director, Solutions Testing and Validation, Telco Solutions Engineering, VMware
- Kandasamy M, Graphic Designer, VMware
- Ramkumar Venketaramani, Director Product Management, Telco Solutions, VMware
- Revathi Govindarajan, Senior Technical Writer - Professional, Telco Solutions, VMware

Special thanks for their valuable feedback to: