

NG.139 Cloud Infrastructure Reference Architecture managed by Kubernetes

Version 2.0

xx January 2025

Security Classification: Non-Confidential

Access to and distribution of this document is restricted to the persons permitted by the security classification. This document is subject to copyright protection. This document is to be used only for the purposes for which it has been supplied and information contained in it must not be disclosed or in any other way made available, in whole or in part, to persons other than those permitted under the security classification without the prior written approval of the Association.

Copyright Notice

Copyright © 2025 GSM Association

Disclaimer

The GSMA makes no representation, warranty or undertaking (express or implied) with respect to and does not accept any responsibility for, and hereby disclaims liability for the accuracy or completeness or timeliness of the information contained in this document. The information contained in this document may be subject to change without prior notice.

Compliance Notice

The information contain herein is in full compliance with the GSMA Antitrust Compliance Policy.

This Permanent Reference Document is classified by GSMA as an Industry Specification, as such it has been developed and is maintained by GSMA in accordance with the provisions set out GSMA AA.35 - Procedures for Industry Specifications.

Table of Contents

1 Introduction 5
1.1 Overview 5
1.1.1 Required Component Versions 5
1.1.2 Principles 5
1.1.3 Approach 8
1.2 Scope 8
1.3 Definitions 8
1.4 Abbreviations 13
1.5 References 16
1.6 Conventions 16
1.7 Bibliography 16
2 Architecture Requirements 29
2.1 Introduction for Architecture Requirements 29
2.2 Key Word Definitions 29
2.3 Reference Model Requirements 29
2.3.1 Cloud Infrastructure Software Profile Capabilities 29
2.3.2 Virtual Network Interface Specifications 32
2.3.3 Cloud Infrastructure Software Profile Requirements 33
2.3.4 Cloud Infrastructure Hardware Profile Requirements 35
2.3.5 Edge Cloud Infrastructure Hardware Profile Requirements 36
2.3.6 Cloud Infrastructure Management Requirements 37
2.3.7 Cloud Infrastructure Monitoring Capabilities 38
2.3.8 Cloud Infrastructure Security Requirements 40
2.4 Kubernetes Architecture Requirements 54
3 High Level Architecture 60
3.1 Introduction to High Level Architecture 60
3.2 Infrastructure Services 61
3.2.1 Container Compute Services 61
3.2.2 Container Networking Services 68
3.2.3 Kubernetes Networking Semantics 72
3.2.4 Container Storage Services 75
3.2.5 Kubernetes Application package manager 76
3.2.6 Custom Resources 76
3.3 Platform services 77
3.4 CaaS Manager - Cluster Lifecycle Management 78
4 Component Level Architecture 79
4.1 Introduction to Component Level Architecture 79
4.2 Kubernetes Node 79
4.3 Node Operating System 85
4.4 Kubernetes 86
4.5 Container Runtimes 91
4.6 Networking Solutions 92

4.7 Storage Components 97
4.8 Service Meshes 99
4.9 Kubernetes Application Package Managers 99
4.10 Kubernetes Workloads 99
4.11 Additional Required Components 106
4.12 Platform Service Components 106
4.13 Kubernetes Cluster Lifecycle Management 110
5 Security Guidance 113
5.1 Introduction to Security Guidance 113
5.1.1 Security Perimeters 113
5.2 Security Principles 114
5.3 Node Hardening 114
5.3.1 Node Hardening: Securing the Kubernetes Hosts 114
5.3.2 Restricting Direct Access to Nodes 114
5.3.3 Vulnerability Assessment 114
5.3.4 Patch Management 115
5.4 Securing the Kubernetes Orchestrator 115
5.4.1 Control Network Access to Sensitive Ports 115
5.4.2 Controlling Access to the Kubernetes API 116
5.4.3 Using Transport Layer Security and Service Mesh 116
5.4.4 API Authentication and Authorization 116
5.4.5 Restricting Access to etcd and Encrypting Contents Within etcd 116
5.4.6 Controlling Access to the Kubelet 117
5.4.7 Securing the Kubernetes Dashboard 117
5.5 Using Namespaces to Establish Security Boundaries 117
5.6 Separating Sensitive Workloads 117
5.7 Creating and Defining Network Policies 117
5.8 Running the Latest Version 117
5.9 Securing Platform Metadata 118
5.10 Enabling Logging and Monitoring 118
5.11 Runtime Security 118
5.12 Secrets Management 118
5.13 Trusted Registry 118
5.14 Isolation 119
5.14.1 VM versus Container Isolation 119
5.14.2 Container Isolation in the Kubernetes Cluster 119
6 API and Feature Testing requirements 120
6.1 Introduction to API and Feature Testing requirement 120
6.1.1 Kubernetes feature gate policy 120
6.1.2 Kubernetes API policy 120
6.2 API Machinery Special Interest Group [126] 121
6.3 Apps [127] 122
6.4 Auth Special Interest Group [128] 122
6.5 Cluster Lifecycle Special Interest Group [129] 123

6.6 Instrumentation Special Interest Group [130] 123
6.7 Network Special Interest Group [131] 124
6.8 Node Special Interest Group [132] 125
6.9 Scheduling Special Interest Group [133] 126
6.10 Storage Special Interest Group [134] 127
6.11 Conformance testing 127
6.11.1 Requirements and Testing Principles 127
6.11.2 Test Case traceability 128
6.11.3 Test Cases Traceability to Requirements 136
6.11.4 Kubernetes Testing Cookbook 137
7 Gaps, Innovation, and Development 139
7.1 Introduction to Gaps, Innovation, and Development 139
7.1.1 Gap Template 139
7.1.2 Container Run-Time Interfaces Towards NFVI Resources 139
7.1.3 Multitenancy and Workload Isolation with Kubernetes 139
7.1.4 Kubernetes as a VM-based VNF Orchestrator 139
7.1.5 Native Multiple Network Interfaces on Pods 140
7.1.6 Dynamic Network Management 140
7.1.7 Control Plane Efficiency 140
7.1.8 Interoperability with VRF-based Networking 140
7.1.9 Hardware Topology-Aware Huge Pages 140
7.1.10 User Namespaces in Kubernetes 141
8 Appendix A - Guidance For workload isolation (Multitenancy) with Kubernetes for application vendors 142
8.1 Overview of Appendix A 142
8.2 Solution areas 143
8.3 Multitenancy Models 143
8.3.1 Soft multitenancy with Kubernetes Namespaces per tenant 144
8.3.2 Hard multitenancy with dedicated Kubernetes clusters per tenant 144
9 Document Management 145
A.1 Document History 145
A.2 Other Information 145

1 Introduction

1.1 Overview

The objective of this Reference Architecture (RA) is to develop an usable Kubernetes-based platform for the Telecom industry. The RA is based on the standard Kubernetes platform wherever possible. This Reference Architecture for Kubernetes describes the high-level system components and their interactions, taking the goals and requirements from the Cloud Infrastructure Reference Model [1] (RM) and mapping them to Kubernetes (and related) components. This document needs to be sufficiently detailed and robust such that it can be used to guide the production deployment of Kubernetes within a network operator, whilst being flexible enough to evolve with and remain aligned with the wider Kubernetes ecosystem outside of Telecom.

To set this in context, it makes sense to start with the high-level definition and understanding of Kubernetes. Kubernetes [2] is a "portable, extendable, open source platform for managing containerised workloads and services, that facilitates both declarative configuration and automation. It has a large and rapidly growing ecosystem. Kubernetes services, support, and tools are widely available" [3]. Kubernetes is developed as an open source project in the kubernetes [4] repository of GitHub.

To assist with the goal of creating a reference architecture that will support Telecom workloads, but at the same time leverage the work that already has been completed in the Kubernetes community, RA2 will take an "RA2 Razor" approach to build the foundation. This can be explained along the lines of "if something is useful for non-Telecom workloads, we will not include it only for Telecom workloads". For example, start the Reference Architecture from a vanilla Kubernetes (say, v1.31) feature set, then provide clear evidence that a functional requirement cannot be met by that system (say, multi-NIC support), only then the RA would add the least invasive, Kubernetes-community aligned extension (say, Multus) to fill the gap. If there are still gaps that cannot be filled by standard Kubernetes community technologies or extensions then the RA will concisely record the requirement in the Introduction to Gaps, Innovation, and Development chapter of this document and approach the relevant project maintainers with a request to add this functionality into the feature set.

The Kubernetes Reference Architecture will be used to determine a Kubernetes Reference Implementation. The Kubernetes Reference Implementation would then also be used to test and validate the supportability and compatibility with Kubernetes-based Network Function workloads, and lifecycle management of Kubernetes clusters, of interest to the Anuket community. The intention is to expand as much of the existing test frameworks to be used for the verification and conformance testing of Kubernetes-based workloads, and Kubernetes cluster lifecycle management.

1.1.1 Required Component Versions

Component	Required version(s)
Kubernetes	1.31

1.1.2 Principles

1.1.2.1 Architectural Principles

This Reference Architecture conforms with the Anuket principles:

1. **Open source preference:** for building Cloud Infrastructure solutions, components and tools, using open source technology.

2. **Open APIs:** to enable interoperability, component substitution, and minimise integration efforts.
3. **Separation of concerns:** to promote lifecycle independence of different architectural layers and modules (e.g., disaggregation of software from hardware).
4. **Automated lifecycle management:** to minimise the end-to-end lifecycle costs, maintenance downtime (target zero downtime), and errors resulting from manual processes.
5. **Automated scalability:** of workloads to minimise costs and operational impacts.
6. **Automated closed loop assurance:** for fault resolution, simplification, and cost reduction of cloud operations.
7. **Cloud nativeness:** to optimise the utilisation of resources and enable operational efficiencies.
8. **Security compliance:** to ensure the architecture follows the industry best security practices and is at all levels compliant to relevant security regulations.
9. **Resilience and Availability:** to withstand Single Point of Failure.

1.1.2.2 Cloud Native Principles

For the purposes of this document, the CNCF TOC's (Technical Oversight Committee) definition of Cloud Native applies:

CNCF Cloud Native Definition v1.0 Approved by TOC: 2018-06-11

“Cloud native technologies empower organizations to build and run **scalable** applications in modern, **dynamic environments** such as public, private, and hybrid clouds. Containers, **service meshes**, **microservices**, **immutable infrastructure**, and **declarative APIs** exemplify this approach.

These techniques enable **loosely coupled** systems that are **resilient**, **manageable**, and **observable**. Combined with **robust automation**, they allow engineers to make **high-impact changes frequently and predictably** with minimal toil.

The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.”

The CNCF TUG (Telecom User Group), formed in June 2019, published a set of Cloud Native Principles suited to the requirements of the Telecom community [5]. There are many similarities with the CNCF principles, briefly that infrastructure needs to be:

- **scalable**
- **dynamic environments**
- **service meshes**
- **microservices**
- **immutable infrastructure**
- **declarative APIs**
- **loosely coupled**
- **resilient**
- **manageable**
- **observable**

- **robust automation**
- **high-impact changes frequently and predictably**

1.1.2.3 Exceptions

Anuket specifications define certain policies and general principles and strive to .. add general principles from common coalesce the industry towards conformant Cloud Infrastructure technologies and configurations. With the currently available technology options, incompatibilities, performance, and operator constraints (including costs), these policies and principles may not always be achievable and, thus, require an exception process. These policies describe how to handle non-conforming technologies. .. add policies:anuket project policies for managing non-conforming technologies from common In general, non-conformance with policies is handled through a set of exceptions. .. add gov/chapters/chapter09:exception types

The following sub-sections list the exceptions to the principles of Anuket specifications and shall be updated whenever technology choices, versions and requirements change. The Exceptions have an associated period of validity and this period shall include time for transitioning.

Technology Exceptions

1.1.2.3.1 The list of Technology Exceptions will be updated or removed when alternative technologies, aligned with the principles of Anuket specifications, develop and mature.

Table 1 Technology Exceptions

Ref	Name	Description	Valid Until	Rationale	Implication
ra2.exc.tec.001	SR-IOV	This exception allows workloads to use SR-IOV over PCI-PassThrough technology.	TBD	Emulation of virtual devices for each virtual machine creates an I/O bottleneck resulting in poor performance and limits the number of virtual machines a physical server can support. SR-IOV implements virtual devices in hardware, and by avoiding the use of a switch, near maximal performance can be achieved. For containerisation the downsides of creating dependencies on hardware is reduced as Kubernetes nodes are either physical, or if virtual have no need to "live migrate" as a VNF VM might.	

1.1.3 Approach

The approach taken in this Reference Architecture is to start with a basic Kubernetes architecture, based on the community distribution, and then add detail and additional features/extensions as is required to meet the requirements of the Reference Model and the functional and non-functional requirements of common cloud native network functions.

This document starts with a description of interfaces and capabilities requirements (the "what") before providing guidance on "how" those elements are deployed, through specifications. The details of how the elements will be used together are documented in full detail in the Reference Implementation.

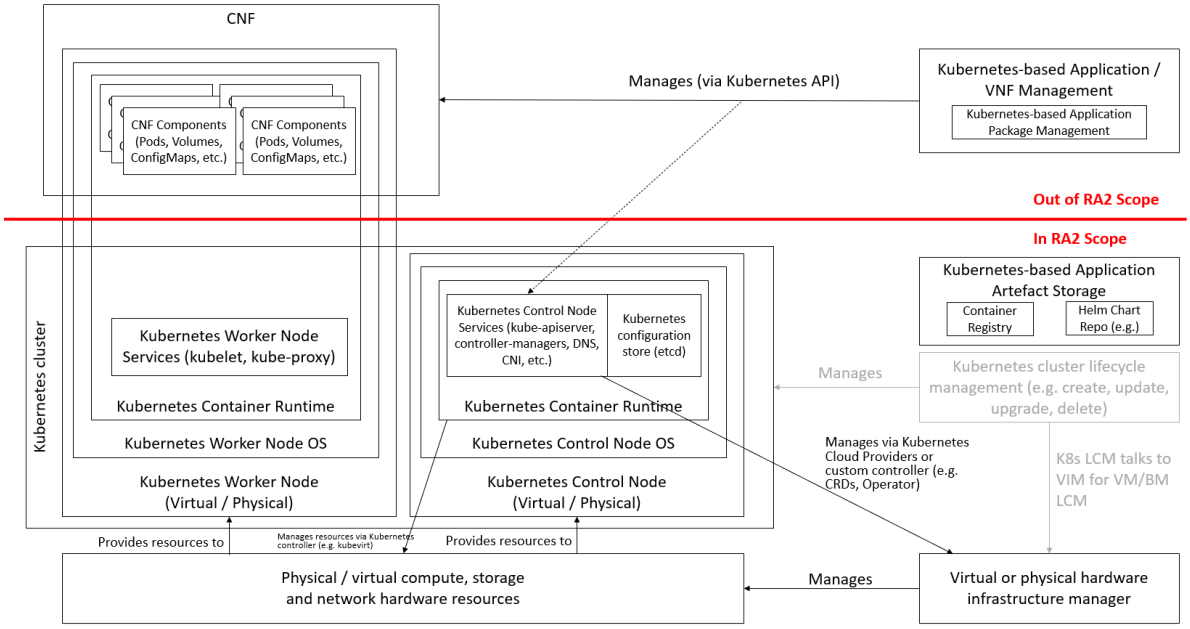
1.2 Scope

The scope of this particular Reference Architecture can be described as follows (the capabilities themselves will be listed and described in subsequent chapters):

- Kubernetes platform capabilities required to conform to the Reference Model requirements
- Support for CNFs that consist wholly of containers
- Support for CNFs that consist partly of containers and partly of VMs, both of which will be orchestrated by Kubernetes
- **Kubernetes Cluster lifecycle management**: including Cluster creation/upgrade/scaling/deletion, and node customisation due to workload requirements.

The following items are considered **out of scope**:

- **Kubernetes-based Application / CNF Management**: this is an application layer capability that is out of scope of Anuket.



Kubernetes Reference Architecture scope

1.3 Definitions

Table 2Definitions

Term	Description
Abstraction	Process of removing concrete, fine-grained or lower-level details or attributes or common properties in the study of systems to focus attention on topics of greater importance or general concepts. It can be the result of decoupling.
Anuket	A Linux Foundation Networking (LFN) open source project developing open reference infrastructure models, architectures, tools, and programs.
CaaS Containers as a Service	A Platform suitable to host and run Containerised workloads, such as Kubernetes. Instances of CaaS Platforms are known as **CaaS Clusters**.
CaaS Manager	A management plane function that manages the lifecycle (instantiation, scaling, healing, etc.) of one or more CaaS instances, including communication with VIM for control plane and node lifecycle management.
Cloud Infrastructure	A generic term covering **NFVI**, **IaaS** and **CaaS** capabilities - essentially the infrastructure on which a **Workload** can be executed. **NFVI**, **IaaS** and **CaaS** layers can be built on top of each other. In case of CaaS some cloud infrastructure features (e.g.: HW management or multitenancy) are implemented by using an underlying **IaaS** layer.
Cloud Infrastructure Hardware Profile	Defines the behaviour, capabilities, configuration, and metrics provided by a cloud infrastructure hardware layer resources available for the workloads.
Cloud Infrastructure Profile	The combination of the Cloud Infrastructure Software Profile and the Cloud Infrastructure Hardware Profile that defines the capabilities and configuration of the Cloud Infrastructure resources available for the workloads.
Cloud Infrastructure Software Profile	Defines the behaviour, capabilities and metrics provided by a Cloud Infrastructure Software Layer on resources available for the workloads.
Cloud Native Network Function (CNF)	A cloud native network function (CNF) is a cloud native application that implements network functionality. A CNF consists of one or more microservices. All layers of a CNF are developed using Cloud Native Principles including immutable infrastructure, declarative APIs, and a “repeatable deployment process”. This definition is derived from the Cloud Native Thinking for Telecommunications Whitepaper, which also includes further detail and examples.
Compute Node	An abstract definition of a server. A compute node can refer to a set of hardware and software that support the VMs or Containers running on it.
Container	A lightweight and portable executable image that contains software and all its dependencies. OCI defines **Container** as "An environment for executing processes with configurable isolation and resource limitations. For example, namespaces, resource limits, and mounts are all part of the container environment." A **Container** provides operating-system-level virtualisation by abstracting the “user space”. One big difference between **Containers** and **VMs** is

	that unlike VMs, where each **VM** is self-contained with all the operating systems components are within the **VM** package, containers "share" the host system's kernel with other containers.
Container Engine	Software components used to create, destroy, and manage containers on top of an operating system.
Container Image	Stored instance of a container that holds a set of software needed to run an application.
Container Runtime	The software that is responsible for running containers. It reads the configuration files for a **Container** from a directory structure, uses that information to create a container, launches a process inside the container, and performs other lifecycle actions.
Core (physical)	An independent computer processing unit that can independently execute CPU instructions and is integrated with other cores on a multiprocessor (chip, integrated circuit die). Please note that the multiprocessor chip is also referred to as a CPU that is placed in a socket of a computer motherboard.
CPU Type	A classification of CPUs by features needed for the execution of computer programs; for example, instruction sets, cache size, number of cores.
Decoupling, Loose Coupling	Loosely coupled system is one in which each of its components has, or makes use of, little or no knowledge of the implementation details of other separate components. Loose coupling is the opposite of tight coupling
Encapsulation	Restriction of direct access to some of an object's components.
External Network	External networks provide network connectivity for a cloud infrastructure tenant to resources outside of the tenant space.
Fluentd	An open source data collector for unified logging layer, which allows data collection and consumption for better use and understanding of data. **Fluentd** is a CNCF graduated project.
Functest	An open source project part of Anuket LFN project. It addresses functional testing with a collection of state-of-the-art virtual infrastructure test suites, including automatic VNF testing.
Hardware resources	Compute, storage and network hardware resources on which the cloud infrastructure platform software, virtual machines and containers run on.
Host Profile	Is another term for a Cloud Infrastructure Hardware Profile.
Huge pages	Physical memory is partitioned and accessed using the basic page unit (in Linux default size of 4 KB). Huge pages, typically 2MB and 1GB size, allows large amounts of memory to be utilised with reduced overhead. In an NFV environment, huge pages are critical to support large memory pool allocation for data packet buffers. This results in fewer Translation Lookaside Buffers (TLB) lookups, which reduces the virtual to physical pages' address translations. Without huge pages enabled high TLB miss rates would occur thereby degrading performance.

Hypervisor	A software that abstracts and isolates workloads with their own operating systems from the underlying physical resources. Also known as a virtual machine monitor (VMM).
Instance	Is a virtual compute resource, in a known state such as running or suspended, that can be used like a physical server. It can be used to specify VM Instance or Container Instance.
Kibana	An open source data visualisation system.
Kubernetes	An open source system for automating deployment, scaling, and management of containerised applications.
Kubernetes Cluster	A set of machines, called nodes (either *workers* or *control plane*), that run containerised applications managed by Kubernetes.
Kubernetes Control Plane	The container orchestration layer that exposes the API and interfaces to define, deploy, and manage the lifecycle of containers.
Kubernetes Node	A node is a worker machine in Kubernetes. A worker node may be a **VM** or physical host, depending on the cluster. It has local daemons or services necessary to run Pods and is managed by the control plane.
Kubernetes Service	An abstract way to expose an application running on a set of Pods as a Kubernetes network service.
Monitoring (Capability)	Monitoring capabilities are used for the passive observation of workload-specific traffic traversing the Cloud Infrastructure. Note, as with all capabilities, Monitoring may be unavailable or intentionally disabled for security reasons in a given cloud infrastructure instance.
Multi-tenancy	Feature where physical, virtual or service resources are allocated in such a way that multiple tenants and their computations and data are isolated from and inaccessible by each other.
Network Function (NF)	Functional block or application that has well-defined external interfaces and well-defined functional behaviour. Within **NFV**, a **Network Function** is implemented in a form of **Virtualised NF** (VNF) or a **Cloud Native NF** (CNF).
NFV Orchestrator (NFVO)	Manages the VNF lifecycle and **Cloud Infrastructure** resources (supported by the **VIM**) to ensure an optimised allocation of the necessary resources and connectivity.
Network Function Virtualisation (NFV)	The concept of separating network functions from the hardware they run on by using a virtual hardware abstraction layer.
Network Function Virtualisation Infrastructure (NFVI)	The totality of all hardware and software components used to build the environment in which a set of virtual applications (VAs) are deployed; also referred to as cloud infrastructure. The NFVI can span across many locations, e.g., places where data centres or edge nodes are operated. The network providing connectivity between these locations is regarded to be part of the cloud infrastructure. **NFVI** and **VNF** are the top-level conceptual entities in the scope of Network Function Virtualisation. All other components are sub-entities of these two main entities.

Network Service (NS)	Composition of **Network Function**(s) and/or **Network Service**(s), defined by its functional and behavioural specification, including the service lifecycle.
Open Network Automation Platform (ONAP)	An LFN project developing a comprehensive platform for orchestration, management, and automation of network and edge computing services for network operators, cloud providers, and enterprises.
ONAP OpenLab	ONAP community lab.
Open Platform for NFV (OPNFV)	A collaborative project under the Linux Foundation. OPNFV is now part of the LFN Anuket project. It aims to implement, test, and deploy tools for conformance and performance of NFV infrastructure.
OPNFV Verification Program (OVP) / Anuket Assured	An open source, community-led compliance and verification program aiming to demonstrate the readiness and availability of commercial NFV products and services using OPNFV and ONAP components.
Platform	A cloud capabilities type in which the cloud service user can deploy, manage and run customer-created or customer-acquired applications using one or more programming languages and one or more execution environments supported by the cloud service provider. Adapted from ITU-T Y.3500. This includes the physical infrastructure, Operating Systems, virtualisation/containerisation software and other orchestration, security, monitoring/logging and life-cycle management software.
Pod	The smallest and simplest Kubernetes object. A Pod represents a set of running containers on a cluster. A Pod is typically set up to run a single primary container. It can also run optional sidecar containers that add supplementary features like logging.
Prometheus	An open source monitoring and alerting system.
Quota	An imposed upper limit on specific types of resources, usually used to prevent excessive resource consumption by a given consumer (tenant, VM, container).
Resource pool	A logical grouping of cloud infrastructure hardware and software resources. A resource pool can be based on a certain resource type (for example, compute, storage and network) or a combination of resource types. A **Cloud Infrastructure** resource can be part of none, one or more resource pools.
Simultaneous Multithreading (SMT)	Simultaneous multithreading (SMT) is a technique for improving the overall efficiency of superscalar CPUs with hardware multithreading. SMT permits multiple independent threads of execution on a single core to better utilise the resources provided by modern processor architectures.
Tenant	Cloud service users sharing access to a set of physical and virtual resources, ITU-T Y.3500. Tenants represent an independently manageable logical pool of compute, storage and network resources abstracted from physical hardware.
Tenant Instance	Refers to an Instance owned by or dedicated for use by a single **Tenant**.

Tenant (Internal) Networks	Virtual networks that are internal to **Tenant Instances**.
User	Natural person, or entity acting on their behalf, associated with a cloud service customer that uses cloud services. Examples of such entities include devices and applications.
Virtual CPU (vCPU)	Represents a portion of the host's computing resources allocated to a virtualised resource, for example, to a virtual machine or a container. One or more vCPUs can be assigned to a virtualised resource.
Virtualised Infrastructure Manager (VIM)	Responsible for controlling and managing the Network Function Virtualisation Infrastructure (NFVI) compute, storage and network resources.
Virtual Machine (VM)	Virtualised computation environment that behaves like a physical computer/server. A **VM** consists of all of the components (processor (CPU), memory, storage, interfaces/ports, etc.) of a physical computer/server. It is created using sizing information or Compute Flavour.
Virtualised Network Function (VNF)	A software implementation of a Network Function, capable of running on the Cloud Infrastructure. **VNFs** are built from one or more VNF Components (VNFC) and, in most cases, the VNFC is hosted on a single VM or Container.
Workload	An application (for example **VNF**, or **CNF**) that performs certain task(s) for the users. In the Cloud Infrastructure, these applications run on top of compute resources such as **VMs** or **Containers**.

1.4 Abbreviations

Term	Description
API	Application Programming Interface
BGP VPN	Border gateway Protocol Virtual Private network
CaaS	Containers as a Service
CI/CD	Continuous Integration/Continuous Deployment
CNF	Containerised Network Function
CNTT	Cloud iNfrastructure Telco Task Force
CPU	Central Processing Unit
DNS	Domain Name System
DPDK	Data Plane Development Kit
DHCP	Dynamic Host Configuration Protocol
ECMP	Equal Cost Multi-Path routing
ETSI	European Telecommunications Standards Institute
FPGA	Field Programmable Gate Array
MB/GB/TB	MegaByte/GigaByte/TeraByte
GPU	Graphics Processing Unit

GRE	Generic Routing Encapsulation
GSM	Global System for Mobile Communications (originally Groupe Spécial Mobile)
GSMA	GSM Association
GSLB	Global Service Load Balancer
GUI	Graphical User Interface
HA	High Availability
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
HW	Hardware
IaaC (also IaC)	Infrastructure as a Code
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IMS	IP Multimedia Sub System
IO	Input/Output
IOPS	Input/Output per Second
IPMI	Intelligent Platform Management Interface
KVM	Kernel-based Virtual Machine
LCM	Life Cycle Management
LDAP	Lightweight Directory Access Protocol
LFN	Linux Foundation Networking
LMA	Logging, Monitoring and Analytics
LVM	Logical Volume Management
MANO	Management And Orchestration
MLAG	Multi-chassis Link Aggregation Group
NAT	Network Address Translation
NFS	Network File System
NFV	Network Function Virtualisation
NFVI	Network Function Virtualisation Infrastructure
NIC	Network Interface Card
NPU	Numeric Processing Unit
NTP	Network Time Protocol
NUMA	Non-Uniform Memory Access
OAI	Open Air Interface
OS	Operating System
OSTK	OpenStack

OPNFV	Open Platform for NFV
OVS	Open vSwitch
OWASP	Open Web Application Security Project
PCIe	Peripheral Component Interconnect Express
PCI-PT	PCIe Passthrough
PXE	Preboot Execution Environment
QoS	Quality of Service
RA	Reference Architecture
RA-2	Reference Architecture 2 (i.e., Reference Architecture for Kubernetes-based Cloud Infrastructure)
RBAC	Role-based Access Control
RBD	RADOS Block Device
REST	Representational state transfer
RI	Reference Implementation
RM	Reference Model
SAST	Static Application Security Testing
SDN	Software Defined Networking
SFC	Service Function Chaining
SG	Security Group
SLA	Service Level Agreement
SMP	Symmetric Multiprocessing
SMT	Simultaneous Multithreading
SNAT	Source Network Address Translation
SNMP	Simple Network Management Protocol
SR-IOV	Single Root Input Output Virtualisation
SSD	Solid State Drive
SSL	Secure Sockets Layer
SUT	System Under Test
TCP	Transmission Control Protocol
TLS	Transport Layer Security
ToR	Top of Rack
TPM	Trusted Platform Module
UDP	User Data Protocol
VIM	Virtualised Infrastructure Manager
VLAN	Virtual LAN
VM	Virtual Machine

VNF	Virtual Network Function
VRRP	Virtual Router Redundancy Protocol
VTEP	VXLAN Tunnel End Point
VXLAN	Virtual eXtensible LAN
WAN	Wide Area Network
ZTA	Zero Trust Architecture

1.5 References

1.6 Conventions

The key words "**MUST**", "**MUST NOT**", "required", "**SHALL**", **SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "recommended", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC 2119 [6].

Table 3References

Ref	Doc Number	Title
[1]	GSMA PRD NG.126 v3.0	Reference Model for Cloud Infrastructure (RM)
[6]	RFC 2119	Key words for use in RFCs to Indicate Requirement Levels
[71]	ETSI GS NFV-SOL 001	ETSI GS NFV-SOL 001: NFV descriptors based on TOSCA specification
[116]	NIST Special Publication 800-204A	Building Secure Microservices-based Applications Using Service-Mesh Architecture
[117]	NIST Special Publication 800-204B	Attribute-based Access Control for Microservices-based Applications Using a Service Mesh
[152]	GSMA PRD NG.133 v2.0	Cloud Infrastructure Reference Architecture managed by OpenStack

1.7 Bibliography

Table 4Bibliography

Ref	Document Title	Source
[2]	Kubernetes Documentation	https://kubernetes.io/docs/home/
[3]	What is Kubernetes	https://kubernetes.io/docs/concepts/overview/
[4]	Kubernetes Github Repository	https://github.com/kubernetes/kubernetes
[5]	Expanded Cloud Native Principles	https://networking.cloud-native-principles.org/cloud-native-principles
[7]	CIS Password Policy Guide	https://www.cisecurity.org/insights/white-papers/cis-password-policy-guide

[8]	CIS Controls List	https://www.cisecurity.org/controls/cis-controls-list
[9]	CVE - Common Vulnerabilities and Exposures	https://cve.mitre.org/
[10]	CVSS - Common Vulnerability Scoring System	https://www.first.org/cvss/
[11]	SBOM - Software Bill Of Materials	https://ntia.gov/page/software-bill-materials
[12]	Cloud Security Alliance	https://cloudsecurityalliance.org/
[13]	OWASP Cheat Sheet Series (OCSS)	https://github.com/OWASP/CheatSheetSeries/
[14]	OWASP Top Ten Security Risks	https://owasp.org/www-project-top-ten/
[15]	OWASP Software Maturity Model (SAMM)	https://owaspsamm.org/
[16]	OWASP Web Security Testing Guide	https://github.com/OWASP/wstg/
[17]	ISO/IEC 27001	https://www.iso.org/obp/ui/#iso:std:iso-iec:27001:ed-2:v1:en
[18]	ISO/IEC 27002	https://www.iso.org/obp/ui/#iso:std:iso-iec:27002:ed-2:v1:en
[19]	ISO/IEC 27032	https://www.iso.org/obp/ui/#iso:std:iso-iec:27032:ed-1:v1:en
[20]	CNCF Kubernetes Conformance	https://github.com/cncf/k8s-conformance
[21]	Kubernetes Alpha API	https://kubernetes.io/docs/reference/using-api/#api-versioning
[22]	Kubernetes Publishing Services (ServiceTypes)	https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types
[23]	Kubernetes Ingress	https://kubernetes.io/docs/concepts/services-networking/ingress/
[24]	Open Container Initiative Runtime Specification, version 1.0.0	https://github.com/opencontainers/runtime-spec/blob/v1.0.0/spec.md
[25]	Kubernetes: Introducing Container Runtime Interface (CRI) in Kubernetes	https://kubernetes.io/blog/2016/12/container-runtime-interface-cri-in-kubernetes/
[26]	Kubernetes Network Custom Resource Definition De-facto Standard	https://github.com/k8snetworkplumbingwg/multi-net-spec/tree/master/v1.2

[27]	Kubernetes CSI Drivers	https://kubernetes-csi.github.io/docs/drivers.html
[28]	Application Service Descriptor (ASD) and packaging Proposals for CNF	https://wiki.onap.org/display/DW/Application+Service+Descriptor+%28ASD%29+and+packaging+Proposals+for+CNF
[29]	Open Container Initiative (OCI) runtime spec	https://github.com/opencontainers/runtime-spec
[30]	Kubernetes Docs - Pod	https://kubernetes.io/docs/concepts/workloads/pods
[31]	Kubernetes Docs - ReplicaSet	https://kubernetes.io/docs/concepts/workloads/controllers/replicaset
[32]	Kubernetes Docs - Deployment	https://kubernetes.io/docs/concepts/workloads/controllers/deployment
[33]	Kubernetes Docs - DaemonSet	https://kubernetes.io/docs/concepts/workloads/controllers/daemonset
[34]	Kubernetes Docs - Job	https://kubernetes.io/docs/concepts/workloads/controllers/job
[35]	Kubernetes Docs - CronJob	https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs
[36]	Kubernetes Docs - StatefulSet	https://kubernetes.io/docs/concepts/workloads/controllers/statefulset
[37]	Kubernetes Docs - CPU Manager	https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies
[38]	Kubernetes Docs - Huge Pages	https://kubernetes.io/docs/tasks/manage-hugepages/scheduling-hugepages
[39]	Kubernetes Docs - Memory Manager	https://kubernetes.io/docs/tasks/administer-cluster/memory-manager
[40]	Kubernetes Docs - Topology Manager	https://kubernetes.io/docs/tasks/administer-cluster/topology-manager
[41]	Kubernetes Node Feature Discovery	https://kubernetes-sigs.github.io/node-feature-discovery/
[42]	Kubernetes Docs - Device Plugin Framework	https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins
[43]	Kubernetes SR-IOV Network Device Plugin	https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin
[44]	Kubernetes Docs - Network Plugins	https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins
[45]	Kubernetes Network Custom Resource Definition	https://github.com/k8snetworkplumbingwg/multi-net-spec
[46]	Kubernetes Docs - Ingress	https://kubernetes.io/docs/concepts/services-networking/ingress

[47]	Kubernetes Docs - Service	https://kubernetes.io/docs/concepts/services-networking/service
[48]	Kubernetes Docs - EndpointSlices	https://kubernetes.io/docs/concepts/services-networking/endpoint-slices
[49]	Kubernetes Docs - Network Policies	https://kubernetes.io/docs/concepts/services-networking/network-policies
[50]	Container Network Interface	https://github.com/containernetworking/cni
[51]	Kubernetes Network Special Interest Group	https://github.com/kubernetes/community/tree/master/sig-network
[52]	Kubernetes Network Plumbing Working Group	https://github.com/k8snetworkplumbingwg/community
[53]	Helm Documentation	https://helm.sh/docs
[54]	Kubernetes Docs - Custom Resources	https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources
[55]	Kubernetes Docs - Custom Resource Definitions	https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions
[56]	Kubernetes Docs - API Server Aggregation	https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/apiserver-aggregation
[57]	Kubernetes Docs - Custom Controllers	https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/#custom-controllers
[58]	Kubernetes Docs - Operator Pattern	https://kubernetes.io/docs/concepts/extend-kubernetes/operator
[59]	Operator Hub	https://operatorhub.io
[60]	Multus CNI	https://github.com/k8snetworkplumbingwg/multus-cni
[61]	AF_XDP Device Plugin	https://github.com/intel/afxdp-plugins-for-kubernetes
[62]	Cloud Native Data Plane	https://cndp.io/
[63]	DPDK AF_XDP Poll Mode Driver	https://doc.dpdk.org/guides/nics/af_xdp.html
[64]	Cert-manager	https://cert-manager.io/
[65]	The Twelve Factor App: Logs	https://12factor.net/logs
[66]	Reference Implementation based on RA2 specifications (RI2)	https://cntt.readthedocs.io/projects/ri2/
[67]	Reference Architecture (RA1) for OpenStack based cloud infrastructure	https://cntt.readthedocs.io/projects/ra1/

[68]	Node Feature Discovery	https://kubernetes-sigs.github.io/node-feature-discovery/stable/get-started/index.html
[69]	Kubernetes Distributions and Platforms document	https://docs.google.com/spreadsheets/d/1uF9BoDzzisHSQemXHIKegMhuythuq_GL3N1mlUUK2h0/
[70]	Kubernetes Supported versions	https://kubernetes.io/releases/version-skew-policy/#supported-versions
[72]	OCI Runtime specification	https://github.com/opencontainers/runtime-spec/blob/master/config.md
[73]	OCI Runtime specification, Mounts Parameter Group	https://github.com/opencontainers/runtime-spec/blob/master/config.md#mounts
[74]	OCI Runtime specification, Process Parameter Group	https://github.com/opencontainers/runtime-spec/blob/master/config.md#process
[75]	OCI Runtime specification, Hostname Parameter Group	https://github.com/opencontainers/runtime-spec/blob/master/config.md#hostname
[76]	OCI Runtime specification, User Parameter Group	https://github.com/opencontainers/runtime-spec/blob/master/config.md#user
[77]	Kubernetes Types of volumes, hostPath	https://kubernetes.io/docs/concepts/storage/volumes/#hostpath
[78]	CNF Testsuite, Rationale, Test if the Helm chart is published: helm_chart_published	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#test-if-the-helm-chart-is-published-helm_chart_published
[79]	CNF Testsuite, Rationale, Test if the Helm chart is valid: helm_chart_valid	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#test-if-the-helm-chart-is-valid-helm_chart_valid
[80]	CNF Testsuite, Rationale, To test if the CNF can perform a rolling update: rolling_update	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-test-if-the-cnf-can-perform-a-rolling-update-rolling_update
[81]	CNF Testsuite, Rationale, To check if a CNF version can be downgraded through a rolling_downgrade: rolling_downgrade	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-a-cnf-version-can-be-downgraded-through-a-rolling_downgrade-rolling_downgrade
[82]	CNF Testsuite, Rationale, To check if the CNF is compatible with different CNIs: cni_compatibility	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-the-cnf-is-compatible-with-different-cnis-cni_compatibility

[83]	CNF Testsuite, Rationale, Test if the CNF crashes when node drain occurs node_drain	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#test-if-the-cnf-crashes-when-node-drain-occurs-node_drain
[84]	CNF Testsuite, Rationale, Test if the CNF crashes when network latency occurs pod_network_latency	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#test-if-the-cnf-crashes-when-network-latency-occurs-pod_network_latency
[85]	CNF Testsuite, Rationale, Test if the CNF crashes when disk fill occurs disk_fill	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#test-if-the-cnf-crashes-when-disk-fill-occurs
[86]	CNF Testsuite, Rationale, Test if the CNF crashes when pod memory hog occurs pod_memory_hog	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#test-if-the-cnf-crashes-when-pod-memory-hog-occurs-pod_memory_hog
[87]	CNF Testsuite, Rationale, Test if the CNF crashes when pod io stress occurs pod_io_stress	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#test-if-the-cnf-crashes-when-pod-io-stress-occurs-pod_io_stress
[88]	CNF Testsuite, Rationale, Test if the CNF crashes when pod network corruption occurs pod_network_corruption	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#test-if-the-cnf-crashes-when-pod-network-corruption-occurs-pod_network_corruption
[89]	CNF Testsuite, Rationale, Test if the CNF crashes when pod network duplication occurs pod_network_duplication	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#test-if-the-cnf-crashes-when-pod-network-duplication-occurs-pod_network_duplication
[90]	CNF Testsuite, Rationale, To test if the CNF uses local storage: no_local_volume_configuration	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-test-if-the-cnf-uses-local-storage-no_local_volume_configuration
[91]	CNF Testsuite, Rationale, To test if there is a liveness entry in the Helm chart: liveness	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-test-if-there-is-a-liveness-entry-in-the-helm-chart-liveness

[92]	CNF Testsuite, Rationale, To test if there is a readiness entry in the Helm chart: readiness	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-test-if-there-is-a-readiness-entry-in-the-helm-chart-readiness
[93]	CNF Testsuite, Rationale, To check if there is automatic mapping of service accounts: service_account_mapping	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-there-is-automatic-mapping-of-service-accounts-service_account_mapping
[94]	CNF Testsuite, Rationale, To check if there is a host network attached to a pod: host_network	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-there-is-a-host-network-attached-to-a-pod-host_network
[95]	CNF Testsuite, Rationale, To check if containers are running with hostPID or hostIPC privileges: host_pid_ipc_privileges	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-containers-are-running-with-hostpid-or-hostipc-privileges-host_pid_ipc_privileges
[96]	CNF Testsuite, Rationale, To check if containers have resource limits defined: resource_policies	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-containers-have-resource-limits-defined-resource_policies
[97]	CNF Testsuite, Rationale, To check if containers have immutable file systems: immutable_file_systems	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-containers-have-immutable-file-systems-immutable_file_systems
[98]	CNF Testsuite, Rationale, To test if there are any (non-declarative) hardcoded IP addresses or subnet masks in the K8s runtime configuration: hardcoded_ip_addresses_in_k8s_runtime_configuration	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-test-if-there-are-any-non-declarative-hardcoded-ip-addresses-or-subnet-masks-in-the-k8s-runtime-configuration-hardcoded_ip_addresses_in_k8s_runtime_configuration
[99]	CNF Testsuite, Rationale, To test the increasing and decreasing of capacity: increase_decrease_capacity	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-test-the-increasing-and-decreasing-of-capacity-increase_decrease_capacity

[100]	CNF Testsuite, Rationale, To check if the CNF has a reasonable image size: reasonable_image_size	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-the-cnf-has-a-reasonable-image-size-reasonable_image_size
[101]	CNF Testsuite, Rationale, To check if the CNF have a reasonable startup time: reasonable_startup_time	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-the-cnf-have-a-reasonable-startup-time-reasonable_startup_time
[102]	CNF Testsuite, Rationale, To check if there are any privileged containers: privileged_containers	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-there-are-any-privileged-containers-privileged_containers
[103]	CNF Testsuite, Rationale, To check if any containers are running as a root user (checks the user outside the container that is running dockerd): non_root_user	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-any-containers-are-running-as-a-root-user-checks-the-user-outside-the-container-that-is-running-dockerd-non_root_user
[104]	CNF Testsuite, Rationale, To check if any containers allow for privilege escalation: privilege_escalation	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-any-containers-allow-for-privilege-escalation-privilege_escalation
[105]	CNF Testsuite, Rationale, To check if containers are running with non-root user with non-root membership: non_root_containers	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-check-if-containers-are-running-with-non-root-user-with-non-root-membership-non_root_containers
[106]	CNF Testsuite, Rationale, To test if there are host ports used in the service configuration: hostport_not_used	https://github.com/cnti-testcatalog/testsuite/blob/main/RATIONALE.md#to-test-if-there-are-host-ports-used-in-the-service-configuration-hostport_not_used
[107]	Kubernetes documentation: Service	https://kubernetes.io/docs/concepts/services-networking/service/
[108]	Kubernetes documentation: LoadBalancer property of the Kubernetes Service	https://kubernetes.io/docs/concepts/services-networking/service/#loadbalancer

[109]	Kubernetes documentation: Immutable ConfigMaps	https://kubernetes.io/docs/concepts/configuration/configmap/#configmap-immutable
[110]	Kubernetes documentation: Horizontal Pod Autoscaling	https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
[111]	Kubernetes documentation: Recommended Labels	https://kubernetes.io/docs/concepts/overview/working-with-objects/common-labels/
[112]	Service Mesh Interface (SMI)	https://smi-spec.io/
[113]	Kubernetes documentation: Images	https://kubernetes.io/docs/concepts/containers/images/
[114]	Cluster API docs	https://cluster-api.sigs.k8s.io/
[115]	Kubernetes Documentation Ports and Protocols	https://kubernetes.io/docs/reference/networking/ports-and-protocols/
[118]	Arstechnica: Tesla cloud resources are hacked to run cryptocurrency-mining malware	https://arstechnica.com/information-technology/2018/02/tesla-cloud-resources-are-hacked-to-run-cryptocurrency-mining-malware/
[119]	Kubernetes Testing Special Interest Group	https://github.com/kubernetes/community/blob/master/sig-testing/charter.md
[120]	Kubernetes End-to-End Testing	https://github.com/kubernetes/community/blob/master/contributors/devel/sig-testing/e2e-tests.md
[121]	Kubernetes Feature gates	https://kubernetes.io/docs/reference/command-line-tools-reference/feature-gates/#feature-gates
[122]	Kubernetes KEP-3136	https://github.com/kubernetes/enhancements/blob/master/keps/sig-architecture/3136-beta-apis-off-by-default/README.md
[123]	Kubernetes API	https://kubernetes.io/docs/reference/using-api/
[124]	Kubernetes API Reference	https://kubernetes.io/docs/reference/kubernetes-api/
[125]	Kubernetes API Groups	https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.31/
[126]	Kubernetes API Machinery Special Interest Group	https://github.com/kubernetes/community/tree/master/sig-api-machinery
[127]	Kubernetes Apps Special Interest Group	https://github.com/kubernetes/community/tree/master/sig-apps
[128]	Kubernetes Auth Special Interest Group	https://github.com/kubernetes/community/tree/master/sig-auth

[129]	Kubernetes Cluster Lifecycle Special Interest Group	https://github.com/kubernetes/community/tree/master/sig-cluster-lifecycle
[130]	Kubernetes Instrumentation Special Interest Group	https://github.com/kubernetes/community/tree/master/sig-instrumentation
[131]	Kubernetes Network Special Interest Group	https://github.com/kubernetes/community/tree/master/sig-network
[132]	Kubernetes Node Special Interest Group	https://github.com/kubernetes/community/tree/master/sig-node
[133]	Kubernetes Scheduling Special Interest Group	https://github.com/kubernetes/community/tree/master/sig-scheduling
[134]	Kubernetes Storage Special Interest Group	https://github.com/kubernetes/community/tree/master/sig-storage
[135]	Kubernetes Feature CrossNamespacePodAffinity	https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#namespace-selector
[136]	Kubernetes Feature PodPriority	https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/
[137]	Kubernetes Feature StorageVersionAPI	https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.23/#storageversion-v1alpha1-internal-apiserver-k8s-io
[138]	Kubernetes Feature DaemonSetUpdateSurge	https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.23/#rollingupdatedaemonset-v1-apps
[139]	Kubernetes Feature IndexedJob	https://kubernetes.io/docs/concepts/workloads/controllers/job/
[140]	Kubernetes Feature StatefulSet	https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/
[141]	Kubernetes Feature SuspendJob	https://kubernetes.io/docs/concepts/workloads/controllers/job/#suspending-a-job
[142]	Kubernetes Feature TaintEviction	https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/#taint-based-evictions
[143]	Kubernetes Feature TTLAfterFinished	https://kubernetes.io/docs/concepts/workloads/controllers/ttlafterfinished/
[144]	Kubernetes Feature BoundServiceAccountTokenVolume	https://github.com/kubernetes/enhancements/blob/master/keps/sig-auth/1205-bound-service-account-tokens/README.md
[145]	Kubernetes Feature IPv6DualStack	https://kubernetes.io/docs/concepts/services-networking/dual-stack/
[146]	Kubernetes Extend Service IP Ranges	https://kubernetes.io/docs/tasks/network/extend-service-ip-ranges/
[147]	Kubernetes Feature ProbeTerminationGracePeriod	https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/#probe-level-terminationgraceperiodseconds

[148]	Kubernetes Feature DownwardAPIHugePages	https://kubernetes.io/docs/tasks/inject-data-application/downward-api-volume-expose-pod-information
[149]	Kubernetes Feature PodReadinessGate	https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#pod-readiness-gate
[150]	Kubernetes Feature Sysctls	https://kubernetes.io/docs/tasks/administer-cluster/sysctl-cluster/
[151]	Kubernetes Feature LocalStorageCapacityIsolation	https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
[153]	Docker	https://www.docker.com/
[154]	Xtesting Python package	https://pypi.org/project/xtesting/
[155]	Test case execution description	https://git.opnfv.org/functest-xtesting/tree/docker/core/testcases.yaml
[156]	CI/CD toolchains in a few commands	https://github.com/collivier/ansible-role-xtesting
[157]	non-disruptive-conformance	https://sonobuoy.io/docs/main/e2eplugin/
[158]	Sonobuoy	https://sonobuoy.io/
[159]	upstream Kubernetes gate	https://github.com/kubernetes/test-infra/blob/master/config/jobs/kubernetes/sig-release/release-branch-jobs/1.29.yaml
[160]	Anuket RC2 verification	http://104.154.71.112:8080/job/functest-kubernetes-v1.29-daily
[161]	deprecated in-tree GitRepo volume type	https://github.com/kubernetes-sigs/kind/issues/2356
[162]	xrally-kubernetes	https://github.com/xrally/xrally-kubernetes
[163]	Functest Kubernetes Benchmarking	https://git.opnfv.org/functest-kubernetes/tree/docker/benchmarking/testcases.yaml?h=stable%2Fv1.29
[164]	xrally_kubernetes_full	https://artifacts.opnfv.org/functest-kubernetes/671YK0WH9PRK/functest-kubernetes-opnfv-functest-kubernetes-benchmarking-v1.29-xrally_kubernetes_full-run-9/xrally_kubernetes_full/xrally_kubernetes_full.html
[165]	Kubernetes perf-tests repository	https://artifacts.opnfv.org/functest-kubernetes/671YK0WH9PRK/functest-kubernetes-opnfv-functest-kubernetes-benchmarking-v1.29-xrally_kubernetes_full-run-9/xrally_kubernetes_full/xrally_kubernetes_full.html
[166]	netperf	https://github.com/kubernetes/perf-tests/tree/master/network/benchmarks/netperf
[167]	iperf	https://github.com/esnet/iperf

[168]	Netperf	https://github.com/HewlettPackard/netperf/
[169]	Functest Kubernetes Security	https://git.opnfv.org/functest-kubernetes/tree/docker/security/testcases.yaml?h=stable%2Fv1.29
[170]	kube-hunter	https://github.com/aquasecurity/kube-hunter
[171]	kube-bench	https://github.com/aquasecurity/kube-bench
[172]	CIS Kubernetes Benchmark	https://www.cisecurity.org/benchmark/kubernetes/
[173]	vulnerability categories	https://github.com/aquasecurity/kube-hunter/blob/v0.6.8/kube_hunter/core/events/types.py
[174]	Clearwater IMS	https://github.com/Metaswitch/clearwater-docker
[175]	clearwater-live-test	https://github.com/Metaswitch/clearwater-live-test
[176]	Xtesting CI	https://galaxy.ansible.com/collivier/xtesting
[177]	Deploy your own Xtesting CI/CD toolchains	https://github.com/collivier/ansible-role-xtesting
[178]	systemd for Docker	https://docs.docker.com/config/daemon/systemd/#httphttps-proxy
[179]	Alibaba Cloud Blog: What Can We Learn from Twitter's Move to Kubernetes	https://www.alibabacloud.com/blog/twitter-announced-switch-from-mesos-to-kubernetes_595156
[180]	YouTube: Kubernetes Failure Stories, or: How to Crash Your Cluster - Henning Jacobs	https://www.youtube.com/watch?v=LpFApeaGv7A
[181]	CNCF Blog: Demystifying Kubernetes as a service – How Alibaba cloud manages 10,000s of Kubernetes clusters	https://www.cncf.io/blog/2019/12/12/demystifying-kubernetes-as-a-service-how-does-alibaba-cloud-manage-10000s-of-kubernetes-clusters/
[182]	GitHub: Multus-CNI	https://github.com/k8snetworkplumbingwg/multus-cni
[183]	Google Docs: KEP: MultiNetwork podNetwork object	https://docs.google.com/document/d/17LhyXsEgjNQ0NWtvqvtgJwVqdJWreizsgAZHWflgP-A/edit
[184]	Kubernetes Docs: User Namespaces	https://kubernetes.io/docs/concepts/workloads/pods/user-namespaces/
[185]	KEP-127: Support User Namespaces in stateless pods	https://github.com/kubernetes/enhancements/tree/master/keps/sig-node/127-user-namespaces
[186]	Wikipedia: Linux Namespaces	https://en.wikipedia.org/wiki/Linux_namespaces

2 Architecture Requirements

2.1 Introduction for Architecture Requirements

This chapter will specialise the requirements defined in the overall Reference Model into Kubernetes-specific requirements. Additional, RA2-specific, entries are included in section Kubernetes Architecture Requirements.

2.2 Key Word Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [6].

2.3 Reference Model Requirements

The tables below contain the requirements from the Reference Model to cover all infrastructure profiles. The table also includes a reference to any linked specification at Component Level Architecture and at Security Guidance to ensure traceability. If the related Specification does not exist, the reference will read "N/A" (and in bold "**N/A**" for mandatory requirements).

To ensure alignment with the infrastructure profile catalogue, the following requirements are referenced through:

- Those relating to Cloud Infrastructure Software Profiles
- Those relating to Cloud Infrastructure Hardware Profiles
- Those relating to Cloud Infrastructure Management
- Those relating to Cloud Infrastructure Security

2.3.1 Cloud Infrastructure Software Profile Capabilities

Table 5Reference Model Requirements: Cloud Infrastructure Software Profile Capabilities

Reference Model Section [1]	Reference	Description	Requirement for Basic Profile	Requirement for High-Performance Profile	Specification Reference
Exposed Infrastructure Capabilities	e.cap.001	Max number of vCPU that can be assigned to a single Pod by the Cloud Infrastructure	At least 16	At least 16	ra2.ch.011
Exposed Infrastructure Capabilities	e.cap.002	Max memory in MB that can be assigned to a single Pod by the Cloud Infrastructure	at least 32 GB	at least 32 GB	ra2.ch.012
Exposed Infrastructure Capabilities	e.cap.003	Max storage in GB that can be assigned to a	at least 320 GB	at least 320 GB	ra2.ch.010

		single Pod by the Cloud Infrastructure			
Exposed Infrastructure Capabilities	e.cap.004	Max number of connection points that can be assigned to a single Pod by the Cloud Infrastructure	6	6	ra2.ntw.003
Exposed Infrastructure Capabilities	e.cap.005	Max storage in GB that can be attached / mounted to Pod by the Cloud Infrastructure	Up to 16TB (1)	Up to 16TB (1)	N/A
Profiles Specifications & Capability Mapping	e.cap.006	CPU pinning support	Optional	Must support	ra2.k8s.009
Profiles Specifications & Capability Mapping	e.cap.007	NUMA support	Optional	Must support	ra2.k8s.006
Exposed Infrastructure Capabilities	e.cap.008	IPsec Acceleration using the virtio-ipsec interface	Optional	Optional	N/A
Exposed Infrastructure Capabilities	e.cap.009	Crypto Acceleration using the virtio-crypto interface	Optional	Optional	N/A
Exposed Infrastructure Capabilities	e.cap.010	Transcoding Acceleration	Optional	Optional	N/A
Exposed Infrastructure Capabilities	e.cap.011	Programmable Acceleration	Optional	Optional	N/A
Exposed Infrastructure Capabilities	e.cap.012	Enhanced Cache Management: L=Lean; E=Equal; X=eXpanded	E	E	N/A
Profiles Specifications & Capability Mapping	e.cap.013	SR-IOV over PCI-PT	Optional	Must support	ra2.ch.002 ra2.ch.003 ra2.k8s.007 ra2.ntw.004 ra2.ntw.008

Exposed Infrastructure Capabilities	e.cap.014	Hardware coprocessor support (GPU/NPU)	Optional	Optional	N/A
Exposed Infrastructure Capabilities	e.cap.015	SmartNICs	Optional	Optional	N/A
Exposed Infrastructure Capabilities	e.cap.016	FPGA/other Acceleration H/W	Optional	Optional	ra2.k8s.007 ra2.ntw.012
Exposed Infrastructure Capabilities	e.cap.017	Ability to monitor L2-L7 data from workload	n/a (2)	*n/a (2)*	N/A
Exposed Infrastructure Capabilities	e.cap.026	Real-Time settings	No	Mandatory for flavour rt-tsn	
Exposed Infrastructure Capabilities	e.cap.027	Time Sensitive Networking with PTP Hardware Clock and syncronization with SyncE	No	Mandatory for flavour rt-tsn	
Internal Infrastructure Capabilities	i.cap.014	Specifies the proportion of vCPUs consumed by the Cloud Infrastructure system on the worker nodes.	2	2	ra2.k8s.008
Internal Infrastructure Capabilities	i.cap.015	Indicates the memory consumed by Cloud Infrastructure on the worker nodes	16 GB	16 GB	
Internal Infrastructure Capabilities	i.cap.016	Number of virtual cores per physical core; also known as CPU overbooking ratio that is required	01:01	01:01	ra2.ch.004, ra2.ch.005 Kubernetes Node
Internal Infrastructure Capabilities	i.cap.017	QoS enablement of the connection	Optional	Must support	N/A

		point (vNIC or interface)			
Internal Infrastructure Capabilities	i.cap.018	Support for huge pages	Optional	Must support	ra2.ch.001
Internal Infrastructure Capabilities	i.pm.001	Monitor worker node CPU usage, per nanosecond	Must support	Must support	N/A
Internal Infrastructure Capabilities	i.pm.002	Monitor Pod CPU usage, per nanosecond	Must support	Must support	N/A
Internal Infrastructure Capabilities	i.pm.003	Monitor worker node CPU utilisation (%)	Must support	Must support	N/A
Internal Infrastructure Capabilities	i.pm.004	Monitor Pod CPU utilisation	Must support	Must support	N/A
Internal Infrastructure Capabilities	i.pm.005	Measure external storage IOPs	Must support	Must support	N/A
Internal Infrastructure Capabilities	i.pm.006	Measure external storage throughput	Must support	Must support	N/A
Internal Infrastructure Capabilities	i.pm.007	Measure external storage capacity	Must support	Must support	N/A
Profiles Specifications & Capability Mapping	i.os.001	Host operating system must provide drivers etc. to support listed capabilities.	Must support	Must support	ra2.ch.004

(1) Defined in the .bronze configuration in RM section Storage extensions [1].

(2) In Kubernetes based infrastructures packet monitoring is out of the scope for the infrastructure.

2.3.2 Virtual Network Interface Specifications

Note: The required number of connection points to a Pod is described in e.cap.004 above. This section describes the required bandwidth of those connection points.

Table 6Reference Model Requirements: Network Interface Specifications

Reference Model Section [1]	Reference	Description	Requirement for Basic Profile	Requirement for High-Performance Profile	Specification Reference

Virtual Network Interface Specifications	n1, n2, n3, n4, n5, n6	1, 2, 3, 4, 5, 6 Gbps	Must support	Must support	N/A
Virtual Network Interface Specifications	n10, n20, n30, n40, n50, n60	10, 20, 30, 40, 50, 60 Gbps	Must support	Must support	N/A
Virtual Network Interface Specifications	n25, n50, n75, n100, n125, n150	25, 50, 75, 100, 125, 150 Gbps	Must support	Must support	N/A
Virtual Network Interface Specifications	n50, n100, n150, n200, n250 , n300	50, 100, 150, 200, 250, 300 Gbps	Must support	Must support	N/A
Virtual Network Interface Specifications	n100, n200, n300, n400, n500, n600	100, 200, 300, 400, 500, 600 Gbps	Must support	Must support	N/A

Virtual Network Interface Specifications

2.3.3 Cloud Infrastructure Software Profile Requirements

Table 7Reference Model Requirements: Cloud Infrastructure Software Profile Requirements

Reference Model Section [1]	Reference	Description	Requirement for Basic Profile	Requirement for High-Performance Profile	Specification Reference
Virtual Compute	infra.com.cfg .001	CPU allocation ratio	1:1	1:1	ra2.ch.005, ra2.ch.006
Virtual Compute	infra.com.cfg .002	NUMA awareness	Optional	Must support	ra2.k8s.006
Virtual Compute	infra.com.cfg .003	CPU pinning capability	Optional	Must support	ra2.k8s.009
Virtual Compute	infra.com.cfg .004	Huge pages	Optional	Must support	ra2.ch.001
Virtual Storage	infra.stg.cfg. 002	Storage Block	Must support	Must support	ra2.stg.004
Virtual Storage	infra.stg.cfg. 003	Storage with replication	Optional	Must support	N/A

Virtual Storage	infra.stg.cfg.004	Storage with encryption	Must support	Must support	N/A
Virtual Storage	infra.stg.acc.cfg.001	Storage IOPS oriented encryption	Optional	Must support	N/A
Virtual Storage	infra.stg.acc.cfg.002	Storage capacity-oriented encryption	Optional	Optional	N/A
Virtual Networking	infra.net.cfg.001	IO virtualisation using virtio1.1	Must support (1)	Must support (1)	N/A
Virtual Networking	infra.net.cfg.002	The overlay network encapsulation protocol needs to enable ECMP in the underlay to take advantage of the scale-out features of the network fabric.(2)	Must support VXLAN, MPLSoUDP, GENEVE, other	No requirement specified	N/A
Virtual Networking	infra.net.cfg.003	Network Address Translation	Must support	Must support	N/A
Virtual Networking	infra.net.cfg.004	Security Groups	Must support	Must support	ra2.k8s.014
Virtual Networking	infra.net.cfg.006	Traffic patterns symmetry	Must support	Must support	N/A
Virtual Networking	infra.net.acc.cfg.001	vSwitch optimisation	Optional	Must support DPDK (3)	ra2.ntw.010
Virtual Networking	infra.net.acc.cfg.002	Support of HW offload	Optional	Optional, SmartNIC	N/A
Virtual Networking	infra.net.acc.cfg.003	Crypto acceleration	Optional	Optional	N/A
Virtual Networking	infra.net.acc.cfg.004	Crypto Acceleration Interface	Optional	Optional	N/A
Virtual Networking	infra.net.acc.cfg.005	AF_XDP	Optional	Optional	N/A

Virtual Networking

(1) Might have other interfaces (such as SR-IOV VFs to be directly passed to a VM or a Pod) or NIC-specific drivers on Kubernetes nodes.

(2) In Kubernetes based infrastructures network separation is possible without an overlay (e.g.: with IPVLAN)

(3) This feature is not applicable for Kubernetes based infrastructures due to lack of vSwitch however workloads need access to user space networking solutions.

2.3.4 Cloud Infrastructure Hardware Profile Requirements

Table 8Reference Model Requirements: Cloud Infrastructure Hardware Profile Requirements

Reference Model Section [1]	Reference	Description	Requirement for Basic Profile	Requirement for High-Performance Profile	Specification Reference
Compute Resources	infra.hw.cpu.cfg.001	Minimum number of CPU sockets	2	2	ra2.ch.008
Compute Resources	infra.hw.cpu.cfg.002	Minimum number of cores per CPU	20	20	ra2.ch.008
Compute Resources	infra.hw.cpu.cfg.003	NUMA alignment	N	Y	ra2.ch.008
Compute Resources	infra.hw.cpu.cfg.004	Simultaneous multithreading/ Symmetric multiprocessing (SMT/SMP)	Must support if available	Optional	ra2.ch.004
Compute Resources	infra.hw.cac.cfg.001	GPU	Optional	Optional	N/A
Storage Configurations`	infra.hw.stg.hdd.cfg.001	Local storage HDD	No requirement specified	No requirement specified	N/A
Storage Configurations`	infra.hw.stg.ssd.cfg.002	Local storage SSD	Should support	Should support	ra2.ch.009
Network Resources	infra.hw.nic.cfg.001	Total number of NIC ports	4	4	ra2.ch.013

		available in the host			
Network Resources	infra.hw.nic.cfg.002	Port speed specified in Gbps (minimum values)	10	25	ra2.ch.014, ra2.ch.015
Network Resources	infra.hw.pci.cfg.001	Number of PCIe slots available in the host	8	8	ra2.ch.016
Network Resources	infra.hw.pci.cfg.002	PCIe speed	Gen 3	Gen 3	ra2.ch.016
Network Resources	infra.hw.pci.cfg.003	PCIe lanes	8	8	ra2.ch.016
Network Resources	infra.hw.nac.cfg.001	Cryptographic acceleration	Optional	Optional	N/A
Network Resources	infra.hw.nac.cfg.002	A SmartNIC that is used to offload vSwitch functionality to hardware	Optional	Optional (1)	N/A
Network Resources	infra.hw.nac.cfg.003	Compression	Optional	Optional	N/A

(1) There is no vSwitch in case of containers, but a SmartNIC can be used to offload any other network processing.

2.3.5 Edge Cloud Infrastructure Hardware Profile Requirements

In the case of Telecom Edge Cloud Deployments, hardware requirements can differ from the above to account for environmental and other constraints. The Reference Model [1] includes considerations specific to deployments at the edge of the network. The infrastructure profiles "Basic" and "High Performance" as per the RM chapter on Profiles and Workload Flavours still apply, but a number of requirements of the above table are relaxed as follows:

Table 9Reference Model Requirements: Edge Cloud Infrastructure Hardware Profile Requirements

Reference Model Section [1]	Reference	Description	Requirement for Basic Profile	Requirement for High-Performance Profile	Specification Reference
Telecom Edge Cloud: Infrastructure Profiles	infra.hw.cpu.cfg.001	sockets			

Telecom Edge Cloud: Infrastructure Profiles	infra.hw.cpu.cfg.002	Minimum number of Cores per CPU	1	1	ra2.ch.008
Telecom Edge Cloud: Infrastructure Profiles	infra.hw.cpu.cfg.003	NUMA alignment	N	Y (1)	ra2.ch.008

Telecom Edge Cloud: Infrastructure Profiles.

(1) immaterial if the number of CPU sockets (infra.hw.cpu.cfg.001) is 1.

2.3.6 Cloud Infrastructure Management Requirements

Table 10Reference Model Requirements: Cloud Infrastructure Management Requirements

Reference Model Section [1]	Reference	Description	Requirement (common to all Profiles)	Specification Reference
Cloud Infrastructure Management Capabilities	e.man.001	Capability to allocate virtual compute resources to a workload	Must support	N/A
Cloud Infrastructure Management Capabilities	e.man.002	Capability to allocate virtual storage resources to a workload	Must support	N/A
Cloud Infrastructure Management Capabilities	e.man.003	Capability to allocate virtual networking resources to a workload	Must support	N/A
Cloud Infrastructure Management Capabilities	e.man.004	Capability to isolate resources between tenants	Must support	N/A
Cloud Infrastructure Management Capabilities	e.man.005	Capability to manage workload software images	Must support	N/A
Cloud Infrastructure Management Capabilities	e.man.006	Capability to provide information related to allocated virtualised resources per tenant	Must support	N/A
Cloud Infrastructure Management Capabilities	e.man.007	Capability to notify state changes of allocated resources	Must support	N/A

Cloud Infrastructure Management Capabilities	e.man.008	Capability to collect and expose performance information on virtualised resources allocated	Must support	N/A
Cloud Infrastructure Management Capabilities	e.man.009	Capability to collect and notify fault information on virtualised resources	Must support	N/A

Cloud Infrastructure Management Capabilities.

2.3.7 Cloud Infrastructure Monitoring Capabilities

Table 11Reference Model Requirements: Cloud Infrastructure Internal Performance Measurement Requirements

Reference Model Section [1]	Reference	Description	Requirement (common to all Profiles)	Specification Reference
Internal Performance Measurement Capabilities	i.pm.001	Capability to monitor host CPU Usage (in ns)	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.002	Capability to monitor per Pod CPU (Virtual compute resource) usage (in ns)	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.003	Capability to monitor host CPU Usage (in percentage)	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.004	Capability to monitor per Pod CPU (Virtual compute resource) usage (in percentage)	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.005	Capability to monitor packet count per physical or virtual node network interface	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.006	Capability to monitor octet (bytes) count per physical or virtual node network interface	Must support	N/A
Internal Performance	i.pm.007	Capability to monitor dropped packet count per physical or virtual	Must support	N/A

Measurement Capabilities		node network interface		
Internal Performance Measurement Capabilities	i.pm.008	Capability to monitor errored packet count per physical or virtual node network interface	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.009	Capability to monitor amount of buffered memory (in KiB) on the node.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.010	Capability to monitor amount of cached memory (in KiB) on the node.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.011	Capability to monitor amount of free memory (in KiB) on the node.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.012	Capability to monitor amount of slab memory (in KiB) on the node.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.013	Capability to monitor amount of total memory (in KiB) on the node.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.014	Capability to monitor amount of free storage space (in B) on the node and on volumes.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.015	Capability to monitor amount of used storage space (in B) on the node and on volumes.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.016	Capability to monitor amount of reserved storage space (in B) on the node and on volumes.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.017	Capability to monitor the storage read latency (in ms) on the node and on volumes.	Must support	N/A
Internal Performance	i.pm.018	Capability to monitor the read operations	Must support	N/A

Measurement Capabilities		rate (in IOPS) on the node and on volumes.		
Internal Performance Measurement Capabilities	i.pm.019	Capability to monitor the storage read throughput (in B/s) on the node and on volumes.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.020	Capability to monitor the storage write latency (in ms) on the node and on volumes.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.021	Capability to monitor the write operations rate (in IOPS) on the node and on volumes.	Must support	N/A
Internal Performance Measurement Capabilities	i.pm.022	Capability to monitor the storage write throughput (in B/s) on the node and on volumes.	Must support	N/A

Internal Performance Measurement Capabilities.

2.3.8 Cloud Infrastructure Security Requirements

Table 12Reference Model Requirements: Cloud Infrastructure Security Requirements

Reference Model Section [1]	Reference	Description	Specification Reference
System Hardening	sec.gen.001	The Platform **must** maintain the specified configuration.	
System Hardening	sec.gen.002	All systems part of Cloud Infrastructure **must** support password hardening as defined in the CIS Password Policy Guide [7]. Hardening: CIS Password Policy Guide	Node Hardening: Securing Kubernetes Hosts
System Hardening	sec.gen.003	All servers part of Cloud Infrastructure **must** support a root of trust and secure boot.	
System Hardening	sec.gen.004	The Operating Systems of all the server's part of Cloud Infrastructure **must** be hardened by removing or disabling unnecessary services, applications, and network protocols, configuring operating system user authentication, configuring resource controls, installing and configuring additional security controls where needed, and testing the security of	Security Principles and Node Hardening

		the Operating System. (NIST SP 800-123)	
System Hardening	sec.gen.005	The Platform **must** support Operating System level access control	Node Hardening
System Hardening	sec.gen.006	The Platform **must** support secure logging. Logging with root account must be prohibited when root privileges are not required.	Restricting Direct Access to Nodes
System Hardening	sec.gen.007	All servers part of Cloud Infrastructure **must** be Time synchronized with authenticated Time service.	
System Hardening	sec.gen.008	All servers part of Cloud Infrastructure **must** be regularly updated to address security vulnerabilities.	Vulnerability Assessment
System Hardening	sec.gen.009	The Platform **must** support Software integrity protection and verification and **must** scan source code and manifests.	Securing the Kubernetes Orchestrator
System Hardening	sec.gen.010	The Cloud Infrastructure **must** support encrypted storage, for example, block, object and file storage, with access to encryption keys restricted based on a need to know. Controlled Access Based on the Need to Know [8].	
System Hardening	sec.gen.011	The Cloud Infrastructure **should** support Read and Write only storage partitions (write only permission to one or more authorized actors).	
System Hardening	sec.gen.012	The Operator **must** ensure that only authorized actors have physical access to the underlying infrastructure.	
System Hardening	sec.gen.013	The Platform **must** ensure that only authorized actors have logical access to the underlying infrastructure.	Securing the Kubernetes Orchestrator
System Hardening	sec.gen.014	All servers part of Cloud Infrastructure **should** support measured boot and an attestation server that monitors the measurements of the servers.	
System Hardening	sec.gen.015	Any change to the Platform must be logged as a security event, and the logged event must include the	

		identity of the entity making the change, the change, the date and the time of the change.	
Platform and Access	sec.sys.001	The Platform **must** support authenticated and secure access to API, GUI and command line interfaces.	Securing the Kubernetes Orchestrator
Platform and Access	sec.sys.002	The Platform **must** support Traffic Filtering for workloads (for example, Firewall).	
Platform and Access	sec.sys.003	The Platform **must** support secure and encrypted communications, and confidentiality and integrity of network traffic.	Network Resources Use Transport Layer Security and Service Mesh
Platform and Access	sec.sys.004	The Cloud Infrastructure **must** support authentication, integrity, and confidentiality on all network channels.	Network Resources Use Transport Layer Security and Service Mesh
Platform and Access	sec.sys.005	The Cloud Infrastructure **must** separate the underlay and overlay networks.	
Platform and Access	sec.sys.006	The Cloud Infrastructure must be able to utilise the Cloud Infrastructure Manager identity lifecycle management capabilities.	Security Principles
Platform and Access	sec.sys.007	The Platform **must** implement controls enforcing separation of duties and privileges, least privilege use and least common mechanism (Role-Based Access Control).	Security Principles Securing the Kubernetes Orchestrator
Platform and Access	sec.sys.008	The Platform **must** be able to assign the Entities that comprise the tenant networks to different trust domains. Communication between different trust domains is not allowed, by default.	
Platform and Access	sec.sys.009	The Platform **must** support creation of Trust Relationships between trust domains.	
Platform and Access	sec.sys.010	For two or more domains without existing trust relationships, the Platform **must not** allow the effect of an attack on one domain to impact the other domains either directly or indirectly.	
Platform and Access	sec.sys.011	The Platform **must not** reuse the same authentication credential	

		(e.g., key-pair) on different Platform components (e.g., on different hosts, or different services).	
Platform and Access	sec.sys.012	The Platform **must** protect all secrets by using strong encryption techniques, and storing the protected secrets externally from the component	
Platform and Access	sec.sys.013	The Platform **must** provide secrets dynamically as and when needed.	
Platform and Access	sec.sys.014	The Platform **should** use Linux Security Modules such as SELinux to control access to resources.	
Platform and Access	sec.sys.015	The Platform **must not** contain back door entries (unpublished access points, APIs, etc.).	
Platform and Access	sec.sys.016	Login access to the platform's components **must** be through encrypted protocols such as SSH v2 or TLS v1.2 or higher. Note: Hardened jump servers isolated from external networks are recommended	Securing the Kubernetes Orchestrator
Platform and Access	sec.sys.017	The Platform **must** provide the capability of using digital certificates that comply with X.509 standards issued by a trusted certificate authority.	
Platform and Access	sec.sys.018	The Platform **must** provide the capability of allowing certificate renewal and revocation.	
Platform and Access	sec.sys.019	The Platform **must** provide the capability of testing the validity of a digital certificate (CA signature, validity period, non-revocation, identity).	
Platform and Access	sec.sys.020	The Cloud Infrastructure architecture **should** rely on Zero Trust principles to build a secure by design environment.	
Confidentiality and Integrity	sec.ci.001	The Platform **must** support Confidentiality and Integrity of data at rest and in-transit. by design environment.	Securing the Kubernetes Orchestrator
Confidentiality and Integrity	sec.ci.002	The Platform **should** support self-encrypting storage devices.	

Confidentiality and Integrity	sec.ci.003	The Platform **must** support confidentiality and integrity of data related metadata.	
Confidentiality and Integrity	sec.ci.004	The Platform **must** support confidentiality of processes and restrict information sharing with only the process owner (e.g., tenant).	
Confidentiality and Integrity	sec.ci.005	The Platform **must** support confidentiality and integrity of process-related metadata and restrict information sharing with only the process owner (e.g., tenant).	
Confidentiality and Integrity	sec.ci.006	The Platform **must** support confidentiality and integrity of workload resource utilization (RAM, CPU, storage, network I/O, cache, hardware offload) and restrict information sharing with only the workload owner (e.g., tenant).	
Confidentiality and Integrity	sec.ci.007	The Platform **must not** allow memory inspection by any actor other than the authorized actors for the entity to which memory is assigned (e.g., tenants owning the workload), for Lawful Inspection, and by secure monitoring services.	
Confidentiality and Integrity	sec.ci.008	The Cloud Infrastructure **must** support tenant networks segregation.	Create and define Network Policies
Confidentiality and Integrity	sec.ci.009	For sensitive data encryption, the key management service **should** leverage a Hardware Security Module to manage and protect cryptographic keys.	
Workload Security	sec.wl.001	The Platform **must** support workload placement policy.	
Workload Security	sec.wl.002	The Cloud Infrastructure **must** provide methods to ensure the platform's trust status and integrity (e.g., remote attestation, Trusted Platform Module).	
Workload Security	sec.wl.003	The Platform **must** support secure provisioning of workloads.	Securing the Kubernetes Orchestrator
Workload Security	sec.wl.004	The Platform **must** support location assertion (for mandated in-country or location requirements).	

Workload Security	sec.wl.005	The Platform **must** support the separation of production and non-production Workloads.	Securing the Kubernetes Orchestrator
Workload Security	sec.wl.006	The Platform **must** support the separation of Workloads based on their categorisation (for example, payment card information, healthcare, etc.).	Securing the Kubernetes Orchestrator
Workload Security	sec.wl.007	The Operator **must** implement processes and tools to verify VNF authenticity and integrity.	Trusted Registry
Image Security	sec.img.001	Images from untrusted sources **must not** be used.	Trusted Registry
Image Security	sec.img.002	Images **must** be scanned to be maintained free from known vulnerabilities.	Trusted Registry
Image Security	sec.img.003	Images **must not** be configured to run with privileges higher than the privileges of the actor authorized to run them.	Runtime Security
Image Security	sec.img.004	Images **must** only be accessible to authorised actors.	
Image Security	sec.img.005	Image Registries **must** only be accessible to authorised actors.	
Image Security	sec.img.006	Image Registries **must** only be accessible over secure networks that enforce authentication, integrity and confidentiality.	Trusted Registry
Image Security	sec.img.007	Image registries **must** be clear of vulnerable and out of date versions.	Trusted Registry
Image Security	sec.img.008	Images **must not** include any secrets. Secrets include passwords, cloud provider credentials, SSH keys, TLS certificate keys, etc.	Secrets Management
Image Security	sec.img.009	CIS Hardened Images **should** be used whenever possible.	
Image Security	sec.img.010	Minimalist base images **should** be used whenever possible.	
Security LCM	sec.lcm.001	The Platform **must** support secure provisioning, availability, and deprovisioning (secure clean-Up) of workload resources where secure clean-Up includes tear-down, defense against virus or other attacks.	

Security LCM	sec.lcm.002	Cloud operations staff and systems **must** use management protocols limiting security risk such as SNMPv3, SSH v2, ICMP, NTP, syslog and TLS v1.2 or higher.	Securing the Kubernetes Orchestrator
Security LCM	sec.lcm.003	The Cloud Operator **must** implement and strictly follow change management processes for Cloud Infrastructure, Cloud Infrastructure Manager and other components of the cloud, and platform change control on hardware.	
Security LCM	sec.lcm.004	The Cloud Operator **should** support automated templated approved changes.	
Security LCM	sec.lcm.005	Platform **must** provide logs and these logs must be regularly monitored for anomalous behavior.	Enabling Logging and Monitoring
Security LCM	sec.lcm.006	The Platform **must** verify the integrity of all Resource management requests.	
Security LCM	sec.lcm.007	The Platform **must** be able to update newly instantiated, suspended, hibernated, migrated and restarted images with current time information.	Securing the Kubernetes Orchestrator
Security LCM	sec.lcm.008	The Platform **must** be able to update newly instantiated, suspended, hibernated, migrated and restarted images with relevant DNS information.	
Security LCM	sec.lcm.009	The Platform **must** be able to update the tag of newly instantiated, suspended, hibernated, migrated and restarted images with relevant geolocation (geographic) information.	
Security LCM	sec.lcm.010	The Platform **must** log all changes to geolocation along with the mechanisms and sources of location information (i.e., GPS, IP block, and timing).	
Security LCM	sec.lcm.011	The Platform **must** implement security life cycle management processes including the proactive update and patching of all deployed Cloud Infrastructure software.	

Security LCM	sec.lcm.012	The Platform **must** log any access privilege escalation.	
Monitoring and Security Audit	sec.mon.001	The Platform **must** provide logs and these logs must be regularly monitored for events of interest. The logs **must** contain the following fields: event type, date/time, protocol, service or program used for access, success/failure, login ID or process ID, IP address and ports (source and destination) involved.	
Monitoring and Security Audit	sec.mon.002	Security logs **must** be time synchronised.	
Monitoring and Security Audit	sec.mon.003	The Platform **must** log all changes to time server source, time, date and time zones.	
Monitoring and Security Audit	sec.mon.004	The Platform **must** secure and protect audit logs (containing sensitive information) both in-transit and at rest.	
Monitoring and Security Audit	sec.mon.005	The Platform **must** monitor and audit various behaviours of connection and login attempts to detect access attacks and potential access attempts and take corrective actions accordingly.	
Monitoring and Security Audit	sec.mon.006	The Platform **must** monitor and audit operations by authorized account access after login to detect malicious operational activity and take corrective actions accordingly.	
Monitoring and Security Audit	sec.mon.007	The Platform **must** monitor and audit security parameter configurations for compliance with defined security policies.	
Monitoring and Security Audit	sec.mon.008	The Platform **must** monitor and audit externally exposed interfaces for illegal access (attacks) and take corrective security hardening measures.	
Monitoring and Security Audit	sec.mon.009	The Platform **must** monitor and audit service handling for various attacks (malformed messages, signalling flooding and replaying, etc.) and take corrective actions accordingly.	
Monitoring and Security Audit	sec.mon.010	The Platform **must** monitor and audit running processes to detect unexpected or unauthorized	

		processes and take corrective actions accordingly.	
Monitoring and Security Audit	sec.mon.011	The Platform **must** monitor and audit logs from infrastructure elements and workloads to detected anomalies in the system components and take corrective actions accordingly.	
Monitoring and Security Audit	sec.mon.012	The Platform **must** monitor and audit traffic patterns and volumes to prevent malware download attempts.	
Monitoring and Security Audit	sec.mon.013	The monitoring system **must not** affect the security (integrity and confidentiality) of the infrastructure, workloads, or the user data (through back door entries).	
Monitoring and Security Audit	sec.mon.014	The Monitoring systems **should not** impact IAAS, PAAS, and SAAS SLAs including availability SLAs.	
Monitoring and Security Audit	sec.mon.015	The Platform **must** ensure that the monitoring systems are never starved of resources and **must** activate alarms when resource utilisation exceeds a configurable threshold.	
Monitoring and Security Audit	sec.mon.016	The Platform Monitoring components **should** follow security best practices for auditing, including secure logging and tracing.	
Monitoring and Security Audit	sec.mon.017	The Platform **must** audit systems for any missing security patches and take appropriate actions.	Vulnerability Assessment
Monitoring and Security Audit	sec.mon.018	The Platform, starting from initialization, **must** collect and analyze logs to identify security events, and store these events in an external system.	Patch Management
Monitoring and Security Audit	sec.mon.019	The Platform's components **must not** include an authentication credential, e.g., password, in any logs, even if encrypted.	
Monitoring and Security Audit	sec.mon.020	The Platform's logging system **must** support the storage of security audit logs for a configurable period of time.	
Monitoring and Security Audit	sec.mon.021	The Platform **must** store security events locally if the external logging system is unavailable and shall	

		periodically attempt to send these to the external logging system until successful.	
Open Source Software	sec.oss.001	Open source code **must** be inspected by tools with various capabilities for static and dynamic code analysis.	Vulnerability Assessment
Open Source Software	sec.oss.002	The CVE (Common Vulnerabilities and Exposures) [9] **must** be used to identify vulnerabilities and their severity rating for open source code part of Cloud Infrastructure and workloads software.	
Open Source Software	sec.oss.003	Critical and high severity rated vulnerabilities **must** be fixed in a timely manner. Refer to the CVSS (Common Vulnerability Scoring System) [9] to know a vulnerability score and its associated rate (low, medium, high, or critical).	
Open Source Software	sec.oss.004	A dedicated internal isolated repository separated from the production environment **must** be used to store vetted open source content.	Trusted Registry
Open Source Software	sec.oss.005	A Software Bill of Materials (SBOM [11]) **should** be provided or build, and maintained to identify the software components and their origins.	
IaaC - Secure Design and Architecture Stage Requirements	sec.arch.001	Threat Modelling methodologies and tools **should** be used during the Secure Design and Architecture stage triggered by Software Feature Design trigger. It may be done manually or using tools like open source OWASP Threat Dragon.	
IaaC - Secure Design and Architecture Stage Requirements	sec.arch.002	Security Control Baseline Assessment **should** be performed during the Secure Design and Architecture stage triggered by Software Feature Design trigger. Typically done manually by internal or independent assessors.	
IaaC - Secure Code Stage Requirements	sec.code.001	Static Application Security Testing (SAST) **must** be applied during Secure Coding stage triggered by Pull, Clone or Comment trigger. Security testing that analyses	

		application source code for software vulnerabilities and gaps against best practices. Example: open source OWASP range of tools.	
IaaC - Secure Code Stage Requirements	sec.code.002	Software Composition Analysis (SCA) **should** be applied during Secure Coding stage triggered by Pull, Clone or Comment trigger. Security testing that analyses application source code or compiled code for software components with known vulnerabilities. Example: open source OWASP range of tools.	
IaaC - Secure Code Stage Requirements	sec.code.003	Source Code Review **should** be performed continuously during Secure Coding stage. Typically done manually.	
IaaC - Secure Code Stage Requirements	sec.code.004	Integrated SAST via IDE Plugins **should** be used during Secure Coding stage triggered by Developer Code trigger. On the local machine: through the IDE or integrated test suites; triggered on completion of coding be developer.	
IaaC - Secure Code Stage Requirements	sec.code.005	SAST of Source Code Repository **should** be performed during Secure Coding stage triggered by Developer Code trigger. Continuous delivery pre-deployment: scanning prior to deployment.	
IaaC - Continuous Build, Integration and Testing Stage Requirements	sec.bld.001	Static Application Security Testing (SAST) **should** be applied during the Continuous Build, Integration and Testing stage triggered by Build and Integrate trigger. Example: open source OWASP range of tools.	
IaaC - Continuous Build, Integration and Testing Stage Requirements	sec.bld.002	SCA - Software Composition Analysis **should** be applied during the Continuous Build, Integration and Testing stage triggered by Build and Integrate trigger. Example: open source OWASP range of tools.	
IaaC - Continuous Build, Integration and Testing	sec.bld.003	Image Scan **must** be applied during the Continuous Build, Integration and Testing stage triggered by Package trigger.	

Stage Requirements		Example: A push of a container image to a container registry may trigger a vulnerability scan before the image becomes available in the registry.	
IaaC - Continuous Build, Integration and Testing Stage Requirements	sec.bld.004	Dynamic Application Security Testing (DAST) **should** be applied during the Continuous Build, Integration and Testing stage triggered by Stage & Test trigger. Security testing that analyses a running application by exercising application functionality and detecting vulnerabilities based on application behaviour and response. Example: OWASP ZAP.	
IaaC - Continuous Build, Integration and Testing Stage Requirements	sec.bld.005	Fuzzing **should** be applied during the Continuous Build, Integration and testing stage triggered by Stage & Test trigger. Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. Example: GitLab Open Sources Protocol Fuzzer Community Edition.	
IaaC - Continuous Build, Integration and Testing Stage Requirements	sec.bld.006	Interactive Application Security Testing (IAST) **should** be applied during the Continuous Build, Integration and Testing stage triggered by Stage & Test trigger. Software component deployed with an application that assesses application behaviour and detects presence of vulnerabilities on an application being exercised in realistic testing scenarios. Example: Contrast Community Edition.	
IaaC - Continuous Delivery and Deployment Stage Requirements	sec.del.001	Image Scan **must** be applied during the Continuous Delivery and Deployment stage triggered by Publish to Artifact and Image Repository trigger. Example: GitLab uses the open source Clair engine for container image scanning.	
IaaC - Continuous Delivery and Deployment	sec.del.002	Code Signing **must** be applied during the Continuous Delivery and Deployment stage triggered by Publish to Artifact and Image Repository trigger. Code Signing	

Stage Requirements		provides authentication to assure that downloaded files are from the publisher named on the certificate.	
IaaC - Continuous Delivery and Deployment Stage Requirements	sec.del.003	Artifact and Image Repository Scan **should** be continuously applied during the Continuous Delivery and Deployment stage. Example: GitLab uses the open source Clair engine for container scanning.	
IaaC - Continuous Delivery and Deployment Stage Requirements	sec.del.004	Component Vulnerability Scan **must** be applied during the Continuous Delivery and Deployment stage triggered by Instantiate Infrastructure trigger. The vulnerability scanning system is deployed on the cloud platform to detect security vulnerabilities of specified components through scanning and to provide timely security protection. Example: OWASP Zed Attack Proxy (ZAP).	
IaaC - Runtime Defence and Monitoring Requirements	sec.run.001	Component Vulnerability Monitoring **must** be continuously applied during the Runtime Defence and Monitoring stage and remediation actions **must** be applied for high severity rated vulnerabilities. Security technology that monitors components like virtual servers and assesses data, applications, and infrastructure for security risks.	
IaaC - Runtime Defence and Monitoring Requirements	sec.run.002	Runtime Application Self-Protection (RASP) **should** be continuously applied during the Runtime Defence and Monitoring stage. Security technology deployed within the target application in production for detecting, alerting, and blocking attacks.	
IaaC - Runtime Defence and Monitoring Requirements	sec.run.003	Application testing and Fuzzing **should** be continuously applied during the Runtime Defence and Monitoring stage. Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. Example: GitLab Open Sources Protocol Fuzzer Community Edition.	
IaaC - Runtime Defence and	sec.run.004	Penetration Testing **should** be continuously applied during the	

Monitoring Requirements		Runtime Defence and Monitoring stage. Typically done manually.	
Compliance With Standards	sec.std.001	The Cloud Operator **should** comply with Center for Internet Security CIS Controls [8].	
Compliance With Standards	sec.std.002	The Cloud Operator, Platform and Workloads **should** follow the guidance in the CSA Security Guidance for Critical Areas of Focus in Cloud Computing (latest version) [12].	
Compliance With Standards	sec.std.003	The Platform and Workloads **should** follow the guidance in the OWASP Cheat Sheet Series (OCSS) [13].	
Compliance With Standards	sec.std.004	The Cloud Operator, Platform and Workloads **should** ensure that their code is not vulnerable to the OWASP Top Ten Security Risks [14].	
Compliance With Standards	sec.std.005	The Cloud Operator, Platform and Workloads **should** strive to improve their maturity on the OWASP Software Maturity Model (SAMM) [15].	
Compliance With Standards	sec.std.006	The Cloud Operator, Platform and Workloads **should** utilize the OWASP Web Security Testing Guide [16].	
Compliance With Standards	sec.std.007	The Cloud Operator, and Platform **should** satisfy the requirements for Information Management Systems specified in ISO/IEC 27001 [17]. ISO/IEC 27002:2013 - ISO/IEC 27001 is the international Standard for best-practice information security management systems (ISMSs).	
Compliance With Standards	sec.std.008	The Cloud Operator, and Platform **should** implement the Code of practice for Security Controls specified in ISO/IEC 27002:2013 (or latest) [17].	
Compliance With Standards	sec.std.009	The Cloud Operator, and Platform **should** implement the ISO/IEC 27032:2012 (or latest) [19] Guidelines for Cybersecurity techniques. ISO/IEC 27032 - ISO/IEC 27032 is the international	

		Standard focusing explicitly on cybersecurity.	
Compliance With Standards	sec.std.010	The Cloud Operator **should** conform to the ISO/IEC 27035 standard for incidence management. ISO/IEC 27035 - ISO/IEC 27035 is the international Standard for incident management.	
Compliance With Standards	sec.std.011	The Cloud Operator **should** conform to the ISO/IEC 27031 standard for business continuity. ISO/IEC 27031 -ISO/IEC 27031 is the international Standard for ICT readiness for business continuity.	
Compliance With Standards	sec.std.012	The Public Cloud Operator **must**, and the Private Cloud Operator **may** be certified to be compliant with the International Standard on Awareness Engagements (ISAE) 3402 (in the US: SSAE 16). International Standard on Awareness Engagements (ISAE) 3402. US Equivalent: SSAE16.	

2.4 Kubernetes Architecture Requirements

The requirements in this section are to be delivered in addition to those in section Reference Model Requirements, and have been created to support the Principles defined in the first chapter Overview of this Reference Architecture.

The Reference Model (RM) defines the Cloud Infrastructure, which consists of the physical resources, virtualised resources and a software management system.

In virtualisation platforms, the Cloud Infrastructure consists of the guest operating system, hypervisor and, if needed, other software such as libvirt. The Cloud Infrastructure Management component is responsible for, among others, tenant management, resources management, inventory, scheduling, and access management.

With regards to containerisation platforms, the scope of the following Architecture requirements include the Cloud Infrastructure Hardware (e.g. physical resources), Cloud Infrastructure Software (e.g. Hypervisor (optional), Container Runtime, virtual or container Orchestrator(s), Operating System), and infrastructure resources consumed by virtual machines or containers.

Table 13Kubernetes Architecture Requirements

Reference	Category	Sub-category	Description	Specification Reference
gen.cnt.02	General	Cloud nativeness	The Architecture must support immutable infrastructure.	ra2.ch.017
gen.cnt.03	General	Cloud nativeness	The Architecture must run conformant	ra2.k8s.001

			Kubernetes as defined by the CNCF [20].	
gen.cnt.04	General	Cloud nativeness	The Architecture must support clearly defined abstraction layers - from the hardware infrastructure (supporting the platform) to the containerisation platform (the main concern of this Architecture) to the applications (workloads running on the platform).	
gen.cnt.05	General	Cloud nativeness	The Architecture should support configuration of all components in an automated manner using openly published API definitions.	
gen.scl.01	General	Scalability	The Architecture should support policy driven horizontal auto-scaling of workloads.	
gen.rsl.01	General	Resiliency	The Architecture must support resilient Kubernetes components that are required for the continued availability of running workloads.	ra2.k8s.004
gen.rsl.02	General	Resiliency	The Architecture should support resilient Kubernetes service components that are not subject to gen.rsl.01.	ra2.k8s.002, ra2.k8s.003
gen.avl.01	General	Availability	The Architecture must provide High Availability for Kubernetes components.	ra2.k8s.002, ra2.k8s.003, ra2.k8s.004
gen.ost.01	Openness	Availability	The Architecture should embrace open-based standards and technologies.	ra2.crt.001, ra2.crt.002, ra2.ntw.002, ra2.ntw.006, ra2.ntw.007

inf.com.01	Infrastructure	Compute	The Architecture must provide compute resources for Pods.	ra2.k8s.004
inf.stg.01	Infrastructure	Storage	The Architecture must support the ability for an operator to choose whether or not to deploy persistent storage for Pods.	ra2.stg.004
inf.ntw.01	Infrastructure	Network	The Architecture must support network resiliency on the Kubernetes nodes.	
inf.ntw.02	Infrastructure	Network	The Architecture must support redundant network connectivity to the Kubernetes nodes. At least two physical network connections are required for each physical Kubernetes node. For virtualized Kubernetes nodes, redundant network interfaces backed by redundant physical connections, are required on each virtualised Kubernetes node.	
inf.ntw.03	Infrastructure	Network	The networking solution should be able to be centrally administered and configured.	ra2.ntw.001, ra2.ntw.004
inf.ntw.04	Infrastructure	Network	The Architecture must support dual stack IPv4 and IPv6 for Kubernetes workloads.	ra2.ch.007, ra2.k8s.010
inf.ntw.05	Infrastructure	Network	The Architecture must support capabilities for integrating SDN controllers.	
inf.ntw.06	Infrastructure	Network	The Architecture must support more than one networking solution.	ra2.ntw.005, ra2.ntw.007
inf.ntw.07	Infrastructure	Network	The Architecture must support the ability for an operator to choose whether or not to	ra2.ntw.005

			deploy more than one networking solution.	
inf.ntw.08	Infrastructure	Network	The Architecture must provide a default network which implements the Kubernetes network model.	ra2.ntw.002
inf.ntw.09	Infrastructure	Network	The networking solution must not interfere with or cause interference to any interface or network it does not own.	
inf.ntw.10	Infrastructure	Network	The Architecture must support Cluster wide coordination of IP address assignment.	
inf.ntw.13	Infrastructure	Network	The platform must allow specifying multiple separate IP pools. Tenants are required to select at least one IP pool that is different from the control infrastructure IP pool or other tenant IP pools.	
inf.ntw.14	Infrastructure	Network	The platform must allow NAT-less traffic (i.e., exposing the Pod IP address directly to the outside), allowing source and destination IP addresses to be preserved in the traffic headers from workloads to external networks. This is needed e.g. for signalling applications, using SIP and Diameter protocols.	ra2.ntw.011
inf.ntw.15	Infrastructure	Network	The platform must support LoadBalancer Publishing Service (ServiceType)	
inf.ntw.16	Infrastructure	Network	The platform must support Ingress.	

inf.ntw.17	Infrastructure	Network	The platform should support NodePort Publishing Service (ServiceTypes).	
inf.ntw.18	Infrastructure	Network	The platform should support ExternalName Publishing Service (ServiceTypes).	
inf.vir.01	Infrastructure	Virtual Infrastructure	The Architecture must support the capability for containers to consume infrastructure resources abstracted by host operating systems that are running within a virtual machine.	ra2.ch.005, ra2.ch.011
inf.phy.01	Infrastructure	Physical Infrastructure	The Architecture must support the capability for containers to consume infrastructure resources abstracted by host operating systems that are running within a physical server.	ra2.ch.008
kcm.gen.01	Kubernetes Cluster	General	The Architecture must support policy driven horizontal auto- scaling of Kubernetes cluster.	N/A
kcm.gen.02	Kubernetes Cluster	General	The Architecture must enable workload resiliency.	ra2.k8s.004
kcm.gen.03	Kubernetes Cluster	General	The Architecture should enable automated TLS certificate management.	ra2.k8s.020
int.api.01	API	General	The Architecture must leverage the Kubernetes APIs to discover and declaratively manage compute (virtual and bare metal resources), network, and storage.	For Networking: ra2.ntw.001, ra2.ntw.008, ra2.app.006. Compute/storage not yet met.
int.api.02	API	General	The Architecture must support the usage of a Kubernetes application package manager	ra2.pkg.001

			using the Kubernetes API, like Helm v3.	
int.api.03	API	General	The Architecture must support stable features in its APIs.	
int.api.04	API	General	The Architecture must support limited backward compatibility in its APIs. Support for the whole API must not be dropped, but the schema or other details can change.	

3 High Level Architecture

3.1 Introduction to High Level Architecture

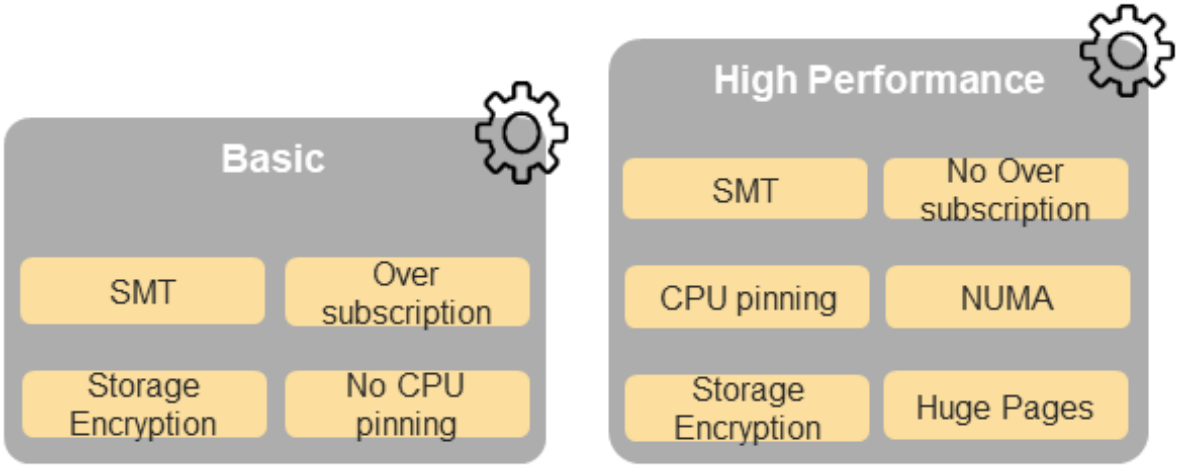
The Anuket Reference Architecture (RA2) for Kubernetes based cloud infrastructure is intended to be an industry standard and independent Kubernetes reference architecture that is not tied to any specific offering or distribution. No vendor-specific enhancements are required to achieve conformance with the Anuket specifications. Conformance to these specifications can be achieved by using upstream components or features that are developed by the open source community, and conformance is ensured by successfully running the RC2 conformance testing suite.

By using the Reference Architecture (RA2) for Kubernetes based cloud infrastructure specifications, operators can deploy infrastructure that will run any VNF or CNF that has successfully run on an RA2-conformant infrastructure. The purpose of this chapter is to outline all the components required to provide Telecom-grade Kubernetes in a consistent and reliable way. The specification of how to setup these components is detailed in the Component Level Architecture chapter.

Kubernetes is already a well-documented and widely deployed open source project of the Cloud Native Computing Foundation (CNCF). For information related to standard Kubernetes features and capabilities, refer to the standard Kubernetes documentation that can be found on the Kubernetes docs page [2]. The following chapters describe the specific features required by the Anuket Reference Architecture, and how they are expected to be implemented.

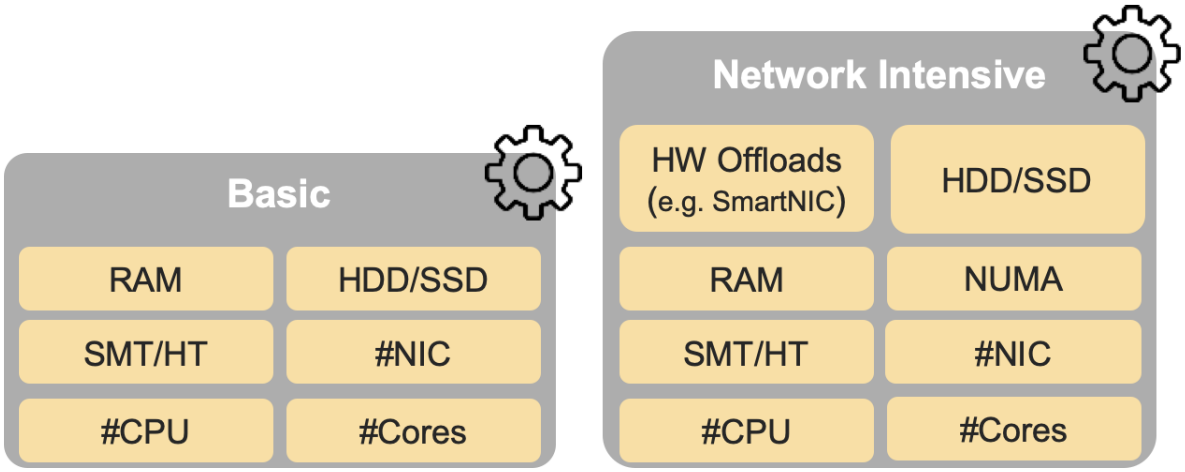
While this reference architecture provides options for modular components, such as service mesh, the focus of the Reference Architecture is on the abstracted interfaces and features that are required for Telecom workload management and execution.

Chapter 4 of the Reference Model (RM) [1] describes the hardware and software profiles that reflect the capabilities and features that the types of Cloud Infrastructure provide to the workloads. The figure below depicts a high-level view of the software features that apply to each instance profile (basic and high-performance).



(from RM): NFVI software profiles

In addition, the Figure (NFVI hardware profiles and host associated capabilities shown below) depicts the hardware profile features that apply to each instance profile.



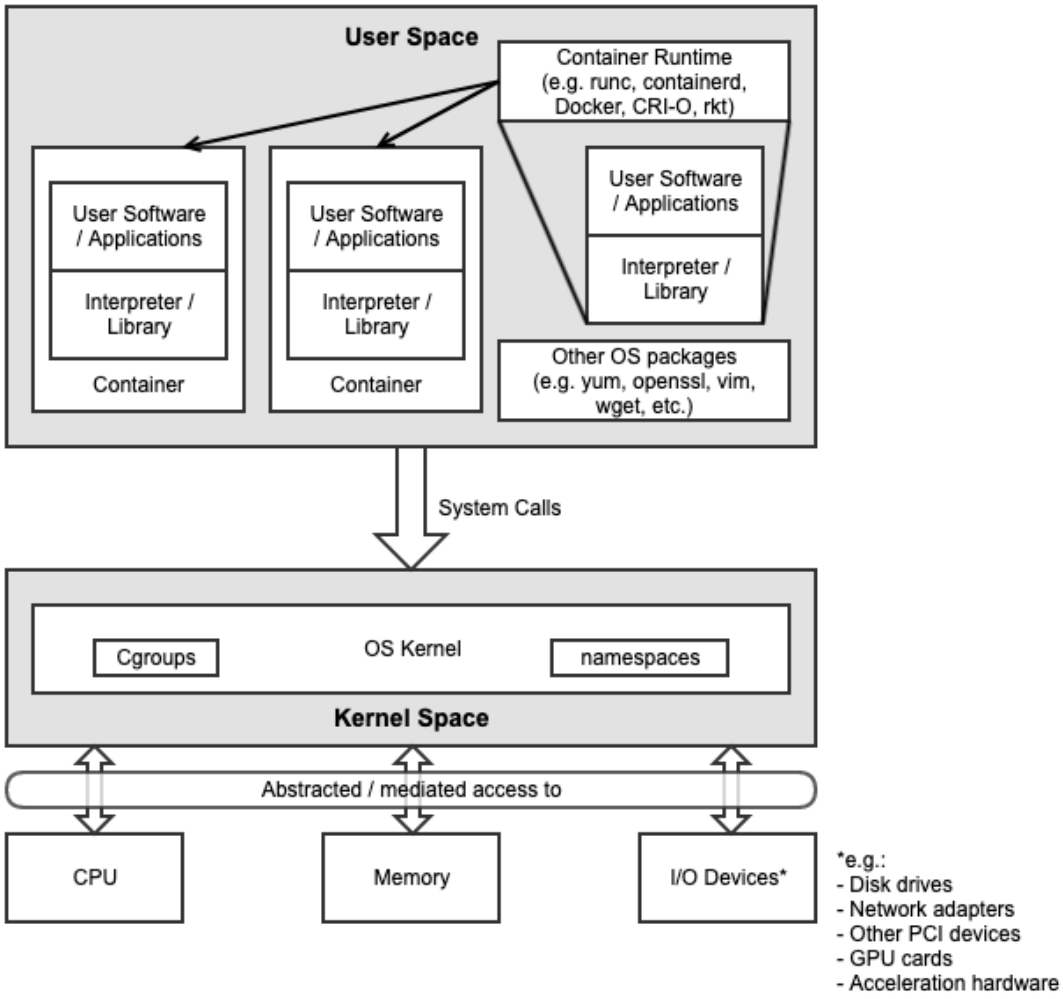
(from RM): NFVI hardware profiles and host associated capabilities

The features and capabilities described in the software and hardware profiles are considered throughout this RA, with the requirements traceability to the RM requirements formally documented in the Architecture Requirements of this RA.

3.2 Infrastructure Services

3.2.1 Container Compute Services

The primary interface between the physical/virtual infrastructure and any container-relevant components is the Kubernetes Node Operating System. This is the OS within which the container runtime exists, and within which the containers run, and therefore, the OS whose kernel is shared by the referenced containers. This is shown in the figure Kubernetes Node Operating System below.



Kubernetes Node Operating System

The Kubernetes Node OS (as with any OS) consists of two main components:

- Kernel space
- User space

The kernel is the core of the operating system, controlling all hardware resources and managing the interaction with software components. Kernel features include key containerisation capabilities, such as control groups (cgroups) and namespaces, used and managed by the container runtime to provide isolation between the user space processes. The kernel provides an API to applications running in the user space (which usually has its own southbound interface provided by an interpreter or libraries). This isolation also includes the container itself, as well as any processes running within it. The security of the Kubernetes Node OS and its relationship to the containers and the applications running within the containers, is essential to the overall security posture of the entire system. The platform must be appropriately secured to ensure that the processes running in one container cannot escalate their privileges on the node or otherwise affect processes running in an adjacent container. An example of this concept, together with further details, can be found in API and Feature Testing requirements.

It is important to note that the container runtime itself is also a set of processes that run in user space, and therefore also interact with the kernel via system calls. Many diagrams show containers as running on top of the runtime, or inside the runtime. More accurately, the containers themselves are simply processes running within an OS. The container runtime is

simply another set of processes that are used to manage these containers (pull, run, delete, and so on) and the kernel features required to provide the isolation mechanisms (cgroups, namespaces, filesystems, and so on) between the containers.

3.2.1.1 Container Runtime Services

The Container Runtime is a component that runs within a Kubernetes Node Operating System (OS). It manages the underlying OS functionality, (such as cgroups and namespaces in Linux), in order to provide an isolated context within which container images can be executed. It also makes use of the infrastructure resources, such as compute, storage, networking and other I/O devices, abstracted by the Node OS, based on API instructions from the kubelet.

There are a number of different container runtimes. The simplest form of runtimes, the low-level container runtimes, only manage the operating system capabilities, such as cgroups and namespaces, and then run commands from within those cgroups and namespaces. An example of this type of runtime is runc, which underpins many of the higher-level runtimes and is considered a reference implementation of the Open Container Initiative (OCI) runtime spec. [29] This specification includes details on how an implementation (that is, an actual container runtime such as runc) must, for example, configure resource shares and limits (such as CPU, memory, IOPS) for the containers that Kubernetes (via the kubelet) schedules on that node. This is important to ensure that the features and capabilities described in the Reference Model [1] are supported by this RA and delivered by any downstream Reference Implementations (RIs) to the instance types defined in the RM.

Where low-level runtimes are used for the execution of a container within an operating system, the more complex and complete high-level container runtimes are used for the general management of container images - moving them to where they need to be executed, unpacking them, and then passing them to the low-level runtime, which then executes the container. These high-level runtimes also include a comprehensive API that other components, such as Kubernetes, can use to interact and manage the containers. An example of this type of runtime is Containerd, which provides the features described above, and depends on runc for execution.

For Kubernetes, the important interface to consider for container management is the Kubernetes Container Runtime Interface (CRI). This is an interface specification for any container runtime to integrate with the control plane (kubelet) of a Kubernetes node. The CRI allows to decouple the kubelet from the runtime that is running in the node OS, allowing to swap container runtime if it is compliant with CRI. Examples CRI-compliant runtimes include Containerd and cri-o, which are built specifically to work with Kubernetes.

To fulfill inf.vir.01, the architecture should support a container runtime which provides the isolation of the Operating System kernels.

The basic semantics of Kubernetes, and the information found in the manifests, define the built-in Kubernetes objects and their desired state. The main objects built into Kubernetes include:

Table 14Kubernetes built-in objects

Pod and workloads	Description
Pod [30]	A pod is a collection of containers that can run on a node. This resource is created by clients and scheduled onto nodes.
ReplicaSet [31]	A ReplicaSet ensures that a specified number of pod replicas are running at any given time.

Deployment [32]	A deployment enables declarative updates for pods and ReplicaSets.
DaemonSet [33]	A DaemonSet ensures that the correct nodes run a copy of a pod.
Job [34]	A job represents a task. It creates one or more pods and ensures that the specified number of successful completions is completed.
CronJob [35]	A CronJob manages time-based jobs, namely, once or repeatedly at specified times.
StatefulSet [36]	A StatefulSet represents a set of pods with consistent identities. Identities are defined as network and storage.

3.2.1.2 CPU Management Policies

CPU management has policies to determine placement preferences to use for workloads that are sensitive to cache affinity or latency. Therefore, these workloads must not be throttled by the kubelet and their processes must not be scheduled across the CPU cores by the OS scheduler. Additionally, some workloads are sensitive to differences between the physical cores and the SMT, while others (such as DPDK-based workloads) are designed to run on isolated CPUs (such as on Linux with a cpuset-based selection of CPUs and isolcpus kernel parameters specifying cores isolated from the general SMP balancing and scheduler algorithms).

The Kubernetes CPU Manager [37] works with the Topology Manager. Special care needs to be taken of:

- Supporting isolated CPUs: Using kubelet Reserved CPUs and Linux isolcpus allows configuration where only isolcpus are allocatable to pods. Scheduling pods to such nodes can be influenced with taints, tolerations, and node affinity.
- Differentiating between physical cores and SMT: When requesting even number of CPU cores for pods, scheduling can be influenced with taints, tolerations, and node affinity.

3.2.1.3 Management of Memory and Huge Pages Resources

The Reference Model requires the support of huge pages in i.cap.018 which is supported by the upstream Kubernetes [38].

For the correct mapping of huge pages to scheduled pods, both need to have huge pages enabled in the operating system (configured in kernel and mounted with the correct permissions), as well as kubelet configuration. Multiple sizes of huge pages can be enabled, such as 2 MiB and 1 GiB.

For some applications, huge pages should be allocated to account for consideration of the underlying hardware topology. The Memory Manager [39] allows the feature guaranteed memory and huge pages allocation for pods in the Guaranteed QoS class. The Memory Manager feeds the Topology Manager with hints for the most suitable NUMA affinity.

3.2.1.4 Hardware Topology Management

NUMA nodes are defined as server system architecture divisions of CPU sockets. Scheduling pods across NUMA boundaries can result in lower performance and higher latencies. This could be an issue for applications that require optimizations of CPU isolation, memory, and device locality.

Kubernetes supports Topology policy per node [40]. The Topology Manager receives topology information from Hint Providers which identify NUMA nodes and preferred scheduling. In the case of the pod with a Guaranteed QoS class having integer CPU requests, the static CPU

Manager policy returns topology hints relating to the exclusive CPU. The Device Manager provides hints for the requested device.

If memory or huge pages are not considered by the Topology Manager, it can be done by the operating system providing best-effort local page allocation for containers, if there is sufficient free local memory on the node, or with a Control Groups (cgroups) cpuset subsystem that can isolate memory to a single NUMA node.

3.2.1.5 Node Feature Discovery

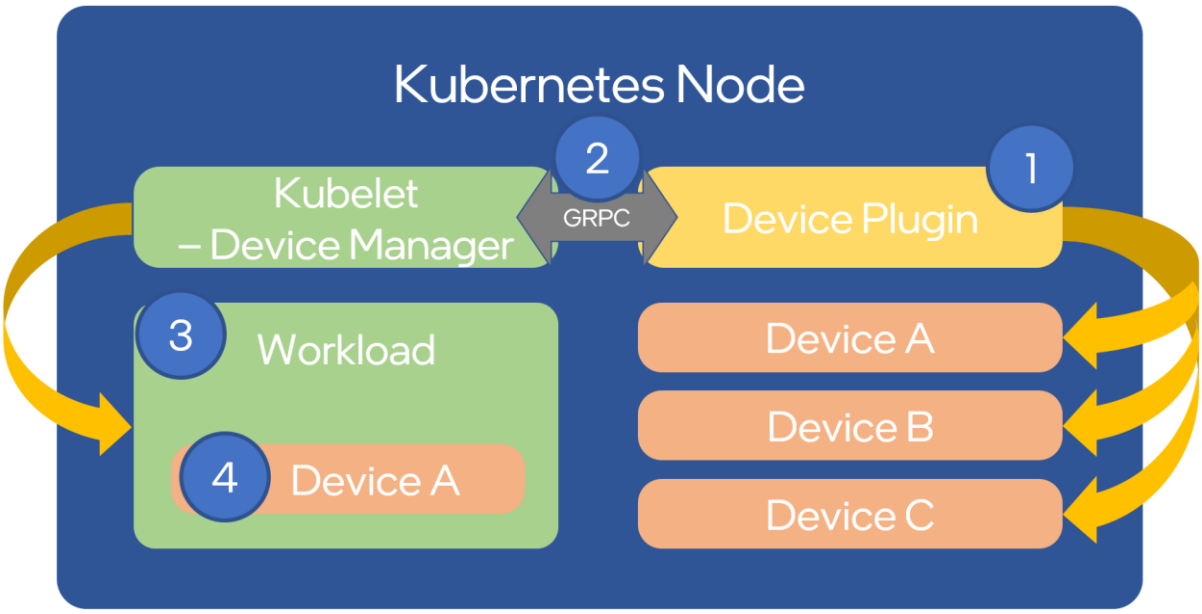
Node Feature Discovery [41] (NFD) can run on every node as a daemon or as a job. NFD detects the hardware and software capabilities of each node and then advertises those capabilities as node labels. Those node labels can be used in scheduling pods by using the Node Selector or Node Affinity for pods that require such capabilities.

3.2.1.6 Device Plugin Framework

Device Plugin Framework [42] advertises device hardware resources to kubelet, with which vendors can implement plugins for devices that may require vendor-specific activation and lifecycle management, and securely maps these devices to the containers.

The figure Device Plugin Operation below shows in four steps how device plugins operate on a Kubernetes node:

- 1: During setup, the cluster administrator (more in Operator Pattern) knows or discovers (as per Node Feature Discovery) what kind of devices are present on the different nodes, selects which devices to enable, and deploys the associated device plugins.
- 2: The plugin reports the devices it found on the node to the Kubelet device manager and starts its gRPC server to monitor the devices.
- 3: The user submits a pod specification (workload manifest file) requesting a certain type of device.
- 4: The scheduler determines a suitable node based on device availability and the local kubelet assigns a specific device to the pod's containers.



Device Plugin Operation

An example of an often-used device plugin is the SR-IOV Network Device Plugin [43]. The SR-IOV Network Device Plugin discovers and advertises SR-IOV Virtual Functions (VFs)

available on a Kubernetes node, and is used to map VFs to scheduled pods. To use it, an SR-IOV CNI is required. A CNI multiplexer plugin (such as Multus CNI [182]) is also required to provision additional secondary network interfaces for VFs (beyond the primary network interface). During pod creation, the SR-IOV CNI allocates an SR-IOV VF to a pod's network namespace using the VF information given by the meta plugin, and on pod deletion releases the VF from the pod.

With the Device Health in Pod Status feature, introduced in Kubernetes 1.31, Device Plugins can expose device health information in the allocatedResourcesStatus field of the Pod status. This enhancement allows device plugins to communicate the health of allocated devices, enabling the kubelet and cluster operators to identify and respond to device failures more effectively. Feature gate DeviceHealth must be enabled to use this capability. Telecom workloads can benefit from improved reliability by leveraging this feature to monitor the health of critical hardware accelerators and ensure continuous service availability.

3.2.1.7 Hardware Acceleration

Hardware Acceleration Abstraction in RM [1] describes types of hardware acceleration (CPU instructions, Fixed function accelerators, Firmware-programmable adapters, SmartNICs and SmartSwitches), and usage for Infrastructure Level Acceleration and Application Level Acceleration.

Scheduling pods that require, or prefer to run on, nodes with hardware accelerators depend on the type of accelerator used:

- CPU instruction sets can be found with Node Feature Discovery
- Fixed-function accelerators, Firmware-programmable network adapters, and SmartNICs can be found and mapped to pods by using Device Plugin.

3.2.1.8 Scheduling Pods with Non-resilient Applications

Non-resilient applications are sensitive to platform impairments on Compute-like pausing CPU cycles (for example, because of the OS scheduler) or Networking-like packet drops, reordering, or latencies. Such applications need to be carefully scheduled on nodes and preferably still decoupled from the infrastructure details of those nodes.

Table 15Categories of applications, requirements for scheduling pods and Kubernetes features

No.	Intensive on	Not intensive on	Using hardware acceleration	Requirements for optimised pod scheduling
1	Compute	Networking (dataplane)	No	CPU Manager
2	Compute	Networking (dataplane)	CPU instructions	CPU Manager, NFD
3	Compute	Networking (dataplane)	Fixed-function acceleration, Firmware-programmable network adapters, or SmartNICs	CPU Manager, Device Plugin
4	Networking (dataplane)		No, or Fixed function acceleration, Firmware-	Huge pages (for DPDK-based applications); CPU Manager with configuration for isolcpus

			programmable network adapters or SmartNICs	and SMT (if supported); Multiple interfaces; NUMA topology; Device Plugin
5	Networking (dataplane)		CPU instructions	Huge pages (for DPDK-based applications); CPU Manager with configuration for isolcpus and SMT (if supported); Multiple interfaces; NUMA topology; Device Plugin; NFD

3.2.1.9 Virtual Machine-based Clusters

Kubernetes clusters can implement worker nodes with "bare metal" servers (running Container Runtime on Linux host Operating Systems) or with virtual machines (VMs, running on a hypervisor).

When running in VMs, the following list of configurations shows what is needed for non-resilient applications:

- CPU Manager managing vCPUs that the hypervisor provides to the VMs.
- Huge pages enabled in the hypervisor, mapped to the VM, enabled in the guest OS, and mapped to the pod.
- Hardware Topology Management with NUMA enabled in the hypervisor, mapped into the VM, enabled in the guest OS, if needed, and mapped into the pod.
- If Node Feature Discovery and Device Plugin Framework are required, the required CPU instructions must be enabled in the VM virtual hardware. The required devices must be virtualised in the hypervisor or passed through to the Node VM, and mapped into the pods.

3.2.1.10 TLS Certificate Management

Network functions (NFs) running in Kubernetes may require PKI TLS certificates for multiple purposes. For example, 3GPP TS 33.501 describes how Inter-NF communications must be secured using mutual TLS and OAuth. cert-manager [64] can automatically provision and manage TLS certificates in Kubernetes, in order for CNFs to use them for TLS communications. It can request PKI certificates from issuers, ensure the certificates are valid and up-to-date, and can renew them before their expiry. Network Functions that are deployed on Kubernetes clusters can delegate the lifecycle management of their certificates to cert-manager.

Example lifecycle steps are listed below:

1. On start-up, the CNF requests the certificate from cert-manager. The certificate parameters are specified using the Certificate Custom Resource Definition (CRD). The CRD includes details of the required X.509 fields and values, the issuing CA to be used, the lifetime, the renewal time, and the name of the K8s Secret resource where the certificate and private key should be stored. So the CNF just provides the intent (“what” the certificate should look like, “where” it should be stored, and “when” it should be renewed). The CNF does not need to be concerned with any aspect of “how” the certificate is obtained, since this is delegated to cert-manager. The certificate request can originate from any container in the CNF Pod- either the NFc “application”, or the service mesh (e.g. where deployed as a sidecar).
2. When it receives the certificate request, cert-manager will generate a new private key, then send a Certificate Signing Request (CSR) to the relevant issuing CA. The CA returns the signed certificate. One of the benefits of cert-manager is its “pluggable” architecture.

It comes with built-in support for a number of issuing CA types and protocols, and developers can easily add support for new ones.

3. Once the certificate is returned by the relevant issuing CA, cert-manager stores the private key and certificate as a K8s Secret (specifically using the built-in “kubernetes.io/tls” Secret type). The Secret name is taken from the Certificate CRD.
4. The containers in the CNF Pods can access the K8s Secret, and use the certificate and private key. All entities belong to the same K8s namespace.
5. Renewal of the certificate before expiry is handled by cert-manager and is transparent to the CNF. Steps 2 and 3 above are repeated, and the CNF will receive the updated certificate when it next accesses the K8s Secret.

3.2.2 Container Networking Services

Kubernetes considers networking as a key component, with several distinct solutions. By default, Kubernetes networking is considered to be an “extension” to the core functionality, and is managed through the use of Network Plugins [44], which can be categorised based on the topology of the networks they manage, and the integration with the switching (such as VLAN vs tunnels) and routing (such as virtual vs physical gateways) infrastructure outside the Cluster:

- **Layer 2 underlay** plugins provide east/west ethernet connectivity between the pods, and north/south connectivity between the pods and the external networks by using the network underlay (such as VLANs on DC switches). When using the underlay for layer 2 segments, configuration is required on the DC network for every network.
- **Layer 2 overlay** plugins provide east/west pod-to-pod connectivity by creating overlay tunnels (for example, VXLAN/GENEVE tunnels) between the nodes, without requiring the creation of per-application layer 2 segments on the underlay. North-south connectivity cannot be provided.
- **Layer 3** plugins create a virtual router (for example, BPF, iptables, and kubeproxy) in each node and can route traffic between multiple layer 2 overlays via these nodes*. North-south traffic is managed by peering (with BGP, for example) virtual routers on the nodes with the DC network underlay, allowing each pod or service IP to be announced independently.

However, for more complex requirements, such as providing connectivity through acceleration hardware, there are three approaches that can be taken. The Comparison of example Kubernetes networking solutions table below shows some of the differences between the networking solutions that consist of these options. It is important to note that different networking solutions require different descriptors from the Kubernetes workloads (specifically, the deployment artefacts, such as YAML files, and so on). Therefore, the networking solution should be agreed between the CNF vendors and the CNF operators. To allow easy integration of CNFs and the platforms, it is recommended to use either the plain CNI API resources or the API resources defined in the v1.2 of the Kubernetes Network Custom Resource Definition De-facto Standard [45].

- The Default CNI Plugin through the use of a deployment-specific configuration
- A **multiplexer/meta-plugin** that integrates with the Kubernetes control plane via a Container Network Interface (CNI) and allows for the use of multiple CNI plugins, in order to provide this specific connectivity that the default Network Plugin may not be able to provide (such as Multus).

Table 16Comparison of example Kubernetes networking solutions

Requirement	Networking Solution with Multus

Additional network connections provider	Multiplexer/meta- plugin
The overlay network encapsulation protocol needs to enable ECMP in the underlay (infra.net.cfg.002)	Supported via the additional CNI plugin
NAT (infra.net.cfg.003)	Supported via the additional CNI plugin
Network Policies (Security Groups) (infra.net.cfg.004)	Supported via a CNI Network Plugin that supports Network Policies
Traffic patterns symmetry (infra.net.cfg.006)	Depends on the CNI plugin that is being used
Centrally administered and configured (inf.ntw.03)	Supported via the Kubernetes API Server
Dual stack IPv4 and IPv6 for Kubernetes workloads (inf.ntw.04)	Supported via the additional CNI plugin
Integrating SDN controllers (inf.ntw.05)	Supported via the additional CNI plugin
More than one networking solution (inf.ntw.06)	Supported
Choose whether or not to deploy more than one networking solution (inf.ntw.07)	Supported
Kubernetes network model (inf.ntw.08)	Supported via the additional CNI plugin
Do not interfere with or cause interference to any interface or network it does not own (inf.ntw.09)	Supported
Cluster-wide coordination of IP address assignment (inf.ntw.10)	Supported via IPAM CNI plugin

For hardware resources that are needed by the Kubernetes applications, Device Plugins [42] can be used to manage those resources and advertise them to the kubelet for use by the Kubernetes applications. This allows resources such as “GPUs, high-performance NICs, FPGAs, InfiniBand adapters, and other similar computing resources that may require vendor-specific initialization and setup” to be managed and consumed via standard interfaces.

The figure Kubernetes Networking Architecture below shows the main building blocks of a Kubernetes networking solution:

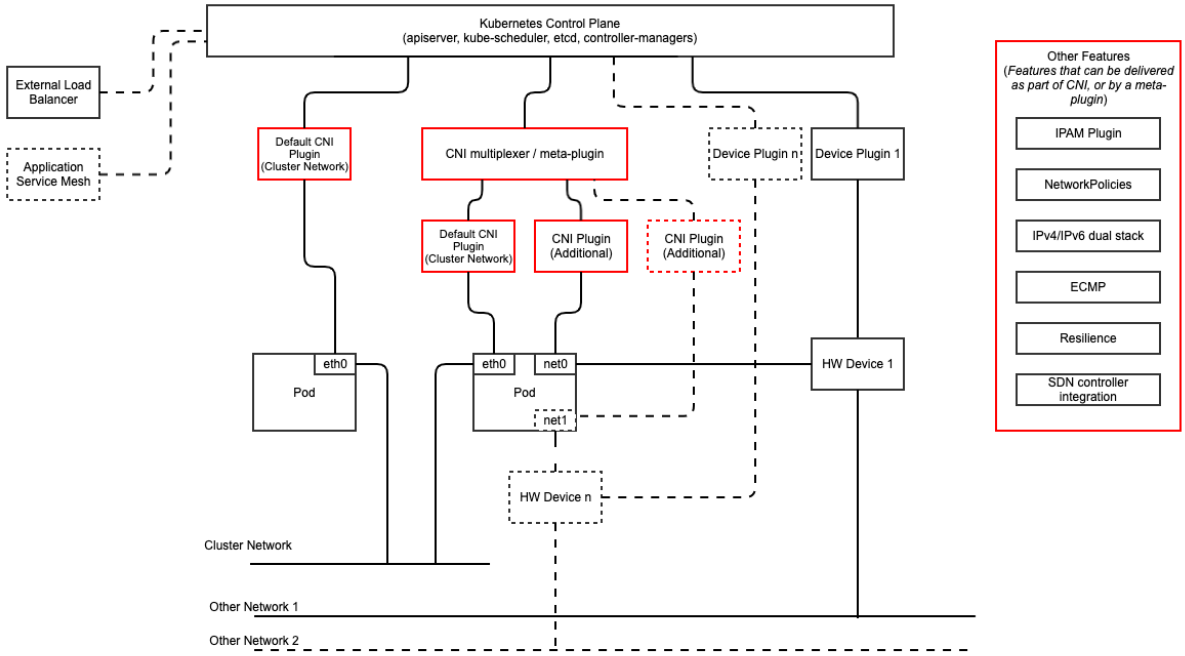
- **Kubernetes Control Plane**: this is the core of a Kubernetes Cluster: the apiserver, the etcd cluster, the kube-scheduler, and the various controller-managers. The control plane (in particular the apiserver) provides a centralised point by which the networking solution is managed using a centralised management API.
- **Default CNI Plugin (Cluster Network)**: this is the default Cluster network plugin that has been deployed within the Cluster to provide IP addresses to the pods.

  Note

  Support for IPv6 requires not only changes in the Kubernetes control plane, but also the use of a CNI Plugin that supports dual-stack networking.
- **CNI multiplexer/meta-plugin**: as described above, this is an optional component that integrates with the Kubernetes control plane via CNI, but allows for the use of multiple CNI plugins and the provision of multiple network connections for each Pod, as shown using additional CNI Plugin and net0 connection in the Pod. Note that the different network characteristics of the interfaces might require different networking technologies,

which would potentially require different CNI plugins. Also note that this is only required for the High Performance profile. An example CNI implementation that meets these requirements is the Multus [182].

- **CNI Plugin (Additional)**: this is a CNI plugin that is used to provide additional networking needs to Pods that are not provided by the default CNI plugin. This can include connectivity to underlay networks via accelerated hardware devices.
- **Device Plugin**: this is a Kubernetes extension that allows for the management and advertisement of vendor hardware devices. In particular, devices such as FPGA, SR-IOV NICs, SmartNICs, etc. can be made available to Pods by using Device Plugins. Note that alignment of these devices, CPU topology and huge pages will need the use of the Topology Manager [40].
- **External / Application Load Balancing**: As Kubernetes Ingress, Egress and Services have no support for all the protocols needed in telecommunication environments (Diameter, SIP, LDAP, etc) and their capacity is limited, the architecture includes the use of alternative load balancers, including external or ones built into the application. Management of external load balancers must be implemented via Kubernetes API objects.
- **Other Features**: these additional features that are required by the networking solution as a whole, may be delivered by the **"Default CNI Plugin"**, or the **"CNI multiplexer/meta-plugin"** if it is deployed. For example:
  - The integration of SDN solutions required by inf.ntw.05 is enabled via CNI integration.
  - IP Address Management (**IPAM**) of the various networks can be provided by one or more IPAM plugins, which can be part of a CNI plugin, or some other component (i.e. external SDN solution) - it is key that there are no overlapping IP addresses within a Cluster, and if multiple IPAM solutions are used that they are coordinated (as required by inf.ntw.10).
- **Service Mesh**: The well-known service meshes are "application service meshes" that address and interact with the application layer 7 protocols (e.g.: HTTP) only. Therefore, their support is not required, as these service meshes are outside the scope of the infrastructure layer of this architecture.



Kubernetes Networking Architecture

There are several different methods involved in managing, configuring and consuming networking resources in Kubernetes, including:

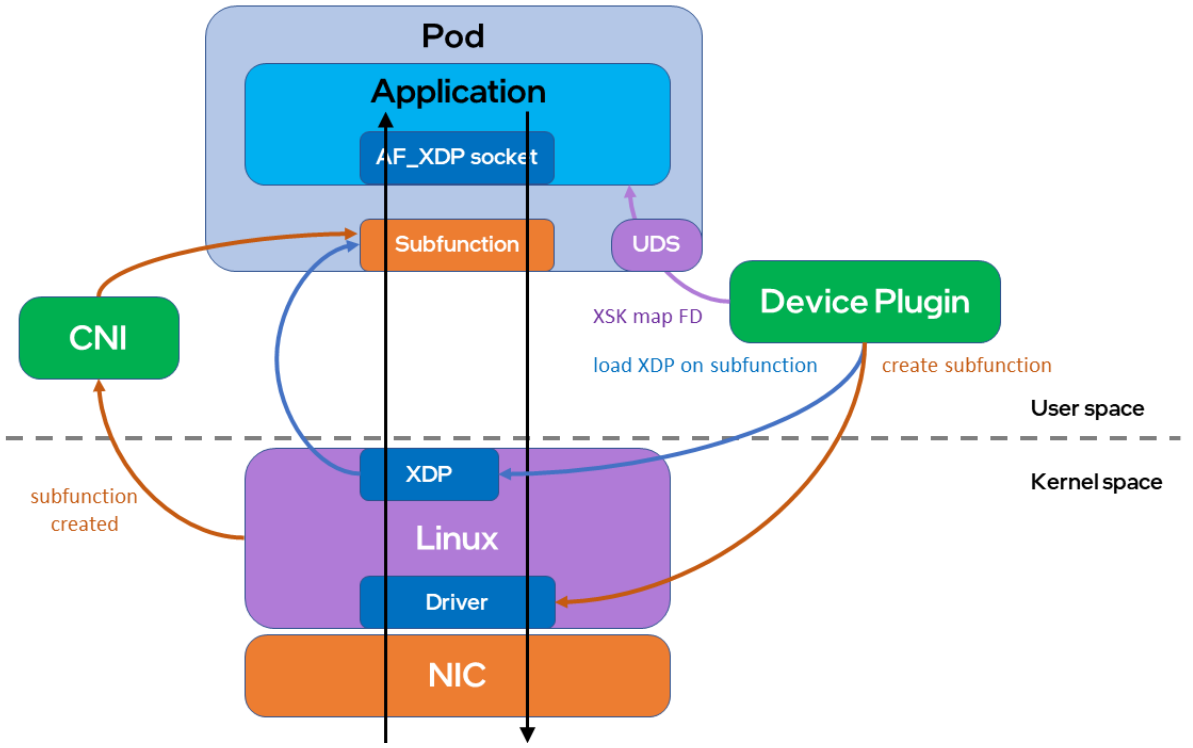
- The Default Cluster Network can be installed and managed by config files, Kubernetes API Server (e.g., Custom Resource Definitions) or a combination of the two.
- Additional networking management plane (e.g., CNI multiplexer/meta-plugin or federated networking manager) can be installed and managed by config files, Kubernetes API Server (e.g., Custom Resource Definitions) or a combination of the two.
- The connecting of Pods to the Default Cluster Network is handled by the Default CNI Plugin (Cluster Network).
- The connecting of Pods to the additional networks is handled by the additional networking management plane through the Kubernetes API (e.g., Custom Resource Definitions, Device Plugin API).
- Configuration of these additional network connections to Pods (i.e., provision of an IP address to a Pod) can either be managed through the Kubernetes API (e.g. Custom Resource Definitions) or an external management plane (e.g., dynamic address assignment from a VPN server).

There are several types of low latency and high throughput networks required by Telecom workloads: for example signalling traffic workloads and user plane traffic workloads. Networks used for signalling traffic are more demanding than what a standard overlay network can handle, but still do not need the use of user space networking. Due to the nature of the signalling protocols used, these type of networks require NAT-less communication documented in infra.net.cfg.003 and will need to be served by a CNI plugin with IPVLAN or MACVLAN support. On the other hand, the low latency, high throughput networks used for handling the user plane traffic, require the capability to use an accelerated user space networking technology.

Note: An infrastructure can provide the possibility to use SR-IOV with DPDK as an additional feature and still be conformant with Anuket.

3.2.2.1 AF_XDP Architecture

AF_XDP (Address Family for eXpress Data Path) is an address family optimized for high-performance packet processing applications, described in Reference Model Chapter 3 section Address Family For XDP (AF_XDP) [1].



The [61] running with higher privileges, is loading the eBPF program. The AF_XDP Device Plugin provides the netdev name which the AF_XDP CNI moves into the pod namespace. That way the AF_XDP Socket file descriptor is provided (via UNIX domain socket) to the application container after the pod has started. The CNI provides support to set appropriate ethtool filters, and does not rename the netdev which remains in the UP state.

Such applications can have pods with multiple network interfaces of which one is AF_XDP, and on that interface run user space libraries for AF_XDP from project [62], or directly receive and send raw packets over AF_XDP. DPDK applications can use the [63].

3.2.3 Kubernetes Networking Semantics

Support for advanced network configuration management does not exist in core Kubernetes. Kubernetes is missing the advanced networking configuration component of Infrastructure as a Service (IaaS). For example, there is no network configuration API and there is no way to create L2 networks and instantiate network services such as L3aaS and LBaaS, and then connect them all together.

Kubernetes networking can be divided into two parts: built-in network functionality, available through the pod's mandatory primary interface, and network functionality, available through the pod's optional secondary interfaces.

3.2.3.1 Built-in Kubernetes Network Functionality

Vanilla Kubernetes only allows for one network, the *cluster* network, and one network attachment for each pod. All pods and containers have a primary interface, which is created by Kubernetes during pod creation and attached to the cluster network. All communication to and from the pod is done through this interface. To only allow for one interface in a pod

removes the need for traditional networking tools such as *VRFs* and additional routes and routing tables inside the pod network namespace.

The basic semantics of Kubernetes, and the information found in the manifests, defines the connectivity rules and behaviour without any references to IP addresses. This has many advantages: it makes it easy to create portable, scalable software services and network policies for them that are not location-aware and therefore can be executed more or less anywhere.

Table 17Kubernetes networking built-in objects

Network objects	Description
Ingress [46]	Ingress is a collection of rules that allow inbound connections to reach the endpoints defined by a backend. An Ingress can be configured to give services URLs that are reachable externally, load balance traffic, terminate SSL, offer name-based virtual hosting, and so on.
Service [47]	A service is a named abstraction of an application that runs on a set of pods. The application consists of a local port (for example, 3306) on which the proxy listens, and a selector that determines which pods answer requests sent through the proxy.
EndpointSlices [48]	Endpoints and EndpointSlices are a collection of objects that contain the IP addresses, (IPv4 and IPv6) of the pods that represent a service.
Network Policies [49]	A Network Policy defines which network traffic is allowed to ingress and egress from a set of pods.

There is no need to explicitly define internal load balancers, server pools, service monitors, firewalls, and so on. The Kubernetes semantics and the relationship between the different objects defined in the object manifests contains all the necessary information.

Example: The manifests for the my-service service and the deployment with the four load balanced pods of the *my-app* type.

Service:

```
apiVersion: v1
kind: Service
metadata:
    name: my-service
    spec:
        selector:
            app: my-app
        ports:
            - protocol: TCP
                port: 123
```

Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata: name: my-app-deployment
spec:
    selector:
        matchLabels:
            app: my-app
            replicas: 4
            template:
```

```
        metadata:
            labels:
                app: my-app
    spec:
        containers:
            - name: my-app
              image: my-app-1.2.3
              ports:
              - containerPort: 123
```

This is all that is required to deploy four pods or containers that are fronted by a service that performs load balancing. The *Deployment* ensures that there are always four pods of the *my-app* type available. The *Deployment* is responsible for the full lifecycle management of the pods. This includes in-service updates and upgrades.

However, when implementing network service functions, such as VNFs/CNFs, that operate on multiple networks and require advanced networking configurations, additional capabilities are required.

3.2.3.2 Multiple Networks and Advanced Configurations

Kubernetes currently does not in itself support multiple networks, multiple-pod network attachments, or advanced network configurations. These are supported by using a Container Network Interface [50] multiplexer such as Multus [182]. A considerable effort is being invested to add better network support to Kubernetes. All such activities are coordinated through the Kubernetes Network Special Interest Group [51] and its sub-groups. One such group, the Network Plumbing Working Group [52] has produced the Kubernetes Network Custom Resource Definition De-facto Standard [45]. This document describes how secondary networks can be defined and attached to pods.

This de-facto standard defines, among other things, the following concepts:

Table 18Kubernetes multiple network concepts

Definition	Description
Kubernetes Cluster-Wide default network	This is a network to which all pods are attached according to the current behavior and requirements of Kubernetes. This is done by attaching the eth0 interface to the pod namespace.
Network Attachment	Network Attachment is a means of allowing a pod to communicate directly with a given logical or physical network. Typically (but not necessarily), each attachment takes the form of a kernel network interface placed into the pod's network namespace. Each attachment may result in zero or multiple IP addresses being assigned to the pod.
NetworkAttachmentDefinition object	The NetworkAttachmentDefinition object defines the resource object that describes how to attach a pod to a logical or physical network. The annotation name is "k8s.v1.cni.cncf.io/networks".
Network Attachment Selection Annotation	Network Attachment Selection Annotation selects one or more networks to which a pod must be attached.

Example: Define three network attachments and attach the three networks to a pod.

Green network

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name:green-network
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "plugin-A",
    "vlan": "1234"
  }'
```

Blue network

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name:blue-network
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "plugin-A",
    "vlan": "3456"
  }'
```

Red network

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name:red-network
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "plugin-B",
    "knid": "123456789"
  }'
```

Pod my-pod

```
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: blue-network, green-network, red-network
```

This is enough to support basic network configuration management. It is possible to map L2 networks from an external network infrastructure into a Kubernetes system and attach pods to these networks. Support for IPv4 and IPv6 address management is, however, limited. The address must be assigned by the CNI plugin as part of the pod creation process.

3.2.4 Container Storage Services

Kubernetes supports the Container Storage Interface (CSI) as the stable solution for storage plugins (as of Kubernetes 1.31, in-tree plugins have been removed, and CSI drivers must be used instead).

Running containers require ephemeral storage on which to run themselves (that is, storage on which the unpacked container image is stored and from which it is executed). This

ephemeral storage lives and dies with the container and is a directory on the worker node on which the container is running.

Note

This means that the ephemeral storage is mounted locally in the worker node filesystem. The filesystem can be physically external to the worker node (for example, iSCSI, NFS, and FC), but the container still references it as part of the local filesystem.

Additional storage could also be attached to a container using Kubernetes Volumes. This can be storage from the worker node filesystem (through hostPaths, although this is not recommended), or it can be external storage that is accessed using a Volume Plugin. Volume Plugins allow the use of a storage protocol (such as iSCSI and NFS) or a management API (such as Cinder and EBS) for attaching and mounting the storage into a pod. This additional storage that is attached to a container using a Kubernetes Volume does not live and die with the container, but instead follows the lifecycle of the pod of which the container is a part. This means the Volume persists across container restarts, if the pod itself is still running. However, it does not necessarily persist when a pod is destroyed. Therefore, it cannot be considered suitable for any scenario requiring persistent data. The lifecycle of the actual data depends on the Volume Plugin used and sometimes also the configuration of the Volume Plugin.

For those scenarios where data persistence is required, Persistent Volumes (PVs) are used in Kubernetes. PVs are resources in a Kubernetes Cluster that are consumed by Persistent Volume Claims (PVCs) and have a lifecycle that is independent of any pod that uses a PV. A pod uses a PVC as the volume in the pod spec. A PVC is a request for persistent storage (a PV) by a pod. By default, PVs and PVCs are manually created and deleted.

Kubernetes also provides Storage Classes. Storage Classes are created by cluster administrators and maps to storage attributes such as quality-of-service, encryption, data resilience, and so on. Storage Classes also enable the dynamic provisioning of Persistent Volumes (as opposed to the default manual creation). This can be beneficial for organizations where the administration of storage is performed separately from the administration of Kubernetes-based workloads.

Kubernetes does not place any restrictions on the storage that can be consumed by a workload, in terms of the requirements that are defined in the RM sections Storage Configurations (hardware) and Virtual Storage (software). The only difference is that Kubernetes does not have a native object storage offering. Addressing this capability gap directly is outside of the scope of this RA.

3.2.5 Kubernetes Application package manager

To manage the lifecycle (for example, install and configure, upgrade, and uninstall) of complex applications consisting of several pods and other Kubernetes objects, the Reference Architecture mandates the use of a specific Kubernetes Application package manager. The package manager must be able to manage the lifecycle of an application and provide a framework to customize a set of parameters for its deployment. The requirement for the clusters is to expose a Kubernetes API for the package managers to use in the lifecycle management of the applications they manage. This must comply with the CNCF CNF Conformance Test. As it is not recommended to use a Kubernetes Application package manager with a server side component installed in the Kubernetes cluster (for example, Tiller), Helm v3 [53] is the chosen Kubernetes Application package manager.

3.2.6 Custom Resources

Custom resources [54] are extensions of the Kubernetes API that represent customizations of the Kubernetes installation. Core Kubernetes functions are also built using custom resources. This makes Kubernetes more modular. Two ways to add custom resources are the following:

- Custom Resource Definitions [55] (CRDs): Defining a CRD object creates new custom resource with a name and schema that are easy to use.
- API Server Aggregation [56]: This is an additional API that, in a flexible way, extends Kubernetes beyond the core Kubernetes API.

3.2.6.1 Operator Pattern

A custom controller [57] is a control loop that watches a custom resource for changes and tries to keep the current state of the resource in sync with the desired state.

The Operator pattern [58] combines custom resources and custom controllers. Operators are software extensions to Kubernetes that capture operational knowledge and automate usage of custom resources to manage applications, their components, and cloud infrastructure. Operators can have different capability levels. According to the OperatorHub.io [59] repository, an operator can have the following different capability levels:

- Basic install: Automated application provisioning and configuration management.
- Seamless upgrades: Patch and minor version upgrades are supported.
- Full lifecycle: Application lifecycle and storage lifecycle (backup and failure recovery).
- Deep insights: Metrics, alerts, log processing, and workload analysis.
- Auto pilot: Horizontal/vertical scaling, automated configuration tuning, abnormality detection, and scheduling tuning.

3.3 Platform services

The archirecture may support functionalities on top of the infrastructure services. With Kubernetes and the cloud native paradigms the demarcation between infrastructure services and platform services are fuzzy. In this specification we considider every service what is provided by Kubernetes with the help of supporting components as infrastructure service, while all services provided by independent open source components are considered as platform services. Based on the specification of Chapter 5.1.5 of the Reference Model (RM) [1] the following platform services may be supported by an architecture:

- Data stores/databases
- Streaming and messaging
- Load balancer and service proxy
- Service mesh
- Security and compliance
- Monitoring
- Logging
- Application definition and image build
- CI/CD
- Ingress/egress controllers
- Network related services
- Coordination and service discovery
- Automation and configuration
- Secrets Store
- Tracing

3.4 CaaS Manager - Cluster Lifecycle Management

Note

detailed requirements and the component specification of cluster LCM are out of scope for this release.

To provision multiple Kubernetes Clusters, which is a common scenario where workloads and network functions require dedicated, single-tenant clusters, the Reference Architecture provides support for a **CaaS Manager**, a component responsible for the lifecycle management of multiple Kubernetes clusters. This component is responsible for delivering an end-to-end lifecycle management (creation and installation, scaling, updating, deleting, and so on, of entire clusters), visibility and control of CaaS clusters, together with verification of security and compliance of Kubernetes clusters across multiple data centres and clouds. Specifically, the scope of the CaaS Manager comprises the following:

- Infrastructure (Kubernetes Clusters) provisioning. This comprises either of the following:
  - LCM of control/worker VM nodes - via IaaS API.
  - Bare metal provisioning for physical nodes.
- Control plane installation (that is, Kubernetes control plane components on the nodes).
- Node OS customization (for example, Kernel customization).
- Management of Cluster add-ons (for example, CNIs, CSIs, and Service Meshes).

The CaaS Manager maintains a catalogue of **cluster templates**. These templates are used to create clusters specific to the requirements of workloads, the underlying virtualization provider, and/or the specific server hardware to be used for the cluster.

The CaaS manager works by integrating with an underlying virtualization provider for VM-based clusters, or with bare metal management APIs for physical clusters, to create cluster nodes and provide other capabilities, such as node scaling (for example, provisioning a new node and attaching it to a cluster).

A CaaS Manager leverages the closed-loop desired state configuration management concept that Kubernetes itself enables. This means that the CaaS Manager takes the desired state of a CaaS cluster as input and the controller must be able to maintain that desired state through a series of closed loops.

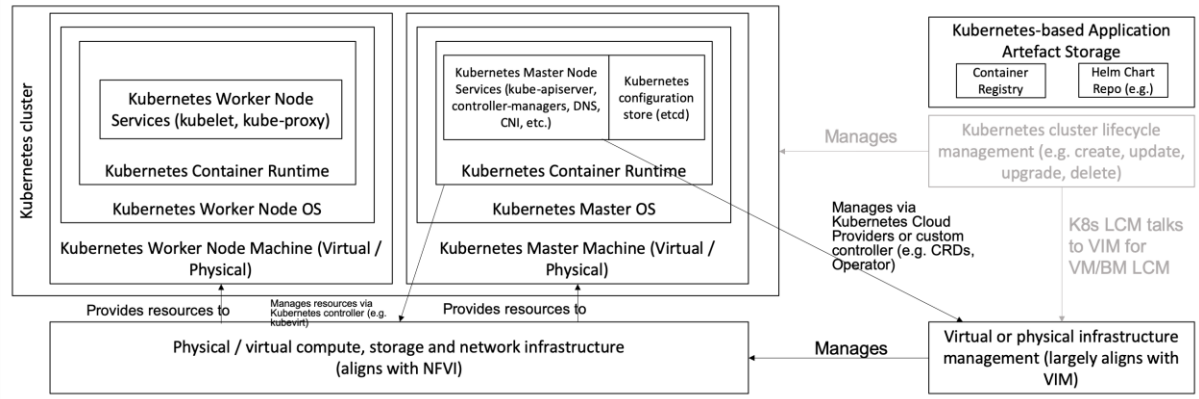
4 Component Level Architecture

4.1 Introduction to Component Level Architecture

This section describes in detail the Reference Architecture (RA2) for Kubernetes-based cloud infrastructure in terms of the functional capabilities and how they relate to the Reference Model (RM) requirements [1], that is, how the infrastructure profiles are determined, documented, and delivered.

The specifications defined in this section will be detailed with unique identifiers, which will follow the pattern: ra2.<section>.<index>, for example, ra2.ch.001 for the first requirement in the Kubernetes Node section. These specifications will then be used as requirements input for the Reference Implementation based on RA2 specifications (RI2) and any vendor or community implementations.

The Kubernetes Reference Architecture figure below shows the architectural components that are described in the subsequent sections of this chapter.



Kubernetes Reference Architecture

4.2 Kubernetes Node

This section describes the configuration that will be applied to the physical or virtual machine and its Operating System. For a Kubernetes Node to be conformant with the Reference Architecture, it must be implemented according to the following specifications:

Table 19Node Specifications

Ref	Specification	Details	Requirement Trace	Reference Implementation Trace
ra2.ch.001	Huge pages	For the node's profile to qualify as high-performance, it must be possible to enable Huge pages (2048KiB and 1048576KiB) within the Kubernetes	infra.com.cfg.004 Reference Model for Cloud Infrastructure (RM) [1] Chapter 5, section Virtual Compute	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 3, section Introduction

		Node OS, exposing schedulable resources hugepages-2Mi and hugepages-1Gi.		
ra2.ch.002	SR-IOV capable NICs	For the node's profile to qualify as high-performance, the physical machines on which the Kubernetes Nodes run must be equipped with NICs that are SR-IOV-capable.	e.cap.013 Reference Model for Cloud Infrastructure (RM) [1] Chapter 4, section Exposed Performance Optimisation Capabilities	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 3, section Infrastructure Requirements
ra2.ch.003	SR-IOV Virtual Functions	For the node's profile to qualify as high-performance, SR-IOV virtual functions (VFs) must be configured within the Kubernetes Node OS, as the SR-IOV Device Plugin does not manage the creation of these VFs.	e.cap.013 Reference Model for Cloud Infrastructure (RM) [1] Chapter 4, section Exposed Performance Optimisation Capabilities	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ch.004	CPU Simultaneous Multi-Threading (SMT)	If SMT is supported, then it must be enabled in the BIOS on the physical machine on which the Kubernetes Node runs.	infra.hw.cpu.cfg.004 Reference Model for Cloud Infrastructure (RM) [1] Chapter 5, section Compute Resources	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 3, section Infrastructure Requirements
ra2.ch.005	CPU Allocation Ratio - VMs	For Kubernetes nodes running as Virtual Machines, the CPU allocation ratio between		

		the vCPU and the physical CPU core must be 1:1.		
ra2.ch.006	CPU Allocation Ratio - Pods	To ensure the CPU allocation ratio between the vCPU and the physical CPU core is 1:1, the sum of the CPU requests and limits by the containers in the Pod specifications must remain less than the allocatable quantity of CPU resources (that is, requests.cpu < allocatable.cpu and limits.cpu < allocatable.cpu).	infra.com.cfg.001 Reference Model for Cloud Infrastructure (RM) [1] Chapter 5, section Virtual Compute Profiles	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 3, section Infrastructure Requirements
ra2.ch.007	IPv6DualStack	To support IPv4/IPv6 dual-stack networking, the Kubernetes Node OS must support and be allocated routable IPv4 and IPv6 addresses.		
ra2.ch.008	Physical CPU Quantity	The physical machines on which the Kubernetes nodes run must be equipped with at least two (2) physical sockets, each with at least 20 CPU cores.	infra.hw.cpu.cfg.001 and infra.hw.cpu.cfg.002 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 8, section Telco Edge Cloud: Infrastructure Profiles	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 3, section Infrastructure Requirements
ra2.ch.009	Physical Storage	The physical machines on which the Kubernetes nodes run	infra.hw.stg.ssd.cfg.002 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 5,	Reference Implementation based on RA2 specifications (RI2) [66]

		should be equipped with solid-state drives (SSDs).	section Storage Configurations	Chapter 3, section Infrastructure Requirements
ra2.ch.010	Local Filesystem Storage Quantity	The Kubernetes nodes must be equipped with local filesystem capacity of at least 320 GB for unpacking and executing containers. Note Extra filesystem storage should be provisioned to cater for any overheads required by the Operating System and any required OS processes, such as the container runtime, Kubernetes agents, and so on.	e.cap.003 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 4, section Exposed Resource Capabilities	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 3, section Infrastructure Requirements
ra2.ch.011	Virtual Node CPU Quantity	If using VMs, the Kubernetes nodes must be equipped with at least 16 vCPUs. Note Extra CPU capacity should be provisioned to cater for any overheads required by the Operating System and any required OS processes, such as the container runtime, Kubernetes agents, and so on.	• e.cap.001 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 4, section Exposed Resource Capabilities	

ra2.ch.012	Kubernetes Node RAM Quantity	The Kubernetes nodes must be equipped with at least 32 GB of RAM. Note Extra RAM capacity should be provisioned to cater for any overheads required by the Operating System and any required OS processes, such as the container runtime, Kubernetes agents, and so on.	e.cap.002 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 4, section Exposed Resource Capabilities	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 3, section Infrastructure Requirements
ra2.ch.013	Physical NIC Quantity	The physical machines on which the Kubernetes nodes run must be equipped with at least four (4) Network Interface Card (NIC) ports.	infra.hw.nic.cfg.001 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 5, section NIC configurations	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 3, section Infrastructure Requirements
ra2.ch.014	Physical NIC Speed - Basic Profile	The speed of the NIC ports housed in the physical machines on which the Kubernetes Nodes run for workloads matching the Basic Profile must be at least 10 Gbps.	infra.hw.nic.cfg.001 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 5, section NIC configurations	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 3, section Infrastructure Requirements
ra2.ch.015	Physical NIC Speed - High Performance Profile	The speed of the NIC ports housed in the physical machines on which the Kubernetes nodes run for	infra.hw.nic.cfg.001 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 5, section NIC configurations	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 3, section

		workloads matching the high-performance profile must be at least 25 Gbps.		Infrastructure Requirements
ra2.ch.016	Physical PCIe slots	The physical machines on which the Kubernetes nodes run must be equipped with at least eight (8) Gen3.0 PCIe slots, each with at least eight (8) lanes.		
ra2.ch.017	Immutable infrastructure	Whether physical or virtual machines are used, the Kubernetes node must not be changed after it is instantiated. New changes to the Kubernetes node must be implemented as new node instances. This covers any changes from the BIOS, through the Operating System, to running processes and all associated configurations.	gen.cnt.02 from Reference Architecture (RA1) for OpenStack based cloud infrastructure [67] Chapter 2, section General Recommendations	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ch.018	NFD	Node Feature Discovery [68] must be used to advertise the detailed software and hardware capabilities of each node in	tbd	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure

		the Kubernetes Cluster.		
ra2.ch.019	AF_XDP Zero Copy capable netdevs	AF_XDP Zero Copy capable netdevs (dependent on AF_XDP Zero Copy NIC driver) must be available in a compliant Kubernetes worker node if optional AF_XDP is used.	e.cap.025 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 4, section Exposed infrastructure capabilities	
ra2.ch.020	Real-Time	For Kubernetes nodes belonging to the rt-tsn (ref. Reference Model for Cloud Infrastructure (RM) [1] Chapter 2) flavour, Real-Time versions and/or configurations in BIOS, kernel and OS services	e.cap.026 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 4, section Exposed infrastructure capabilities	

4.3 Node Operating System

For a Host OS to be compliant with this Reference Architecture, it must meet the following requirements:

Table 20Operating System requirements

Ref	Specification	Details	Requirement Trace	Reference Implementation Trace
ra2.os.001	Linux Distribution	A deb-/rpm-compatible distribution of Linux. It must be used for the control plane nodes. It can also be used for worker nodes.	tbd	tbd
ra2.os.002	Linux kernel version	A version of the Linux kernel that is compatible with container runtimes and kubeadm - this has been chosen as the baseline because kubeadm is focused on installing and	tbd	tbd

		managing the lifecycle of Kubernetes and nothing else, hence it is easily integrated into higher-level tooling for the full lifecycle management of the infrastructure, cluster add-ons, and so on.		
ra2.os.003	Windows server	The Windows server can be used for worker nodes, but beware of the limitations.	tbd	tbd
ra2.os.004	Disposable OS	In order to support gen.cnt.02 in Kubernetes Architecture Requirements (immutable infrastructure), the Host OS must be disposable, meaning the configuration of the Host OS (and associated infrastructure such as VM or bare metal server) must be consistent - e.g. the system software and configuration of that software must be identical apart from those areas of configuration that must be different such as IP addresses and hostnames.	tbd	tbd
ra2.os.005	Automated deployment	This approach to configuration management supports lcm.gen.01 (automated deployments).	tbd	tbd

Table 4.3 lists the kernel versions that comply with this Reference Architecture specification.

Table 21Operating System versions

OS Family	Kernel Version(s)	Notes
Linux	4.x	The overlay filesystem snapshotter, used by default by containerd, uses features that were finalized in the 4.x kernel series.
Linux	>= 4.18	If using optional AF_XDP (see ra2.ch.019).
Linux	rt/realtime	If using optional Real-Time (see ra2.ch.020).
Windows	1809 (10.0.17763)	For worker nodes only.

4.4 Kubernetes

For the Kubernetes components to be conformant with the Reference Architecture they must be implemented according to the following specifications:

Table 22Kubernetes Specifications

Ref	Specification	Details	Requirement Trace	Reference Implementation Trace
ra2.k8s.001	Kubernetes conformance	The Kubernetes distribution, product, or installer used in the implementation must be listed in the Kubernetes Distributions and Platforms document [69] and marked (X) as conformant for the Kubernetes version defined in Required Component Versions.	gen.cnt.03 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.k8s.002	Highly available etcd	An implementation must consist of either three, five or seven nodes running the etcd service (can be colocated on the control plane nodes, or can run on separate nodes, but not on worker nodes).	gen.rsl.02 in Kubernetes Architecture Requirements, gen.avl.01 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.k8s.003	Highly available control plane	An implementation must consist of at least one control plane node per availability zone or fault domain to ensure the high availability and resilience of the Kubernetes control plane services.		
ra2.k8s.012	Control plane services	A control plane node must run at least the following Kubernetes control plane services: kube-apiserver, kube-scheduler and kube-controller-manager.	gen.rsl.02 in Kubernetes Architecture Requirements, gen.avl.01 in Kubernetes Architecture	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section

			Requirements	Installation on Bare Metal Infrastructure
ra2.k8s.004	Highly available worker nodes	An implementation must consist of at least one worker node per availability zone or fault domain to ensure the high availability and resilience of workloads managed by Kubernetes	en.rsl.01 in Kubernetes Architecture Requirements, gen.avl.01 in Kubernetes Architecture Requirements, kcm.gen.02 in Kubernetes Architecture Requirements,	
ra2.k8s.005	Kubernetes API Version	An implementation must use a Kubernetes version as per the subcomponent versions table in Required Component Versions. In alignment with the Kubernetes Supported versions [70], the difference between the kubernetes release of the control plane nodes and the kubernetes release of the worker nodes must be at most **3** releases (i.e. a n-3 skew).		
ra2.k8s.006	NUMA support	When hosting workloads matching the high-performance profile, the TopologyManager and CPUManager feature gates must be enabled and configured in the kubelet. --feature-gates="…, TopologyManager=true,CPUManager=true" --topology-manager-policy=single-numa-node --cpu-manager-policy=static Note The TopologyManager feature is enabled by default in Kubernetes v1.18 and later, and the CPUManager feature is enabled by default in Kubernetes v1.10 and later.	e.cap.007 in Cloud Infrastructure Software Profile Capabilities, infra.com.cfg.002 in Reference Model for Cloud Infrastructure (RM) [1], e.cap.013 Reference Model for Cloud Infrastructure (RM) [1] Chapter 8,	

			section Exposed Performance Optimisation Capabilities	
ra2.k8s.007	DevicePlugins feature gate	When hosting workloads matching the high-performance profile, the DevicePlugins feature gate must be enabled. Additionally, to utilize device health reporting, the DeviceHealth feature gate should be enabled. --feature-gates="…,DevicePlugins=true,DeviceHealth=true,…" Note The DevicePlugins feature is enabled by default in Kubernetes v1.10 or later. Device plugins can report device health status directly in the Pod's allocatedResources field.	Various, e.g. e.cap.013 in Reference Model for Cloud Infrastructure (RM) [1] Chapter 8, section Exposed Performance Optimisation Capabilities	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.k8s.008	System resource reservations	To avoid resource starvation issues on the nodes, the implementation of the architecture must reserve compute resources for system daemons and Kubernetes system daemons such as kubelet, container runtime, and so on. Use the following kubelet flags: --reserved-cpus=[a-z], using two of a-z to reserve 2 SMT threads.	i.cap.014 in Cloud Infrastructure Software Profile Capabilities	
ra2.k8s.009	CPU pinning	When hosting workloads matching the high-performance profile, in order to support CPU pinning, the kubelet must be started with the --cpu-manager-policy=static option. Note Only containers in Guaranteed pods - where CPU resource requests and limits are identical - and configured with positive-integer CPU requests will take advantage of this. All other pods will run on CPUs in the remaining shared pool.	infra.com.cfg.003 in Reference Model for Cloud Infrastructure (RM) [1] Chapter 5, section	
ra2.k8s.010	IPv6DualStack	To support IPv6 and IPv4, the IPv6DualStack feature gate must be enabled on various components (requires Kubernetes v1.16 or later). kube-apiserver: --feature-gates="IPv6DualStack=true". kube-	inf.ntw.04 in Kubernetes Architecture Requirements	

		controller-manager: --feature-gates="IPv6DualStack=true" --cluster-cidr=<IPv4 CIDR>,<IPv6 CIDR> --service-cluster-ip-range=<IPv4 CIDR>, <IPv6 CIDR> --node-cidr-mask-size-ipv4 ¦ --node-cidr-mask-size-ipv6 defaults to /24 for IPv4 and /64 for IPv6. kubelet: --feature-gates="IPv6DualStack=true". kube-proxy: --cluster-cidr=<IPv4 CIDR>, <IPv6 CIDR> --feature-gates="IPv6DualStack=true" Note The IPv6DualStack feature is enabled by default in Kubernetes v1.21 or later.		
ra2.k8s.011	Anuket profile labels	To clearly identify which worker nodes are compliant with the different profiles defined by Anuket, the worker nodes must be labeled according to the following pattern: an anuket.io/profile/basic label must be set to true on the worker node if it can fulfill the requirements of the basic profile and an anuket.io/profile/network-intensive label must be set to true on the worker node if it can fulfill the requirements of the high-performance profile. The requirements for both profiles can be found in Architecture Requirements.		
ra2.k8s.012	Kubernetes APIs	Kubernetes Kubernetes Alpha API [21] are recommended only for testing, therefore all Alpha APIs must be disabled, except for those required by RA2 Ch4 Specifications currently NFD).		
ra2.k8s.013	Kubernetes APIs	Backward compatibility of all supported GA APIs of Kubernetes must be supported.		
ra2.k8s.014	Security groups	Kubernetes must support the NetworkPolicy feature.		
ra2.k8s.015	Publishing Services (ServiceTypes)	Kubernetes must support LoadBalancer Service (ServiceTypes) [22].		
ra2.k8s.016	Publishing Services (ServiceTypes)	Kubernetes must support Ingress [23].		

ra2.k8s.017	Publishing Services (ServiceTypes)	Kubernetes should support NodePort Service (ServiceTypes) [22].	inf.ntw.17 in Kubernetes Architecture Requirements	
ra2.k8s.018	Publishing Services (ServiceTypes)	Kubernetes should support ExternalName Service (ServiceTypes) [22].		
ra2.k8s.019	Kubernetes APIs	Kubernetes Beta APIs must be disabled, except for existing APIs as of Kubernetes 1.24 and only when a stable GA of the same version does not exist, or for APIs listed in RA2 Ch6 list of Mandatory API Groups.	int.api.04 in Kubernetes Architecture Requirements	
ra2.k8s.020	TLS Certificate management for workloads	Cert-manager [64] should be supported and integrated with a PKI certificate provider for workloads to request/renew TLS certificates. It must be configured to use strong hashing algorithms such as SHA-256 for all certificates. SHA-1 signed certificates are deprecated and will be rejected by default starting with Kubernetes 1.31.	int.api.04 in Kubernetes Architecture Requirements	kcm.gen.03

4.5 Container Runtimes

Table 23Container runtime specifications

Ref	Specification	Details	Requirement Trace	Reference Implementation Trace
ra2.crt.001	Conformance with the Open Container Initiative (OCI) 1.0 runtime specification	The container runtime must be implemented as per the OCI 1.0 [24] specification.	gen.ost.01 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.crt.002	Kubernetes Container Runtime Interface (CRI)	The Kubernetes container runtime must be implemented as per the Kubernetes Container Runtime Interface (CRI) [25]	gen.ost.01 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure

4.6 Networking Solutions

For the networking solutions to be conformant with the Reference Architecture, they must be implemented according to the following specifications:

Table 24Networking Solution Specifications

Ref	Specification	Details	Requirement Trace	Reference Implementation Trace
ra2.ntw.001	Centralized network administration	The networking solution deployed within the implementation must be administered through the Kubernetes API using native Kubernetes API resources and objects, or Custom Resources.	inf.ntw.03 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ntw.002	Default Pod Network - CNI	The networking solution deployed within the implementation must use a CNI-conformant Network Plugin for the Default Pod Network, as the alternative (kubenet) does not support cross-node networking or Network Policies.	gen.ost.01 in Kubernetes Architecture Requirements, inf.ntw.08 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ntw.003	Multiple connection points	The networking solution deployed within the implementation must support the capability to connect at least 5 connection points to each Pod, which are additional to the default connection point managed by the default Pod network CNI plugin.	e.cap.004 in Cloud Infrastructure Software Profile Capabilities	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ntw.004	Multiple connection points presentation	The networking solution deployed within the implementation must ensure that all	inf.ntw.03 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications

		additional non-default connection points are requested by Pods using standard Kubernetes resource scheduling mechanisms, such as annotations, or container resource requests and limits.		(RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ntw.005	Multiplexer/meta-plugin	The networking solution deployed within the implementation may use a multiplexer/meta-plugin.	inf.ntw.06 in Kubernetes Architecture Requirements, inf.ntw.07 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ntw.006	Multiplexer/meta-plugin CNI conformance	If used, the selected multiplexer/meta-plugin must integrate with the Kubernetes control plane via CNI.	gen.ost.01 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ntw.007	Multiplexer/meta-plugin CNI Plugins	If used, the selected multiplexer/meta-plugin must support the use of multiple CNI-conformant Network Plugins.	gen.ost.01 in Kubernetes Architecture Requirements, inf.ntw.06 Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ntw.008	SR-IOV device plugin for high performance	When hosting workloads that match the high-performance profile and require SR-IOV acceleration, a Device Plugin for SR-IOV must be used to configure the SR-IOV devices and	e.cap.013 in Reference Model for Cloud Infrastructure (RM) [1] Chapter 4, section Exposed Performance	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on

		advertise them to the kubelet.	Optimisation Capabilities`	Bare Metal Infrastructure
ra2.ntw.009	Multiple connection points with multiplexer / meta-plugin	When a multiplexer/meta-plugin is used, the additional non-default connection points must be managed by a CNI-conformant Network Plugin.	gen.ost.01 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ntw.010	User plane networking	When hosting workloads that match the high-performance profile, CNI network plugins that support the use of DPDK, VPP, and/or SR-IOV must be deployed as part of the networking solution.	infra.net.acc.cfg .001 in Reference Model for Cloud Infrastructure (RM) [1], Chapter 5, section Virtual Networking Profiles	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ntw.011	NATless connectivity	When hosting workloads that require source and destination IP addresses to be preserved in the traffic headers, a NATless CNI plugin that exposes the pod IP directly to the external networks (e.g. Calico, MACVLAN or IPVLAN CNI plugins) must be used.	inf.ntw.14 in Kubernetes Architecture Requirements	
ra2.ntw.012	Device Plugins	When hosting workloads matching the High Performance profile that require the use of FPGA, SR-IOV or other Acceleration Hardware, a Device Plugin for that FPGA or Acceleration Hardware must be used.	e.cap.016 and e.cap.013 in Reference Model for Cloud Infrastructure (RM) [1], Chapter 4, section Exposed Performance Optimisation Capabilities`	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure

ra2.ntw.013	Dual-stack CNI	The networking solution deployed within the implementation must use a CNI-conformant network plugin that is able to support dual-stack IPv4/IPv6 networking.	inf.ntw.04 in Kubernetes Architecture Requirements	
ra2.ntw.014	Security groups	The networking solution deployed within the implementation must support network policies.	infra.net.cfg.004 Reference Model for Cloud Infrastructure (RM) [1] Chapter 5, section Virtual Networking Profiles	
ra2.ntw.015	IPAM plugin for multiplexer	When a multiplexer/meta-plugin is used, a CNI-conformant IPAM network plugin must be installed to allocate IP addresses for secondary network interfaces across all nodes of the cluster.	inf.ntw.10 in Kubernetes Architecture Requirements	
ra2.ntw.016	Kubernetes Network Custom Resource Definition De-Facto Standard-compliant multiplexer/meta-plugin	When a multiplexer/meta-plugin is used, the multiplexer/meta-plugin must implement version 1.2 of the Kubernetes Network Custom Resource Definition De-facto Standard [26].	gen.ost.01 in Kubernetes Architecture Requirements	Reference Implementation based on RA2 specifications (RI2) [66] Chapter 4, section Installation on Bare Metal Infrastructure
ra2.ntw.017	Kubernetes Load Balancer	The networking solution deployed within the implementation must include a L4 (TCP/UDP - except QUIC) Load Balancer to steer inbound traffic across the primary interfaces of multiple CNF pods.	inf.ntw.15 in Kubernetes Architecture Requirements	

ra2.ntw.018	Kubernetes Load Balancer - API	The Load Balancer solution deployed per ra2.ntw.017 must support the Service type Loadbalancer API.	inf.ntw.15 in Kubernetes Architecture Requirements	
ra2.ntw.019	Kubernetes Load Balancer - API	The Load Balancer solution deployed per ra2.ntw.017 may support the Gateway API additionally.	inf.ntw.15 in Kubernetes Architecture Requirements	
ra2.ntw.020	Kubernetes Load Balancer - Advertisements	The Load Balancer solution deployed per ra2.ntw.017 must be capable of advertising the IPs of Services to external networks.	inf.ntw.15 in Kubernetes Architecture Requirements	
ra2.ntw.021	Kubernetes Load Balancer - Active/active Multipath	The Load Balancer solution deployed per ra2.ntw.017 must support multi-path advertisements in an active/active design, allowing the same service IP to be advertised by multiple cluster nodes.	inf.ntw.15 in Kubernetes Architecture Requirements	
ra2.ntw.022	Kubernetes Load Balancer - High Availability	The networking solution deployed per ra2.ntw.017 must be capable of fast failover. Upon node or pod failure, it must redirect traffic (i.e., advertisements/routes must be updated) in less than 5 seconds.	inf.ntw.15 in Kubernetes Architecture Requirements	
ra2.ntw.023	Time Sensitive Networking	Timing accuracy with PTP Hardware Clock and synchronization with SyncE.	e.cap.027 from Reference Model for Cloud Infrastructure (RM) [1] Chapter 4, section Exposed infrastructure capabilities	

4.7 Storage Components

For the storage solutions to be conformant with the Reference Architecture they must be implemented according to the following specifications:

Table 25Storage solution specifications

Ref	Specification	Details	Requirement Trace	Reference Implementation Trace
ra2.stg.001	Ephemeral storage	An implementation must support ephemeral storage, for the unpacked container images to be stored and executed from, as a directory in the filesystem on the worker node on which the container is running. See the Container runtimes section above for more information on how this meets the requirement for ephemeral storage for containers.		
ra2.stg.002	Kubernetes Volumes	An implementation may attach additional storage to containers using Kubernetes Volumes.		
ra2.stg.003	Kubernetes Volumes	An implementation may use Volume Plugins (see ra2.stg.005 below) to allow the use of a storage protocol (such as iSCSI and NFS) or management APIs (such as Cinder and EBS) for the attaching and mounting of storage into a Pod.		
ra2.stg.004	Persistent Volumes	An implementation may support Kubernetes Persistent Volumes (PV) to provide persistent storage for Pods. Persistent Volumes exist independent of the lifecycle of containers and/or pods.	inf.stg.01 in Kubernetes Architecture Requirements	
ra2.stg.005	Storage Volume Types	An implementation must support the following Volume types: emptyDir, ConfigMap, Secret, and		

		PersistentVolumeClaim. Other Volume plugins may be supported to allow for the use of a range of backend storage systems.		
ra2.stg.006	Container Storage Interface (CSI)	An implementation may support the Container Storage Interface (CSI). In-tree storage plugins for Ceph have been removed in Kubernetes 1.31, so corresponding CSI drivers must be used. To support CSI, the feature gates CSIDriverRegistry and CSINodeInfo must be enabled. The implementation must use a CSI driver (full list of CSI drivers [27]). An implementation may support ephemeral storage through a CSI-compatible volume plugin. In this case, the CSIInlineVolume feature gate must be enabled. An implementation may support Persistent Volumes through a CSI-compatible volume plugin. In this case, the CSIPersistentVolume feature gate must be enabled.		
ra2.stg.007	Storage Classes	An implementation should use Kubernetes Storage Classes to support automation and the separation of concerns between providers of a service and consumers of the service.		

Note

This Reference Architecture does not include any specifications for object storage, as this is neither a native Kubernetes object, nor something that is required by CSI drivers. Object storage is an application-level requirement that would ordinarily be provided by a highly scalable service offering, rather than being something an individual Kubernetes cluster could offer.

4.8 Service Meshes

Application service meshes are not in the scope of the architecture. The service mesh is a dedicated infrastructure layer for handling service-to-service communication. It is recommended to secure service-to-service communications within a cluster and to reduce the attack surface. The benefits of the service mesh framework are described in Using Transport Layer Security and Service Mesh. In addition to securing communications, the use of a service mesh extends Kubernetes capabilities regarding observability and reliability.

Network service mesh specifications are handled in Networking solutions.

4.9 Kubernetes Application Package Managers

For the application package managers to be conformant with the Reference Architecture, they must be implemented according to the following specifications:

Table 26Kubernetes Application Package Managers Specifications

Ref	Specification	Details	Requirement Trace	Reference Implementation Trace
ra2.pkg.001	API-based package management	A package manager must use the Kubernetes APIs to manage application artifacts. Cluster-side components such as Tiller must not be required.	int.api.02 in Kubernetes Architecture Requirements	
ra2.pkg.002	Helm version 3	All workloads must be packaged using Helm (version 3) charts.		

Helm version 3 has been chosen as the Application packaging mechanism to ensure compliance with the ONAP ASD NF descriptor specification [28] and ETSI SOL-001 rel. 4 MCIOP specification [71].

4.10 Kubernetes Workloads

For the Kubernetes workloads to be conformant with the Reference Architecture, they must be implemented according to the following specifications:

Table 27Kubernetes Workload specifications

Ref	Specification	Details	Requirement Trace	Reference Implementation Trace
ra2.app.006	Consumption of additional, non-default connection points	Any additional non-default connection points must be requested with workload annotations or resource requests and limits within the	int.api.01 in Kubernetes Architecture Requirements	N/A

		container spec passed to the Kubernetes API Server.		
ra2.app.007	Host Volumes	Workloads must not use hostPath volumes [77], as Pods with identical configuration (such as those created from a PodTemplate) may behave differently on different nodes due to different files on the nodes.	kcm.gen.02 in Kubernetes Architecture Requirements	N/A
ra2.app.008	Infrastructure dependency	Workloads must not rely on the availability of the control plane nodes for the successful execution of their functionality (that is, loss of the control plane nodes may affect non-functional behaviours, such as healing and scaling. However, components that are already running will continue to do so without issue).	TBD	N/A
ra2.app.009	Device plugins	Workload descriptors must use the resources advertised by the device plugins to indicate their need for an FPGA, SR-IOV, or other acceleration device.	TBD	N/A
ra2.app.010	Node Feature Discovery (NFD)	If the workload requires special hardware or software features from the worker node, these requirements must be described in the workload descriptors using the labels advertised by Node Feature Discovery [68].	TBD	N/A
ra2.app.011	Published helm chart	Helm charts of the CNF must be published in a helm registry and must not	CNF Testsuite, Rationale, Test if the Helm chart is published:	N/A

		be used from local copies.	helm_chart_published [78]	
ra2.app.012	Valid Helm chart	Helm charts of the CNF must be valid and should pass the helm lint validation.	CNF Testsuite, Rationale, Test if the Helm chart is valid: helm_chart_valid [79]	N/A
ra2.app.013	Rolling update	Rolling updates of the CNF must be possible using Kubernetes deployments.	CNF Testsuite, Rationale, To test if the CNF can perform a rolling update: rolling_update [80]	N/A
ra2.app.014	Rolling downgrade	Rolling downgrades of the CNF must be possible using Kubernetes deployments.	CNF Testsuite, Rationale, To check if a CNF version can be downgraded through a rolling_downgrade: rolling_downgrade [81]	N/A
ra2.app.015	CNI compatibility	The CNF must use CNI compatible networking plugins.	CNF Testsuite, Rationale, To check if the CNF is compatible with different CNIs: cni_compatibility [82]	N/A
ra2.app.016	Kubernetes API stability	The CNF must not use any Kubernetes alpha APIs, except for those required by the specifications in this chapter (for example, NFD).	CNF Testsuite, Rationale, To check if the CNF is compatible with different CNIs: cni_compatibility [82]	N/A
ra2.app.017	CNF resiliency (node drain)	The CNF must not lose data. It must continue to run, and its readiness probe outcome must be Success, even in the event of a node drain and consequent rescheduling.	CNF Testsuite, Rationale, Test if the CNF crashes when node drain occurs: node_drain [83]	N/A
ra2.app.018	CNF resiliency (network latency)	The CNF must not lose data. It must continue to run, and its readiness probe outcome must be Success, even if network latency of up to 2000 ms occurs.	CNF Testsuite, Rationale, Test if the CNF crashes when network latency occurs: pod_network_latency [84]	N/A
ra2.app.019	CNF resiliency (pod delete)	The CNF must not lose data. It must continue to run, and its readiness probe	CNF Testsuite, Rationale, Test if the CNF crashes when disk fill occurs: disk_fill [85]	N/A

		outcome must be Success, even if a pod delete occurs.		
ra2.app.020	CNF resiliency (pod memory hog)	The CNF must not lose data. It must continue to run, and its readiness probe outcome must be Success, even if a pod memory hog occurs.	CNF Testsuite, Rationale, Test if the CNF crashes when pod memory hog occurs: pod_memory_hog [86]	N/A
ra2.app.021	CNF resiliency (pod I/O stress)	The CNF must not lose data. It must continue to run, and its readiness probe outcome must be Success, even if pod I/O stress occurs.	CNF Testsuite, Rationale, Test if the CNF crashes when pod io stress occurs: pod_io_stress [87]	N/A
ra2.app.022	CNF resiliency (pod network corruption)	The CNF must not lose data. It must continue to run, and its readiness probe outcome must be Success, even if pod network corruption occurs.	CNF Testsuite, Rationale, Test if the CNF crashes when pod network corruption occurs: pod_network_corruption [88]	N/A
ra2.app.023	CNF resiliency (pod network duplication)	The CNF must not lose data. It must continue to run, and its readiness probe outcome must be Success, even if a pod network duplication occurs.	CNF Testsuite, Rationale, Test if the CNF crashes when pod network duplication occurs: pod_network_duplication [89]	N/A
ra2.app.024	CNF resiliency (pod DNS error)	The CNF must not lose data. It must continue to run, and its readiness probe outcome must be Success, even if a pod DNS error occurs.		N/A
ra2.app.025	CNF local storage	The CNF must not use local storage.	CNF Testsuite, Rationale, To test if the CNF uses local storage: no_local_volume_configuration [90]	N/A
ra2.app.026	Liveness probe	All Pods of the CNF must have livenessProbe defined.	CNF Testsuite, Rationale, To test if there is a liveness entry in the Helm chart: liveness [91]	N/A

ra2.app.027	Readiness probe	All Pods of the CNF must have readinessProbe defined.	CNF Testsuite, Rationale, To test if there is a readiness entry in the Helm chart: readiness [92]	N/A
ra2.app.028	No access to container daemon sockets	The CNF must not have any of the container daemon sockets (for example, /var/run/docker.sock, /var/run/containerd.sock or /var/run/crio.sock) mounted.		N/A
ra2.app.029	No automatic service account mapping	Non-specified service accounts must not be automatically mapped. To prevent this, the automountServiceAccountToken: false flag must be set in all Pods of the CNF.	CNF Testsuite, Rationale, To check if there is automatic mapping of service accounts: service_account_mapping [93]	N/A
ra2.app.030	No host network access	Host network must not be attached to any of the Pods of the CNF. The hostNetwork attribute of the Pod specifications must be False, or it should not be specified.	CNF Testsuite, Rationale, To check if there is a host network attached to a pod: host_network [94]	N/A
ra2.app.031	Host process namespace separation	The Pods of the CNF must not share the host process ID namespace or the host IPC namespace. The Pod manifests must not have the hostPID or the hostIPC attribute set to true.	CNF Testsuite, Rationale, To check if containers are running with hostPID or hostIPC privileges: host_pid_ipc_privileges [95]	N/A
ra2.app.032	Resource limits	All containers and namespaces of the CNF must have defined resource limits for at least the CPU and memory resources.	CNF Testsuite, Rationale, To check if containers have resource limits defined: resource_policies [96]	N/A
ra2.app.033	Read only filesystem	All the containers of the CNF must have a read-only filesystem. The readOnlyRootFilesystem attribute of the Pods	CNF Testsuite, Rationale, To check if containers have immutable file systems: immutable_file_systems [97]	N/A

		in their securityContext should be set to true.		
ra2.app.034	Container image tags	All the referred container images in the Pod manifests must be referred by a version tag pointing to a concrete version of the image. The latest tag must not be used.	Kubernetes documentation: Images [113]	N/A
ra2.app.035	No hardcoded IP addresses	The CNF must not have any hardcoded IP addresses in its Pod specifications.	CNF Testsuite, Rationale, To test if there are any (non-declarative) hardcoded IP addresses or subnet masks in the K8s runtime configuration: hardcoded_ip_addresses_in_k8s_runtime_configuration [98]	N/A
ra2.app.036	No node ports	The service declarations of the CNF must not contain a nodePort definition.	[107]	N/A
ra2.app.037	Immutable config maps	ConfigMaps used by the CNF must be immutable.	[109]	N/A
ra2.app.038	Horizontal scaling	If the CNF supports scaling, increasing and decreasing its capacity must be implemented using horizontal scaling. If horizontal scaling is supported, automatic scaling must be possible using Kubernetes Horizontal Pod Autoscaler (HPA) [110] feature.	TBD	N/A
ra2.app.039	CNF image size	The different container images of the CNF should not be bigger than 5GB.	CNF Testsuite, Rationale, To check if the CNF has a reasonable image size: reasonable_image_size [100]	N/A
ra2.app.040	CNF startup time	The startup time of the Pods of a CNF should not exceed 60 seconds, where the startup time is the time between the starting of	CNF Testsuite, Rationale, To check if the CNF have a reasonable startup time:	N/A

		the Pod and the readiness probe outcome registering Success.	reasonable_startup_time [101]	
ra2.app.041	No privileged mode	Pods of the CNF must not run in privileged mode.	CNF Testsuite, Rationale, To check if there are any privileged containers: privileged_containers [102]	N/A
ra2.app.042	No root user	Pods of the CNF must not run as a root user.	CNF Testsuite, Rationale, To check if any containers are running as a root user (checks the user outside the container that is running dockerd): non_root_user [103]	N/A
ra2.app.043	No privilege escalation	None of the containers of the CNF should allow privilege escalation.	CNF Testsuite, Rationale, To check if any containers allow for privilege escalation: privilege_escalation [104]	N/A
ra2.app.044	Non-root user	All the Pods of the CNF must be able to execute with a non-root user having a non-root group. Both the runAsUser and the runAsGroup attributes must be set to a value greater than 999.	CNF Testsuite, Rationale, To check if containers are running with non-root user with non-root membership: non_root_containers [105]	N/A
ra2.app.045	Labels	The Pods of the CNF should define at least the following labels: app.kubernetes.io/name, app.kubernetes.io/version and app.kubernetes.io/part-of	Kubernetes documentation: Recommended Labels [111]	N/A
ra2.app.046	Log output	The Pods of the CNF must direct their logs to sdout or stderr. This enables the treatment of the logs as event steams.	The Twelve Factor App: Logs [65]	N/A
ra2.app.047	Host ports	The Pods of the CNF should not use the host	CNF Testsuite, Rationale, To test if	N/A

		ports. Using the host ports ties the CNF to a specific node, thereby making the CNF less portable and scalable.	there are host ports used in the service configuration: hostport_not_used [106]	
ra2.app.048	SELinux options	If SELinux is used in the Pods of the CNF, the options used to escalate privileges should not be allowed. The options spec.securityContext.seLinuxOptions.type, spec.containers[*].securityContext.seLinuxOptions.type, spec.initContainers[].securityContext.seLinuxOptions, and spec.ephemeralContainers[*].securityContext.seLinuxOptions.type must either be unset altogether or set to one of the following allowed values container_t, container_init_t, or container_kvm_t.		N/A

4.11 Additional Required Components

This chapter should list any additional components needed to provide the services defined in the chapter Infrastructure Services (for example, Prometheus).

4.12 Platform Service Components

The architecture may support additional platform services, this chapter defines the requirements for the platform service components when the platform service is supported.

Table 28Platform service components requirements

Reference	Platform service category	Requirement	RM reference
ra2.plat.001	Data stores/databases	The platform may support any open source datastore or database technology	Reference Model [1] Chapter 5.1.5
ra2.plat.002	Streaming and messaging	The platform may support any Streaming and messaging technology	Reference Model [1] Chapter 5.1.5
ra2.plat.003	Load balancer and service proxy	If an external load balancer is used it must be exposed via the LoadBalancer property of the Kubernetes Service [108]	Reference Model [1] Chapter 5.1.5

ra2.plat.004	Load balancer and service proxy	If a load balancer is supported it must support workload resource scaling	pas.lb.001 in Reference Model [1] Chapter 5.1.5
ra2.plat.005	Load balancer and service proxy	If a load balancer is supported it must support resource resiliency	pas.lb.002 in Reference Model [1] Chapter 5.1.5
ra2.plat.006	Load balancer and service proxy	If a load balancer is supported it must support scaling and resiliency in the local environment	pas.lb.003 in Reference Model [1] Chapter 5.1.5
ra2.plat.007	Load balancer and service proxy	If a load balancer is supported it must support OSI Layer 3/4 load balancing	pas.lb.004 in Reference Model [1] Chapter 5.1.5
ra2.plat.008	Load balancer and service proxy	If a load balancer is supported it must support round-robin load balancing	pas.lb.005 in Reference Model [1] Chapter 5.1.5
ra2.plat.009	Load balancer and service proxy	If a load balancer is supported it must create event logs with the appropriate severity levels (catastrophic, critical, and so on)	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.010	Load balancer and service proxy	If a load balancer is supported it must support monitoring of endpoints	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.011	Load balancer and service proxy	If a load balancer is supported it must support Direct Server Return (DSR)	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.012	Load balancer and service proxy	If a load balancer is supported it must support stateful TCP load balancing	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.013	Load balancer and service proxy	If a load balancer is supported it must support UDP load balancing	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.014	Load balancer and service proxy	If a load balancer is supported it must support load balancing and the correct handling of fragmented packets	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.015	Service mesh	If a service mesh is supported the service must should support the Service Mesh Interface [112]	Reference Model [1] Chapter 5.1.5

ra2.plat.016	Monitoring	The platform may support any open source monitoring technology	Reference Model [1] Chapter 5.1.5
ra2.plat.017	Logging	The platform may support any open source logging technology	Reference Model [1] Chapter 5.1.5
ra2.plat.018	Logging	If a logging framework is supported it must support log management from multiple distributed sources	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.019	Logging	If a logging framework is supported it must manage log rotation at configurable periods	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.020	Logging	If a logging framework is supported it must manage log rotation at configurable log file status (%full)	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.021	Logging	If a logging framework is supported it must manage archival and retention of logs for configurable periods by different log types	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.022	Logging	If a logging framework is supported it must ensure log file integrity (no changes, particularlychanges that may affect the completeness, consistency, and accuracy, including event times, of the log file content) different log types	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.023	Logging	If a logging framework is supported it must monitor log rotation and log archival processes	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.024	Logging	If a logging framework is supported it must monitor the logging status of all the log sources	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.025	Logging	If a logging framework is supported it must ensure that the clock of each logging host is synchronized to a common time source	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.026	Logging	If a logging framework is supported it must support the reconfiguring of logging as needed, based on policy changes, technology changes, and other factors	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.027	Logging	If a logging framework is supported it must support the documenting and	pas.lb.006 in Reference

		reporting of anomalies in log settings, configurations, and processes	Model [1] Chapter 5.1.5
ra2.plat.028	Logging	If a logging framework is supported it must support the correlating of entries from multiple logs that relate to the same event	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.029	Logging	If a logging framework is supported it must support the correlating of multiple log entries from a single source or multiple sources, based on logged values (for example, event types, timestamps, and IP addresses)	pas.lb.006 in Reference Model [1] Chapter 5.1.5
ra2.plat.030	Application definition and image build	Kubernetes Application package managers must follow the specifications defined in Chapter 4.9 Kubernetes Application Package Managers	Reference Model [1] Chapter 5.1.5
ra2.plat.031	CI/CD	The platform may support any open source CI/CD technology	Reference Model [1] Chapter 5.1.5
ra2.plat.032	Ingress/egress controllers	The platform may support any open source Ingress/egress controllers technology	Reference Model [1] Chapter 5.1.5
ra2.plat.033	Ingress/egress controllers	The platform may support any open source Ingress/egress controllers technology	Reference Model [1] Chapter 5.1.5
ra2.plat.034	Ingress/egress controllers	If an egress controller is supported it must provide fixed and consistent source IP addresses for any given egress traffic.	
ra2.plat.035	Ingress/egress controllers	If an egress controller is supported it must support several source IP addresses on egress control.	
ra2.plat.036	Ingress/egress controllers	If an egress controller is supported it must provide a way to preserve the client IP address on egress control	
ra2.plat.037	Ingress/egress controllers	If an ingress and egress controller is supported it must support symmetric IP/VIP for ingress and egress	
ra2.plat.038	Ingress/egress controllers	If an egress controller is supported it must provide capabilities to route and isolate egress traffic based on traffic types (OAM, Signaling, etc), connected to e.g., to separate VRFs	
ra2.plat.039	Ingress/egress controllers	If an egress controller is supported it must support VLAN tagging for egress traffic	

ra2.plat.040	Ingress/egress controllers	If an egress controller is supported it must support the separation for overlapping destination address.	
ra2.plat.041	Network service	The platform may support any open source network service technology	Reference Model [1] Chapter 5.1.5
ra2.plat.042	Coordination and service discovery	The platform may support any open source coordination and service discovery technology	Reference Model [1] Chapter 5.1.5
ra2.plat.043	Automation and configuration	The platform may support any open source automation and configuration technology	Reference Model [1] Chapter 5.1.5
ra2.plat.044	Key management	The platform may support any open source key management technology	Reference Model [1] Chapter 5.1.5
ra2.plat.045	Tracing	The platform may support any open source tracing technology	Reference Model [1] Chapter 5.1.5

4.13 Kubernetes Cluster Lifecycle Management

This section describes the requirements for Kubernetes Cluster Lifecycle Management (LCM). A CaaS Manager is an architectural block responsible for the automated lifecycle management of Kubernetes clusters. It provides a centralized platform for performing operations like cluster creation, scaling, upgrades, and destruction. These guidelines align with the lifecycle management principles outlined in [1] Chapter 9, emphasizing automation, modularity, and decoupling of deployment and test steps. This adherence ensures consistent and reliable cluster operations throughout their lifecycle.

As there is no Reference Implementation for this part of the specification, the following table includes examples of implementation from the Cluster API spec, a sub-project of the Kubernetes Cluster Lifecycle SIG [114].

Table 29Kubernetes Cluster Lifecycle Management specifications

Ref	Specification	Details	Requirement Trace	Reference Implementation Trace
ra2.lcm.001	Cluster Creation and Destruction	The CaaS Manager must support automated cluster creation and destruction. This includes provisioning the underlying infrastructure and installing the Kubernetes control plane	Reference Model [1] Chapter 9 - Infrastructure lifecycle management automation	Cluster API: Cluster object

		and worker nodes.		
ra2.lcm.002	Cluster Scaling	The CaaS Manager must support automated cluster scaling. This includes scaling the number of worker nodes up or down based on demand.	Reference Model [1] Chapter 9 - Infrastructure lifecycle management automation	Cluster API: MachineDeployment object, scaling feature
ra2.lcm.003	Cluster Upgrades	The CaaS Manager must support automated cluster upgrades. This includes upgrading the Kubernetes control plane and worker nodes to newer versions.	Reference Model [1] Chapter 9 - Infrastructure lifecycle management automation	Cluster API: MachineDeployment object, upgrade strategies
ra2.lcm.004	Cluster Object	The CaaS Manager API must support a "Cluster" object. Represents a Kubernetes cluster.	Reference Model [1] Chapter 9 - Infrastructure lifecycle management automation	Cluster API: Cluster object
ra2.lcm.005	Node Object	The CaaS Manager API must support a "Node" object. Represents a single machine in the cluster, either a control plane or worker node.	Reference Model [1] Chapter 9 - Infrastructure lifecycle management automation	Cluster API: Machine object
ra2.lcm.006	Node Pool Object	The CaaS Manager API must support a "Node Pool" object. Represents a group of similar	Reference Model [1] Chapter 9 - Infrastructure lifecycle management automation	Cluster API: MachineSet object (for managing homogeneous groups of Machines)

		nodes with shared characteristics (e.g., machine type, labels).		
ra2.lcm.007	Node Deployment Strategy	The CaaS Manager API must support defining a strategy for node deployments, such as rolling updates or blue/green deployments.	Reference Model [1] Chapter 9 - Infrastructure lifecycle management automation	Cluster API: MachineDeployment object with rolling update strategies
ra2.lcm.008	Node Health Check	The CaaS Manager must support mechanisms for monitoring the health of nodes in the cluster and taking appropriate actions (e.g., replacing unhealthy nodes).	Reference Model [1] Chapter 9 - Infrastructure lifecycle management automation	Cluster API: MachineHealthCheck object (and integrations with provider-specific health checking)
ra2.lcm.009	Node Initialization Configuration	The CaaS Manager must support providing initial configuration data for nodes during their creation, such as cloud-init scripts or other bootstrap mechanisms.	Reference Model [1] Chapter 9 - Infrastructure lifecycle management automation	Cluster API: BootstrapConfig object (e.g., using KubeadmConfigTemplate)

5 Security Guidance

5.1 Introduction to Security Guidance

Securing Kubernetes requires a multi-layered security approach to protect cloud-native applications. The following best practices are recommended:

- Security testing should be fully integrated into the CI/CD pipelines of all stakeholders (e.g., vendors and operators).
- Automated security policies should flag builds with identified issues.
- Image registries should be monitored to automatically block or replace images with known vulnerabilities. Policies should also govern what can be deployed and who can deploy from the registry.
- A layered packaging model should be adopted to support separation of concerns during image builds.

The following functionalities are recommended for securing Kubernetes platforms:

- Image certification (vulnerability scanning) and signing.
- Role-Based Access Control (RBAC).
- Secret management.
- Addressing the limitations of hard Kubernetes cluster multitenancy:
  - For tenants without strict multitenancy requirements (such as multiple development teams within the same organization), namespaces can provide sufficient separation.
  - For strict multitenancy, a dedicated Kubernetes cluster per tenant is recommended.
- Integration with other security tools, such as monitoring and alerting systems.

5.1.1 Security Perimeters

Securing applications and workloads on Kubernetes involves securing several layers. Each layer requires its own perimeter security:

- **Container Registry:** The container registry stores and manages container images. Access must be secured to prevent unauthorized access and image tampering.
- **Container Images:** Container images are lightweight, portable executables containing software and dependencies. Before uploading to a registry, images should undergo security checks, including vulnerability analysis and scans. Images should also be signed by trusted sources.
- **Containers:** A container is a running instance of a container image. Containers should be prevented from accessing underlying OS capabilities (e.g., loading kernel modules, mounting OS directories) and should not run in privileged mode.
- **Pods:** A pod represents a set of running containers in a Kubernetes cluster. Kubernetes' Pod Security Policies define conditions for pod execution, ensuring necessary checks are performed.
- **Kubernetes Node (Worker Node):** A Kubernetes node (worker node) is a physical or virtual server running workloads. Unsecured nodes pose a significant threat. Nodes should be hardened by disabling unused ports, prohibiting root access, etc.

- **Kubernetes Control Plane Node:** An unsecured control plane node presents a significant threat. These nodes should be hardened by disabling unused ports, prohibiting root access, etc.
- **Kubernetes Control Plane:** The Kubernetes control plane orchestrates containers, exposing APIs and interfaces for managing their lifecycle. API communication should be secured using mechanisms such as TLS encryption and API authentication (e.g., via LDAP).

5.2 Security Principles

Core principles for securing cloud-native applications include:

- Deploy only secure and trusted applications and code.
- Deploy applications only from validated and verified images.
- Deploy applications only from trusted registries.
- Secure Kubernetes container orchestration with administrative boundaries between tenants:
  - Use Namespaces to establish security boundaries between tenants.
  - Create and define cluster Network Policies.
  - Implement cluster-wide pod security policies.
  - Enable audit logging.
  - Isolate sensitive workloads using Namespaces.
  - Secure tenant metadata access.
- Segment container networks using security zoning and network standards.
- Harden the host OS running the containers.
- Use container-aware runtime defense tools.
- Enable Role-Based Access Control (RBAC).

5.3 Node Hardening

5.3.1 Node Hardening: Securing the Kubernetes Hosts

Operating systems and applications often have insecure default settings (open ports, enabled services). Kubernetes nodes must be secured, hardened, and correctly configured according to established security frameworks (e.g., CIS benchmarks). These benchmarks provide configuration standards and best practices for hardening digital assets.

5.3.2 Restricting Direct Access to Nodes

Restrict root/administrative access to Kubernetes nodes and avoid direct access for operational tasks (debugging, troubleshooting, etc.).

5.3.3 Vulnerability Assessment

Regular vulnerability assessments are crucial for IT risk management. Implement vulnerability scanner tools (e.g., OpenVAS, or other open source or commercial tools) to identify and mitigate threats and vulnerabilities.

5.3.4 Patch Management

A robust patch management process should be implemented to address security vulnerabilities, ensure compliance, maintain uptime, and enable feature enhancements.

5.4 Securing the Kubernetes Orchestrator

Kubernetes control plane API communication must be secured using TLS encryption, API authentication (e.g., via LDAP), etc. Unsecured control plane nodes represent a significant threat; harden them by disabling unused ports and prohibiting root access.

Security recommendations for the orchestration manager include:

- Cluster management network isolation to protect the control plane and restrict administrative command execution. Use network isolation techniques, configure RBAC on the cluster manager, and configure node service accounts using the principle of least privilege.
- Enforce access control on registries using unique credentials to limit who can control builds or add images.
- Use TLS for all network access.
- Configure user roles and access levels to ensure segregation of duties.
  - Do not co-locate container and non-container services on the same node.
  - Do not run containers as root.
- Implement multi-factor authentication for all administrative access.
- Harden the configuration using Center for Internet Security (CIS) benchmarks for container runtimes and Kubernetes.
- Deploy security products providing allowlisting, behavior monitoring, and anomaly detection to prevent malicious activity.
- Avoid privileged container applications through policy management to mitigate potential attacks.
- Integrate with other security ecosystems (SIEM).
- Isolate environments (dev/test/production) within the cluster.
- Create administrative boundaries between resources using namespaces and avoid using default namespaces.
- Enable Seccomp to restrict actions available within container applications.
- Limit discovery by restricting access to cluster management metadata on configurations, containers, and nodes.

5.4.1 Control Network Access to Sensitive Ports

Kubernetes clusters use a range of ports, making them potential attack targets. Configure authentication and authorization on the cluster and nodes. The Kubernetes documentation [115] details default ports. Block access to unnecessary ports and limit access to the Kubernetes API server to trusted networks only.

Control Plane Node(s):

Protocol	Port Range	Purpose
TCP	6443	Kubernetes API Server

TCP	2379-2380	etcd server client API
TCP	10250	Kubelet API
TCP	10259	kube-scheduler
TCP	10257	kube-controller-manager

Worker Nodes:

Protocol	Port Range	Purpose
TCP	10250	Kubelet API
TCP	30000-32767	NodePort Services

5.4.2 Controlling Access to the Kubernetes API

The Kubernetes API is a primary security target. Control access and allowed actions carefully.

5.4.3 Using Transport Layer Security and Service Mesh

Secure inter-service communication within the cluster using TLS, encrypting all traffic by default. Kubernetes expects default TLS encryption for API communication; most installation methods facilitate certificate creation and distribution.

Note

Some components and installation methods might enable local ports over HTTP. Administrators should review component settings to identify potential unsecured traffic.

Service meshes (e.g., Linkerd, Istio) provide default TLS encryption and additional telemetry. A service mesh uses layer 7 proxies for service-to-service communication, comprising data plane (proxies paired with microservices) and control plane (proxy configuration, TLS certificate and policy management) components. NIST SP 800-204A Building Secure Microservices-based Applications Using Service-Mesh Architecture [116] and NIST SP 800-204B Attribute-based Access Control for Microservices-based Applications Using a Service Mesh [117] provide guidance on deploying service meshes.

5.4.4 API Authentication and Authorization

Secure Kubernetes cluster connections using the following authentication mechanisms:

- Configure user roles and access levels for segregation of duties (RBAC).
- Use multi-factor authentication for all administrative access.
- Use token-based or certificate-based service and session authentication.
- Integrate with existing identity management platforms (e.g., SAML, AD) for access control.

5.4.5 Restricting Access to etcd and Encrypting Contents Within etcd

etcd, a critical Kubernetes component storing state and secrets, requires robust protection. Write access to the API server's etcd grants root access to the entire cluster; even read access can be exploited for privilege escalation.

The Kubernetes scheduler uses etcd to find unscheduled pods, sending them to available kubelets. The API server validates submitted pods before writing them to etcd. Directly writing to etcd bypasses many security mechanisms (e.g., PodSecurityPolicies).

Use strong credentials (e.g., mutual TLS authentication via client certificates) for API server-etcd communication. Isolate etcd servers behind a firewall accessible only by the API servers.

5.4.6 Controlling Access to the Kubelet

Kubelets expose HTTPS endpoints controlling nodes and containers. Production clusters should enable kubelet authentication and authorization (unauthenticated access is insecure by default).

5.4.7 Securing the Kubernetes Dashboard

The Kubernetes dashboard, a web application for cluster management (not a core Kubernetes component), requires careful security configuration. Many tutorials create highly privileged service accounts, making it a vulnerability (Reference: Tesla cloud resources are hacked to run cryptocurrency-mining malware [118]).

To prevent dashboard attacks:

- Do not publicly expose the dashboard without additional authentication.
- Enable RBAC to limit the dashboard's service account privileges.
- Review and restrict the service account's assigned privileges.
- Implement granular permissions per user.
- Block dashboard requests from internal pods using network policies (kubectl proxy will still function).
- Verify that no cluster admin role binding exists (a vulnerability in versions prior to 1.8).
- Deploy the dashboard with an authenticating reverse proxy and multi-factor authentication (using embedded OpenID Connect (OIDC) id_tokens or Kubernetes Impersonation). This allows using user credentials instead of a privileged ServiceAccount, suitable for on-prem and managed cloud clusters.

5.5 Using Namespaces to Establish Security Boundaries

Kubernetes namespaces provide the first level of isolation between components. Apply security controls (network policies, pod policies, etc.) to workloads in separate namespaces.

5.6 Separating Sensitive Workloads

Run sensitive workloads on dedicated nodes to minimize the impact of a compromise. This reduces the risk of accessing sensitive applications through less secure applications sharing a runtime or host. Node pools and Kubernetes namespaces can facilitate this separation.

5.7 Creating and Defining Network Policies

Network policies control network access to cloud-native applications. Define clear ingress and egress policies and modify default policies (e.g., blocking or allowing traffic from other namespaces/clusters) where policy support is enabled.

5.8 Running the Latest Version

Regularly update to the latest Kubernetes release to benefit from new security features and patches.

5.9 Securing Platform Metadata

Kubernetes metadata contains sensitive information (including kubelet admin credentials). Secure it using encryption to prevent theft and privilege escalation.

- Limit access to cluster management metadata (configurations, containers, nodes).
- Ensure all metadata is encrypted and network access runs over TLS.

5.10 Enabling Logging and Monitoring

Enable and monitor audit logs for anomalous or unauthorized API calls, paying close attention to authorization failures. Logging, monitoring, and alerting are critical for Kubernetes security.

5.11 Runtime Security

Container runtime best practices include:

- Integrate runtime processes with Security Information and Event Monitoring (SIEM).
- Use container-aware runtime defense tools.
- Ensure all applications are from secure and verified images.
- Do not run applications with root privileges.
- Properly segment sensitive workloads using namespaces or clusters to limit the impact of compromises.

5.12 Secrets Management

Apply the principle of least privilege to secrets management in Kubernetes:

- Applications should only access necessary secrets.
- Use different secrets for different environments (production, development, testing).

Protect secret values (sensitive data) at rest and in transit. TLS encrypts traffic between Kubernetes control plane and worker nodes.

Avoid storing secrets in scripts or code; instead, provide them dynamically at runtime. Use a secure data repository (key manager, vault) for sensitive data (SSH keys, API keys, database credentials). Retrieve credentials on demand over secure channels to prevent writing unprotected data to disk. Back up key manager or vault encryption keys using a FIPS 140-2 Hardware Security Module.

- Check for hardcoded passwords, keys, and other sensitive items in container applications.
- Use security tools to automate scanning for hardcoded sensitive items.

5.13 Trusted Registry

Use trusted container registries accepting images only from validated sources. For third-party images, establish a formal validation process ensuring compliance with security requirements. Enforce access control using unique credentials to limit who can control builds or add images.

- Secure registry network access using TLS/SSL/VPN.
- Validate and scan container applications for viruses and vulnerabilities. Deploy only images signed with trusted keys.
- Use image versioning to ensure the latest certified applications are deployed.
- Trusted registries should reject improperly signed containers.

- Use approved registries for production images.
- Consider using third-party products for pre- and post-deployment container validation.

Remove stale, unsafe, and vulnerable images from the registry (using time triggers and image labels).

5.14 Isolation

5.14.1 VM versus Container Isolation

Directly comparing container and VM isolation is misleading. They differ fundamentally:

- **VMs:** Provide hard isolation at the layers underlying application software.
- **Containers:** Rely on OS, container runtime, and Kubernetes software-based mechanisms. Container workloads are sets of Linux processes; additional software-based isolation (e.g., kernel namespaces) is possible.

The primary isolation mechanism in Kubernetes should be VM-based or physical machine-based. Deploy multiple applications in the same Kubernetes cluster only after careful planning and verification of their compatibility. The default recommendation is one namespace per Cloud-Native Network Function (CNF).

5.14.2 Container Isolation in the Kubernetes Cluster

5.14.2.1 Namespaces

Use Kubernetes namespaces for resource isolation within a cluster. Do *not* use them to isolate deployment stages (development, production, testing). Dedicated clusters provide the most reliable separation for sensitive workloads.

6 API and Feature Testing requirements

6.1 Introduction to API and Feature Testing requirement

The CNCF has defined a [119] to help the community to write and run tests, and to contribute, analyze, and act upon test results. This chapter maps the requirements written in the previous chapters as mandatory Special Interest Group features. It enforces the overall requirements traceability to testing, especially those offered for [120]. The Anuket Reference Conformance testing matches the features and tests defined here.

6.1.1 Kubernetes feature gate policy

[121] are a set of key-value pairs that describe the Kubernetes features. The components of the control plane of the Kubernetes Clusters can be run with different Feature Gate settings.

A feature can be in the Alpha, Beta, or General Availability (GA) stage:

- Alpha features are disabled by default. Breaking API changes may be expected. They may contain bugs, and support may be dropped.
- Beta features are disabled by default. They are well tested, and support will not be dropped, although breaking API changes may happen. As of 1.24, any existing Beta feature will continue to be enabled by default. However, new beta APIs and features will not be enabled by default after Kubernetes 1.24. For more information, see [122]
- GA features are stable. They are always enabled and cannot be disabled.

Only those Kubernetes features can be made mandatory in this Reference Architecture which are GA or were Beta before Kubernetes 1.24.

A list of feature gates is available here [121].

6.1.2 Kubernetes API policy

The [123] supports all operations and communications between components, and external user commands. Everything in the Kubernetes platform is treated as an API object. Different API versions indicate different levels of stability and support. An API can have Alpha, Beta or Stable versions. The version of APIs that are backed by a feature will match the stage of the feature itself (i.e. Alpha, Beta or GA or Stable).

The policy for RA2 to include Kubernetes APIs as mandatory is as follows:

In these Reference Architecture APIs, only those APIs which are in any of the following stages are mandatory:

- Stable.
- Beta when introduced before Kubernetes version 1.24.
- Alpha or Beta when required by RA2 Ch4 specifications, or when included on the list of Mandatory API Groups below.

The Kubernetes API reference is available here [124].

The list of [125] that are mandatory is as follows:

Table 30Mandatory API Groups

Group	Version
admissionregistration.k8s.io	v1
apiextensions.k8s.io	v1

apiregistration.k8s.io	v1
apps	v1
authentication.k8s.io	v1
authorization.k8s.io	v1
autoscaling	v1, v2
batch	v1
certificates.k8s.io	v1
coordination.k8s.io	v1
core	v1
discovery.k8s.io	v1
events.k8s.io	v1
flowcontrol.apiserver.k8s.io	v1
networking.k8s.io	v1
node.k8s.io	v1
policy	v1
rbac.authorization.k8s.io	v1
scheduling.k8s.io	v1
storage.k8s.io	v1

6.2 API Machinery Special Interest Group [126]

Table 31API Machinery Special Interest Group

Labels	Mandatory	Description
Conformance	X	Kubernetes conformance test
None	X	Kubernetes mainstream features
Feature:ComprehensiveNamespaceDraining	X	The deletion of namespaces should always be fast (all 100 namespaces in 150 seconds).
Feature: CrossNamespacePodAffinity [135]		The CrossNamespacePodAffinity feature verifies the ResourceQuota with the cross namespace pod affinity scope using scope-selectors.
Feature: PodPriority [135]	X	The PodPriority feature verifies the ResourceQuota's priority class scope against a pod with a different priority class.

Feature:ScopeSelectors	X	Verify ResourceQuota with terminating scopes through scope selectors
Feature: StorageVersionAPI [137]		Enable the storage version API.
Feature:WatchList [121]		Enable support for streaming initial state of objects in watch requests.

6.3 Apps [127]

Table 32Apps Special Interest Group

Labels	Mandatory	Description
Conformance	X	Kubernetes conformance test
None	X	Kubernetes mainstream features
Feature: DaemonSetUpdateSurge [138]		The Daemon set should surge the pods onto the nodes when the specification is updated and the update strategy is RollingUpdate.
Feature: IndexedJob [139]		The IndexedJob feature should create pods for an indexed job with completion indexes.
Feature: StatefulSet [140]		The StatefulSet feature should create a working zookeeper cluster.
Feature:StatefulUpgrade		The StatefulUpgrade feature should maintain a functioning cluster.
Feature: SuspendJob [141]		The SuspendJob feature should not create pods when they have been created in a suspended state.
Feature: TaintEviction [142]		All pods on the unreachable node should be marked as NotReady when the node condition is set to NotReady. All pods should be evicted after eviction timeout has passed.
Feature: TTLAfterFinished [143]	X	The job should be deleted once it has finished, after the TTL has elapsed.

6.4 Auth Special Interest Group [128]

Table 33Auth Special Interest Group

Labels	Mandatory	Description
Conformance	X	Kubernetes conformance test
None	X	Kubernetes mainstream features
Feature: BoundServiceAccountTokenVolume [144]		The ServiceAccount admission controller migration upgrade should maintain a functioning cluster.

Feature:ClusterTrustBundle [121]		Enable ClusterTrustBundle objects and kubelet integration.
Feature:NodeAuthenticator	X	The kubelet's main port 10250 should reject requests with no credentials.
Feature:NodeAuthorizer	X	Setting existing and non-existent attributes should return with a Forbidden error, not a NotFound error.
NodeFeature:FSGroup	X	ServiceAccounts should set ownership and permission when RunAsUser or FsGroup is present.

6.5 Cluster Lifecycle Special Interest Group [129]

Table 34Cluster Lifecycle Special Interest Group

Labels	Mandatory	Description
Conformance	X	Kubernetes conformance test
None	X	Kubernetes mainstream features
Feature:BootstrapTokens	X	The BootstrapTokens feature should delete the token secret when the secret has expired.

6.6 Instrumentation Special Interest Group [130]

Table 35Instrumentation Special Interest Group

Labels	Mandatory	Description
Conformance	X	Kubernetes conformance test
None	X	Kubernetes mainstream features
Feature:Elasticsearch		The Elasticsearch feature should check that the Kibana logging instance is alive.
Feature: StackdriverAcceleratorMonitoring		Stackdriver Monitoring should have accelerator metrics.
Feature:StackdriverCustomMetrics		Stackdriver Monitoring should run Custom Metrics - Stackdriver Adapter for the new resource model.
Feature:StackdriverExternalMetrics		Stackdriver Monitoring should run Custom Metrics - Stackdriver Adapter for external metrics.
Feature:StackdriverMetadataAgent		Stackdriver Monitoring should run Stackdriver Metadata Agent.
Feature:StackdriverMonitoring		

6.7 Network Special Interest Group [131]

Table 36Network Special Interest Group

Labels	Mandatory	Description
Conformance	X	Kubernetes conformance test.
None	X	Kubernetes mainstream features.
Feature:Example		The example feature should create a pod that uses DNS.
Feature:Ingress		The Ingress feature should prevent ingress creation if more than one IngressClass is marked as a default.
Feature: IPv6DualStack [145]		IPv4/IPv6 dual-stack networking enables the allocation of both IPv4 and IPv6 addresses to Pods and Services. IPv4/IPv6 dual-stack networking is enabled by default for your Kubernetes cluster from 1.21 onwards, allowing the simultaneous assignment of IPv4 and IPv6 addresses.
Feature:kubemci		The kubemci feature should create ingress with a preshared certificate.
Feature:KubeProxyDaemonSetMigration		The upgrade of kube-proxy from static pods to a DaemonSet should maintain a functioning cluster.
Feature:KubeProxyDaemonSetUpgrade		The upgrade of kube-proxy from static pods to a DaemonSet should maintain a functioning cluster.
Feature:NEG		The NEG feature should sync the endpoints to NEG.
Feature:NoSNAT	X	The NoSNAT feature should be able to send traffic between the Pods without SNAT.
Feature:Networking-IPv4	X	Networking-IPv4 should provide an IPv4 connection for the containers.
Feature:Networking-IPv6		Networking-IPv6 should provide an IPv6 connection for the containers.
Feature:Networking-Performance	X	Measure network responsiveness, latency (both RTT and OWD), and throughput with the iperf2 tool.

Feature:NetworkPolicy		NetworkPolicy between the server and the client should enforce a policy to allow traffic only from a different namespace, based on NamespaceSelector.
Feature:PerformanceDNS		The PerformanceNDS feature should answer DNS queries for a maximum number of services per cluster.
Feature:SCTP		SCTP should allow the creation of a basic SCTP service with the pod and the endpoints.
Feature:SCTPConnectivity		The Pods should function for intra-pod communication: sctp.
Feature:ServiceCIDRs [146]		Track IP address allocations for Service cluster IPs using IPAddress objects.

6.8 Node Special Interest Group [132]

Table 37Node Special Interest Group

Labels	Mandatory	Description
Conformance	X	Kubernetes conformance test.
None	X	Kubernetes mainstream features.
Feature:DynamicResourceAllocation [121]		Enables support for resources with custom parameters and a lifecycle that is independent of a Pod.
Feature:Example	X	The liveness pods should be automatically restarted.
Feature: ExperimentalResourceUsageTracking		Resource tracking for 100 pods per node.
Feature:GPUUpgrade		The Control Plane node upgrade should not disrupt the GPU Pod.
Feature:InPlacePodVerticalScaling [121]		Enables in-place Pod vertical scaling.
Feature:NodeLogQuery [121]		Enables querying logs of node services using the /logs endpoint.
Feature:PodGarbageCollector		The PodGarbageCollector feature should handle the creation of 1000 pods.
Feature:PodLifecycleSleepAction [121]		Enables the sleep action in Container lifecycle hooks.
Feature:RegularResourceUsageTracking		Resource tracking for 0 pods per node.

Feature:SidecarContainers [121]		Allow setting the restartPolicy of an init container to Always so that the container becomes a sidecar container (restartable init containers).
Feature:UserNamespacesSupport [121]		Enable user namespace support for Pods.
Feature: ProbeTerminationGracePeriod [147]	X	The probing container should override timeoutGracePeriodSeconds when the LivenessProbe field is set.
NodeFeature: DownwardAPIHugePages [148]		Downward API tests for huge pages should provide the container's limits.hugepages-pagesize, and requests.hugepages-pagesize as environmental variables.
NodeFeature: PodReadinessGate [149]	X	The Pods should support the pod readiness gates.
NodeFeature:RuntimeHandler		The RuntimeClass feature should run a Pod requesting a RuntimeClass with a configured handler.
NodeFeature: Sysctls [150]	X	The Sysctls feature should not launch unsafe, but not explicitly enabled sysctls on the node.

6.9 Scheduling Special Interest Group [133]

Table 38Scheduling Special Interest Group

Labels	Mandatory	Description
Conformance	X	Kubernetes conformance test.
None	X	Kubernetes mainstream features.
Feature:GPUDevicePlugin		The GPUDevicePlugin feature runs Nvidia GPU Device Plugin tests.
Feature: LocalStorageCapacityIsolation [151]	X	The LocalStorageCapacityIsolation feature validates local ephemeral storage resource limits of pods that are allowed to run.
Feature:Recreate		The Recreate feature runs Nvidia GPU Device Plugin tests with a recreation.

6.10 Storage Special Interest Group [134]

Table 39Storage Special Interest Group

Labels	Mandatory	Description
Conformance	X	Kubernetes conformance test.
None	X	Kubernetes mainstream features.
Feature:ExpandInUsePersistentVolumes		
Feature:Flexvolumes		
Feature:RecoverVolumeExpansionFailure [121]		Enables users to edit their PVCs to smaller sizes so as they can recover from previously issued volume expansion failures.
Feature:SELinux		
Feature:GKELocalSSD		
Feature:VolumeSnapshotDataSource		
Feature:Volumes	X	
Feature:vsphere		
Feature:Windows		
NodeFeature:EphemeralStorage	X	
NodeFeature:FSGroup	X	

6.11 Conformance testing

The objective of this section is to provide an automated mechanism to validate Kubernetes based cloud infrastructure against the standard set of requirements defined in Architecture Requirements. Through this validation mechanism, a provider of cloud infrastructure will be able to test their cloud infrastructure's conformance to this reference architecture. This will ease the integration of network functions into operator environments that host compatible cloud infrastructures, thereby reducing cost, complexity, and time of integration.

The overall workstream requires the close coordination of the following:

- **Requirements** - The agreed upon capabilities and conditions that a compliant cloud infrastructure must provide or satisfy.
- **Tests** - The verification mechanism that determines that a given cloud infrastructure complies with one or more requirements.
- **Conformance Specifications** - The definition of the requirements, tests, and circumstances (test case integration, etc.) that must be met to be deemed conformant.

6.11.1 Requirements and Testing Principles

In addition to the description of the test case integration and tooling, Cloud Infrastructure Reference Architecture managed by OpenStack [152] defines the requirements and the

testing principles to which every Anuket Conformance test suite must comply as this one for Kubernetes.

In two words, the verification, validation, and conformance processes leverage existing Anuket testing knowledge (projects) and experience (history) by utilising the toolchain design already in-place.

The Kubernetes based cloud infrastructure suite **MUST** utilise the Anuket test case integration toolchain to deliver overall integration, the same end user actions, and a unique test result format (e.g. Anuket test result database) needed by the end users and any test case result verification program.

To reach all goals in terms of verification, validation, compliance, and conformance, all test cases **MUST** be delivered as Docker containers [153] to simplify the CI toolchain setup including:

- the common test case execution
- the unified way to manage all the interactions with the CI/CD components and with third-parties (e.g. dump all test case logs and results for conformance)

The Docker containers proposed by the test projects **MUST** also embed the Xtesting Python package [154] and the related test case execution description files [155] as required by Xtesting.

Xtesting CI [176] leverages the common test case execution proposed by Xtesting. Thanks to a simple test case list, this tool deploys plug-and-play CI/CD toolchains in a few commands [156] to run this Conformance testing.

See Cloud Infrastructure Reference Architecture managed by OpenStack [152] for more details.

6.11.2 Test Case traceability

6.11.2.1 Kubernetes API testing

The primary objectives of the e2e tests [120] are to ensure a consistent and reliable behavior of the Kubernetes code base, and to catch hard-to-test bugs before users do, when unit and integration tests are insufficient. They are partially selected for the Software Conformance Certification program [20] run by the Kubernetes community (under the aegis of the CNCF).

Anuket shares the same goal to give end users the confidence that when they use a certified product they can rely on a high level of common functionality. Then the Conformance testing starts with the test list defined by the Certified Kubernetes Conformance Program [20] which is expected to grow according to the ongoing requirement traceability.

End-to-End Testing [120] basically asks for focus and skip regexes to select or to exclude single tests:

- focus basically matches Conformance or Testing Special Interest Groups [119] in sub-sections below
- skip excludes the SIG labels listed as optional in API and Feature Testing requirements.

The Reference Conformance suites must be stable and executed on real deployments. Then all the following labels are defacto skipped in the End-to-End Testing [120]:

- alpha
- Disruptive
- Flaky

It is worth mentioning that no alpha or Flaky test can be included in Conformance as per the rules.

K8s Conformance

It must be noted that the default K8s Conformance [20] testing is disruptive thus this Conformance testing rather picks non-disruptive-conformance [157] testing as defined by Sonobuoy [158].

focus: [Conformance]

6.11.2.1.1 skip:

- [Disruptive]
- NoExecuteTaintManager

API Machinery Testing

focus: [sig-api-machinery]

6.11.2.1.2 skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:CrossNamespacePodAffinity]
- [Feature:CustomResourceValidationExpressions]
- [Feature:StorageVersionAPI]
- [Feature:WatchList]

See API Machinery Special Interest Group [126] and API and Feature Testing requirements for more details.

6.11.2.1.3

Apps Testing

focus: [sig-apps]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:DaemonSetUpdateSurge]
- [Feature:IndexedJob]
- [Feature:StatefulSet]
- [Feature:StatefulSetAutoDeletePVC]
- [Feature:StatefulUpgrade]
- [Feature:SuspendJob]

See Apps Special Interest Group [127] and API and Feature Testing requirements for more details.

Auth Testing

focus: [sig-auth]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:BoundServiceAccountTokenVolume]
- [Feature:ClusterTrustBundle]
- [Feature:PodSecurityPolicy]

See Auth Special Interest Group [128] and API and Feature Testing requirements for more details.

Cluster Lifecycle Testing

focus: [sig-cluster-lifecycle]

skip:

- [alpha]
- [Disruptive]
- [Flaky]

See Cluster Lifecycle Special Interest Group [129] and API and Feature Testing requirements for more details.

Instrumentation Testing

focus: [sig-instrumentation]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:Elasticsearch]
- [Feature:StackdriverAcceleratorMonitoring]
- [Feature:StackdriverCustomMetrics]
- [Feature:StackdriverExternalMetrics]
- [Feature:StackdriverMetadataAgent]
- [Feature:StackdriverMonitoring]

See Instrumentation Special Interest Group [130] and API and Feature Testing requirements for more details.

Network Testing

The regexes load.balancer, LoadBalancer and Network.should.set.TCP.CLOSE_WAIT.timeout are currently skipped because they haven't been covered successfully neither by the upstream kubernetes gate [159] nor by the Anuket verification [160].

Please note that a couple of tests must be skipped by name below as they are no appropriate labels.

focus: [sig-network]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:Example]
- [Feature:Ingress]
- [Feature:IPv6DualStack]
- [Feature:kubemci]
- [Feature:KubeProxyDaemonSetMigration]
- [Feature:KubeProxyDaemonSetUpgrade]
- [Feature:NEG]
- [Feature:Networking-IPv6]
- [Feature:NetworkPolicy]
- [Feature:PerformanceDNS]
- [Feature:ProxyTerminatingEndpoints]
- [Feature:SCTP]
- [Feature:SCTPConnectivity]
- [Feature:ServiceCIDRs]
- DNS configMap nameserver
- load.balancer
- LoadBalancer
- Network.should.set.TCP.CLOSE_WAIT.timeout

6.11.2.4.8 See Network Special Interest Group [131] and API and Feature Testing requirements for more details.

Node Testing

focus: [sig-node]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:DynamicResourceAllocation]
- [Feature:ExperimentalResourceUsageTracking]
- [Feature:GRPCContainerProbe]
- [Feature:GPUUpgrade]

- [Feature:InPlacePodVerticalScaling]
- [Feature:KubeletCredentialProviders]
- [Feature:NodeLogQuery]
- [Feature:PodGarbageCollector]
- [Feature:PodLifecycleSleepAction]
- [Feature:RegularResourceUsageTracking]
- [Feature:SidecarContainers]
- [Feature:UserNamespacesSupport]
- [Feature:UserNamespacesStatelessPodsSupport]
- [NodeFeature:DownwardAPIHugePages]
- [NodeFeature:RuntimeHandler]

See Node Special Interest Group [132] and API and Feature Testing requirements for more details.

Scheduling Testing

6.11.2.1.9 focus: [sig-scheduling]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:GPUDevicePlugin]
- [Feature:Recreate]

See Scheduling Special Interest Group [133] and API and Feature Testing requirements for more details.

6.11.2.1.10

Storage Testing

It should be noted that all in-tree driver testing, [Driver:+], is skipped. Conforming to the upstream gate [159], all PersistentVolumes NFS testing is also skipped. The following exclusions are about the deprecated in-tree GitRepo volume type [161]:

- should provision storage with different parameters
- should not cause race condition when used for git_repo

Please note that a couple of tests must be skipped by name below as they are no appropriate labels.

focus: [sig-storage]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Driver:+]
- [Feature:ExpandInUsePersistentVolumes]

- [Feature:Flexvolumes]
- [Feature:GKELocalSSD]
- [Feature:VolumeSnapshotDataSource]
- [Feature:Flexvolumes]
- [Feature:RecoverVolumeExpansionFailure]
- [Feature:SELinux]
- [Feature:vsphere]
- [Feature:Volumes]
- [Feature:Windows]
- [NodeFeature:EphemeralStorage]
- PersistentVolumes.NFS
- should provision storage with different parameters
- should not cause race condition when used for git_repo

See Storage Special Interest Group [134] and API and Feature Testing requirements for more details.

6.11.2.2 Kubernetes API benchmarking

Rally [162] is a tool and framework that performs Kubernetes API benchmarking.

Functest Kubernetes Benchmarking [163] proposed a Rally-based test case, xrally_kubernetes_full [164], which iterates 10 times the mainline xrally-kubernetes [162] scenarios.

At the time of writing, no KPI is defined in Architecture Requirements which would have asked for an update of the default SLA (maximum failure rate of 0%) proposed in Functest Kubernetes Benchmarking [163]

Functest xrally_kubernetes_full [163]:

Table 40Kubernetes API benchmarking

Scenarios	Iterations
Kubernetes.create_and_delete_deployment	10
Kubernetes.create_and_delete_job	10
Kubernetes.create_and_delete_namespace	10
Kubernetes.create_and_delete_pod	10
Kubernetes.create_and_delete_pod_with_configmap_volume	10
Kubernetes.create_and_delete_pod_with_configmap_volume [2]	10
Kubernetes.create_and_delete_pod_with_emptydir_volume	10
Kubernetes.create_and_delete_pod_with_emptydir_volume [2]	10
Kubernetes.create_and_delete_pod_with_hostpath_volume	10
Kubernetes.create_and_delete_pod_with_secret_volume	10
Kubernetes.create_and_delete_pod_with_secret_volume [2]	10

Kubernetes.create_and_delete_replicaset	10
Kubernetes.create_and_delete_replication_controller	10
Kubernetes.create_and_delete_statefulset	10
Kubernetes.create_check_and_delete_pod_with_cluster_ip_service	10
Kubernetes.create_check_and_delete_pod_with_cluster_ip_service [2]	10
Kubernetes.create_check_and_delete_pod_with_node_port_service	10
Kubernetes.create_rollout_and_delete_deployment	10
Kubernetes.create_scale_and_delete_replicaset	10
Kubernetes.create_scale_and_delete_replication_controller	10
Kubernetes.create_scale_and_delete_statefulset	10
Kubernetes.list_namespaces	10

The following software versions are considered to benchmark Kubernetes v1.29 (latest stable release) selected by Anuket:

Table 41Software versions

Software	Version
Functest	v1.29
xrally-kubernetes	1.1.1.dev14+g2ffa85a

6.11.2.3 Dataplane benchmarking

Kubernetes perf-tests repository [165] hosts various Kubernetes-related performance test related tools especially netperf [166] which benchmarks Kubernetes networking performance.

As listed in netperf's README [166]'s README, the 5 major network traffic paths are combination of pod IP vs virtual IP and whether the pods are co-located on the same node versus a remotely located pod:

- same node using pod IP
- same node using cluster/virtual IP
- remote node using pod IP
- remote node using cluster/virtual IP
- same node pod hairpin to itself using cluster/virtual IP

It should be noted that netperf [166] leverages iperf [167] (both TCP and UDP modes) and Netperf [168].

At the time of writing, no KPI is defined in Anuket chapters which would have asked for an update of the default SLA proposed in Functest Kubernetes Benchmarking [163].

6.11.2.4 Security testing

There are a couple of opensource tools that help securing the Kubernetes stack. Amongst them, Functest Kubernetes Security [169] offers two test cases based on kube-hunter [170] and kube-bench [171].

kube-hunter [170] hunts for security weaknesses in Kubernetes clusters and kube-bench [171] checks whether Kubernetes is deployed securely by running the checks documented in the CIS Kubernetes Benchmark [172].

kube-hunter [170] classifies all vulnerabilities as low, medium, and high. In context of this conformance suite, all vulnerabilities are only printed for information.

Here are the vulnerability categories [173] tagged as high by kube-hunter [170]:

- ExposedSensitiveInterfacesTechnique
- MountServicePrincipalTechnique
- ListK8sSecretsTechnique
- InstanceMetadataApiTechnique
- ExecIntoContainerTechnique
- SidecarInjectionTechnique
- NewContainerTechnique
- GeneralPersistenceTechnique
- HostPathMountPrivilegeEscalationTechnique
- PrivilegedContainerTechnique
- ClusterAdminBindingTechnique
- CoreDNSPoisoningTechnique
- DataDestructionTechnique
- GeneralDefenseEvasionTechnique
- CVERemoteCodeExecutionCategory
- CVEPrivilegeEscalationCategory

At the time of writing, none of the Center for Internet Security (CIS) rules are defined as mandatory (e.g., sec.std.001: The Cloud Operator **should** comply with Center for Internet Security CIS Controls) else it would have required an update of the default kube-bench [171] behavior (all failures and warnings are only printed) as integrated in Functest Kubernetes Security [169].

The following software versions are considered to verify Kubernetes v1.29 (latest stable release) selected by Anuket:

Table 42Software versions

Software	Version
Functest	v1.29
kube-hunter	0.6.8
kube-bench	v0.6.10

6.11.2.5 Opensource CNF onboarding and testing

Running opensource containerized network functions (CNF) is a key technical solution to ensure that the platforms meet Network Functions Virtualization requirements.

Functest CNF offers 2 test cases which automatically onboard and test Clearwater IMS [174] via kubectl and Helm. It's worth mentioning that this CNF is covered by the upstream tests (see clearwater-live-test [175]).

The following software versions are considered to verify Kubernetes v1.29 (latest stable release) selected by Anuket:

Table 43Software versions

Software	Version
Functest	v1.29
clearwater	release-130
Helm	v3.3.1

6.11.3 Test Cases Traceability to Requirements

The following test case must pass as they are for Reference Conformance:

Table 44Mandory test cases

Container	Test suite	Criteria	Requirements
opnfv/functest-kubernetes-smoke:v1.29	xrally_kubernetes	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	k8s_conformance	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	k8s_conformance_serial	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_api_machinery	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_api_machinery_serial	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_apps	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_apps_serial	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_auth	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_cluster_lifecycle	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_instrumentation	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_network	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_node	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_scheduling_serial	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.29	sig_storage	PASS	Kubernetes API testing

opnfv/functest-kubernetes-smoke:v1.29	sig_storage_serial	PASS	Kubernetes API testing
opnfv/functest-kubernetes-security:v1.29	kube_hunter	PASS	Security testing
opnfv/functest-kubernetes-security:v1.29	kube_bench_master	PASS	Security testing
opnfv/functest-kubernetes-security:v1.29	kube_bench_node	PASS	Security testing
opnfv/functest-kubernetes-benchmarking:v1.29	xrally_kubernetes_full	PASS	Kubernetes API benchmarking
opnfv/functest-kubernetes-benchmarking:v1.29	netperf	PASS	Dataplane benchmarking
opnfv/functest-kubernetes-cnf:v1.29	k8s_vims	PASS	Opensource CNF onboarding and testing
opnfv/functest-kubernetes-cnf:v1.29	helm_vims	PASS	Opensource CNF onboarding and testing

6.11.4 Kubernetes Testing Cookbook

At the time of writing, the CI description file is hosted in Functest and only runs the containers selected here. It will be completed by the next Anuket mandatory test cases and then a new CI description file will be proposed in a shared tree.

Xtesting CI [176] <https://galaxy.ansible.com/collivier/xtesting>__ only requires internet access, GNU/Linux as Operating System and asks for a few dependencies as described in Deploy your own Xtesting CI/CD toolchains [177] <https://github.com/collivier/ansible-role-xtesting>`__:

- python-virtualenv
- git

Please note the next two points depending on the GNU/Linux distributions and the network settings:

- SELinux: you may have to add --system-site-packages when creating the virtualenv (“Aborting, target uses selinux but python bindings (libselinux-python) aren’t installed!”)
- Proxy: you may set your proxy in env for Ansible and in systemd for Docker [178]

To deploy your own CI toolchain running Anuket Compliance:

```
virtualenv functest-kubernetes --system-site-packages
. functest-kubernetes/bin/activate
pip install ansible
ansible-galaxy install collivier.xtesting
ansible-galaxy collection install ansible.posix community.general community.grafana kubern
etes.core community.docker community.postgresql
git clone https://gerrit.opnfv.org/gerrit/functest-kubernetes functest-kubernetes-src
(cd functest-kubernetes-src && git checkout -b stable/v1.29 origin/stable/v1.29)
ansible-playbook functest-kubernetes-src/ansible/site.cntt.yml
```

6.11.4.1 Configure Kubernetes API testing

Place the kubeconfig configuration file corresponding to the Kubernetes cluster under test in the following location on the machine running the cookbook:

/home/opnfv/functest-kubernetes/config

6.11.4.2 Run Kubernetes conformance suite

Open http://127.0.0.1:8080/job/functest-kubernetes-v1.29-daily/ in a web browser, login as admin/admin and click on “Build with Parameters” (keep the default values).

If the System under test (SUT) is Anuket compliant, a link to the full archive containing all test results and artifacts will be printed in functest-kubernetes-v1.29-zip’s console. Be free to download it and then to send it to any reviewer committee.

To clean your working dir:

deactivate
rm -rf functest-kubernetes-src functest-kubernetes

7 Gaps, Innovation, and Development

7.1 Introduction to Gaps, Innovation, and Development

Functional gaps exist between available open source technology and this Reference Architecture's (RA) or Reference Model's (RM) requirements. This chapter details these gaps and proposes solutions. This may identify and target various "upstream" community projects for development efforts.

7.1.1 Gap Template

Related requirements: List requirement references (e.g., abc.xyz.00) from RA2 or RM addressed by this gap.

Baseline project: Identify the upstream project (e.g., *Kubernetes*) where the gap exists. If none, state "none".

7.1.2 Container Run-Time Interfaces Towards NFVI Resources

This describes the southbound interface from the container to IaaS-provided infrastructure resources; e.g., the network interface type presented to a running container.

7.1.3 Multitenancy and Workload Isolation with Kubernetes

Related requirements: e.man.004, sec.ci.008, sec.wl.005, sec.wl.006

Baseline project: *Kubernetes*

Gap description: Kubernetes currently lacks robust multitenancy, preventing secure infrastructure resource sharing among untrusted tenants. This poses security risks when separating workloads by category (e.g., production vs. non-production). Tenant network segmentation is also needed, while maintaining central administration. Beyond security, this creates operational challenges. Deploying many CNFs in one cluster risks version and configuration conflicts, and software lifecycle management problems. Lack of isolation increases cascading failure risk.

Proposals & Resolution: Kubernetes isn't a single-cluster solution. Industry case studies (Alibaba Cloud Blog: What Can We Learn from Twitter's Move to Kubernetes [179], YouTube: Kubernetes Failure Stories, or: How to Crash Your Cluster - Henning Jacobs [180], CNCF Blog: Demystifying Kubernetes as a service – How Alibaba cloud manages 10,000s of Kubernetes clusters [181]) and CNCF surveys show growing multi-cluster deployments within organizations. A multi-cluster approach addresses security and software lifecycle management challenges.

Without in-cluster multitenancy, separate clusters provide necessary CNF isolation (based on vendor, environment, category, or independent lifecycles). Co-locating similar CNFs allows for simultaneous upgrades, while separate clusters accommodate independent upgrades for CNFs with different versions, configurations, and dependencies.

However, managing numerous clusters at scale poses significant operational challenges if done manually. Operators should carefully consider their multi-cluster management strategy, including application management.

7.1.4 Kubernetes as a VM-based VNF Orchestrator

Related requirements: None.

Baseline project: *Kubernetes*, *Kubevirt*

Gap description: Kubernetes and a CRI-compliant runtime should support VNF execution without altering VNF architecture or deployment artifacts.

7.1.5 Native Multiple Network Interfaces on Pods

Related requirements: Virtual Network Interface Specifications (Chapter 4 of Reference Model for Cloud Infrastructure (RM) [1])

Baseline project: *Kubernetes*

Gap description: Kubernetes lacks native support for multiple Pod interfaces, requiring a CNI multiplexer (like GitHub: Multus-CNI [182]). Network service implementation (Network Policies, Ingress, Egress, Load Balancers) depends on the multiplexer and its CNI plugins, leading to inconsistency.

Status: A KEP (Google Docs: KEP: MultiNetwork podNetwork object [183]) aims to natively support multiple Pod interfaces.

7.1.6 Dynamic Network Management

Related requirements: inf.ntw.03 (chapters/chapter02:kubernetes architecture requirements)

Baseline project: *Kubernetes*

Gap description: Kubernetes lacks an API for network service management (e.g., VPNs). A CNI plugin (GitHub: Multus-CNI [182]) or integration with SDN controllers (using NetConf, etc.) is needed to provide APIs for network services and connect VPNs (e.g., L3VPN) to CNFs on demand.

7.1.7 Control Plane Efficiency

Related requirements: None

Baseline project: *Kubernetes*

Gap description: Multi-site/availability zone deployments often utilize multiple Kubernetes clusters for security, multitenancy, fault tolerance, resilience, and latency. This creates Kubernetes control plane node overhead. More efficient multi-cluster operation is needed to meet non-functional requirements.

7.1.8 Interoperability with VRF-based Networking

Related requirements: None

Baseline project: *Kubernetes*

Gap description: L3 VRFs/VPNs are commonly used for traffic separation (signaling, charging, LI, O&M). CNFs must interoperate with existing network elements, requiring Kubernetes Pod connection to L3 VPNs (currently only possible via Multus). However, network orchestration (connecting the interface to a gateway router terminating the L3 VPN) is not handled by Kubernetes, and lacks a production-grade open source solution.

Note

While possible with IaaS, this creates an undesirable dependency between Kubernetes workload orchestration and IaaS infrastructure orchestration.

7.1.9 Hardware Topology-Aware Huge Pages

Related requirements: infra.com.cfg.004 and infra.com.cfg.002 (Virtual Compute Profiles, Chapter 5 of Reference Model for Cloud Infrastructure (RM) [1]).

Baseline project: *Kubernetes*

Gap description: The Memory Manager (alpha feature in v1.21) is addressed in Management of Memory and Huge Pages Resources.

7.1.10 User Namespaces in Kubernetes

Related requirements: e.man.004 (Cloud Infrastructure Management Capabilities, Chapter 4 of Reference Model for Cloud Infrastructure (RM) [1]), inf.ntw.03 (Platform and Access Requirements, Chapter 2 of Reference Architecture (RA1) for OpenStack based cloud infrastructure [67])

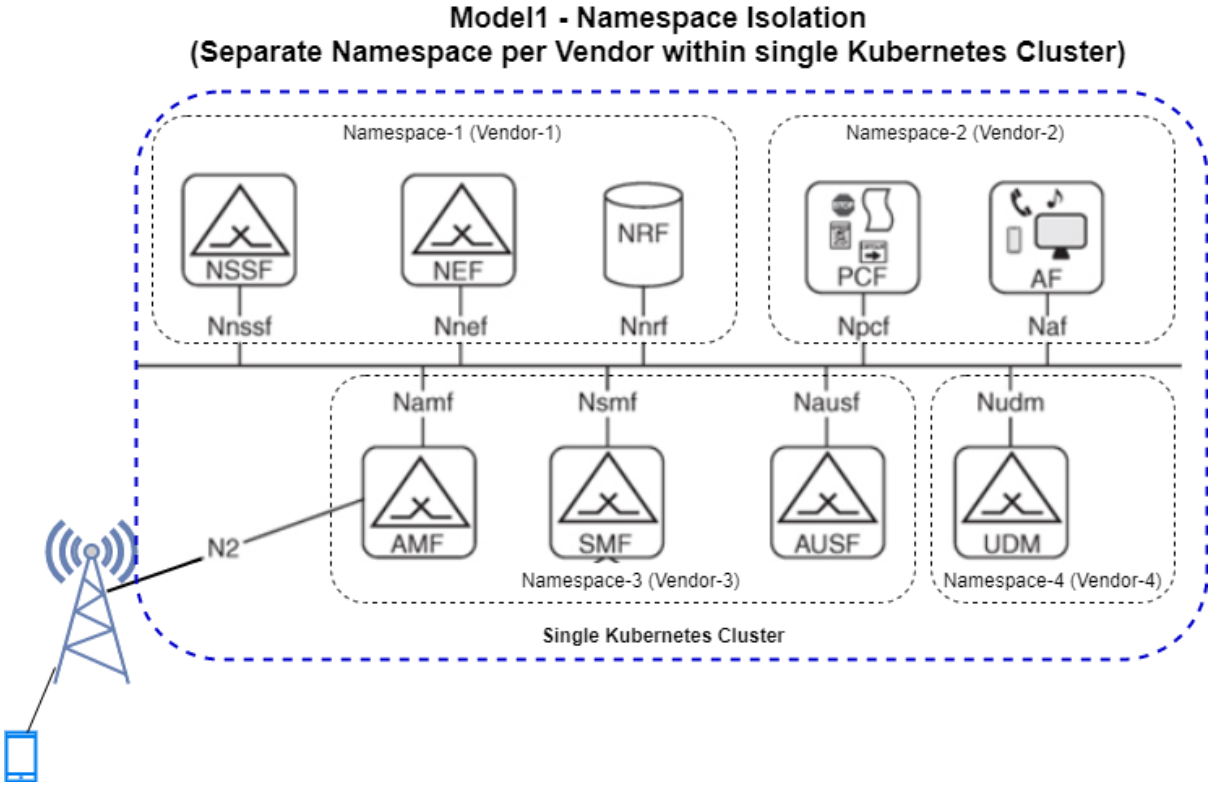
Baseline project: *Kubernetes*

Gap description: Kubernetes lacks namespace-scoped UIDs. CNFs requiring system privileges must run in privileged mode or use random system UIDs. Random UIDs cause errors when setting kernel capabilities (e.g., VLAN trunking) or sharing data via persistent storage. Privileged mode is insecure; random UIDs are error-prone. Proper user namespaces (alpha in Kubernetes 1.25 Kubernetes Docs: User Namespaces [184], KEP KEP-127: Support User Namespaces in stateless pods [185]) are needed.

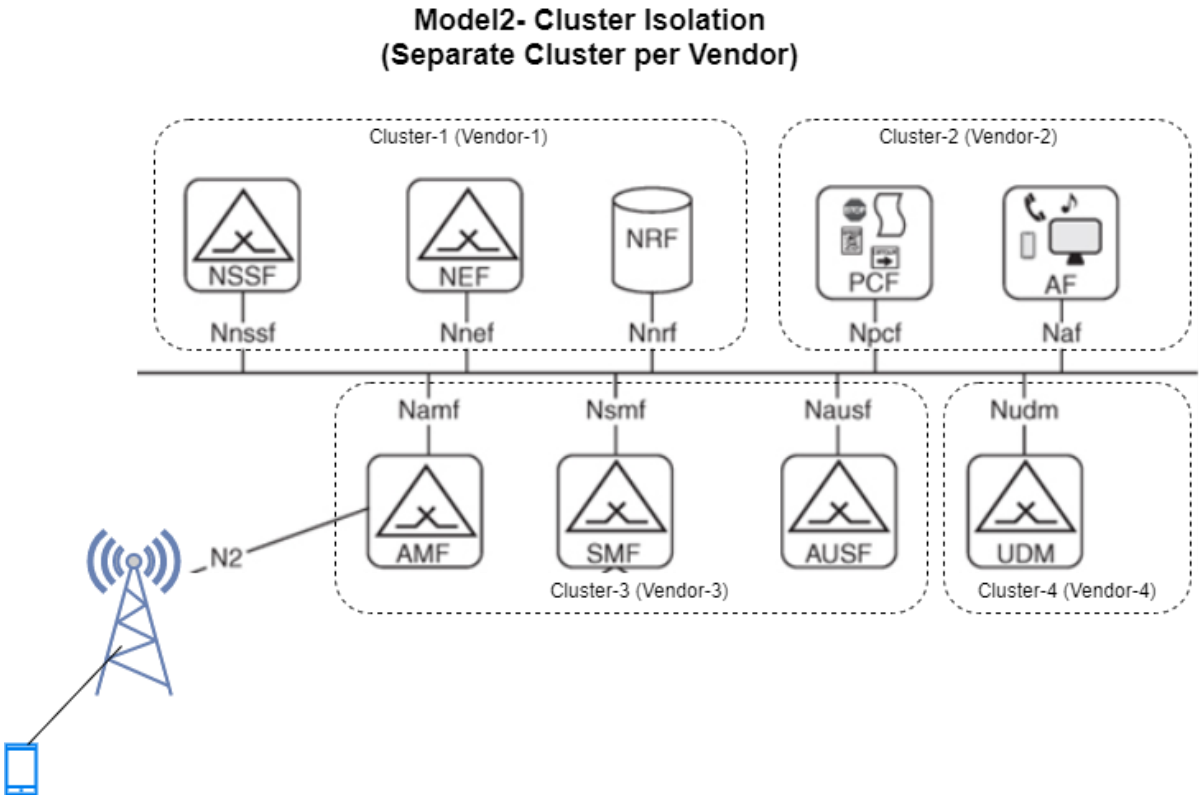
8 Appendix A - Guidance For workload isolation (Multitenancy) with Kubernetes for application vendors

8.1 Overview of Appendix A

Problem statement: A single Kubernetes Cluster does not provide hard multitenancy* by design. Within a Cluster, the Kubernetes Namespace is a mechanism to provide Soft isolation multitenancy. A Kubernetes Namespace does provide isolation by means of role-based access control (RBAC), Resource Isolation, and Network Policy. However, they are still within the same trust domain and a potential breach of the Cluster Admin Role could have serious ramifications across the entire Cluster and all its Kubernetes Namespaces. Therefore, it is necessary to define various use cases or ways to build Multitenancy Deployment Models and define the Best Practices to secure each Model. The Kubernetes Namespace is a logical representation of namespace (boundary for resources) within the Kubernetes Cluster. This is different from the Wikipedia: Linux Namespaces [186], which are defined at the operating system kernel level.



Network service



Cluster isolation

Use cases:

1. Two CNFs which are in the same trust domain (for example, they are from the same vendor) are running in a container infrastructure.
2. Two CNFs which are in different trust domains (for example, they are from different vendors) are running in a container infrastructure.

8.2 Solution areas

The scope is to identify the solution area which is needed to secure the CNF workloads. Securing the platform might happen as part of it. However, it is not directly the focus or objective here.

1. Isolation of platform and tenant deployment based on the solution model.
2. Separate CICD, manifest, and image repository for platform and tenants.
3. Isolation of networking per tenant.
4. Securing the CNF workload network traffic using a network policy and service mesh.
5. RBAC rules and secrets management for each tenant.
6. Separate isolated view of logging, monitoring, alerting, and tracing per tenant.

8.3 Multitenancy Models

Solution models :

1. **Soft Multitenancy**: Separate Kubernetes Namespace per tenant within a single Kubernetes cluster. The same Kubernetes Cluster and its control plane are shared between multiple tenants.

2. **Hard Multitenancy**: Separate Kubernetes cluster per tenant. The Kubernetes clusters can be created using baremetal nodes or virtual machines, either in the private cloud or the public cloud. The workloads do not share the same resources, and the clusters are isolated.

8.3.1 Soft multitenancy with Kubernetes Namespaces per tenant

Soft multitenancy with namespaces per tenant can be implemented, resulting in a multitenant cluster, in which multiple trusted workloads share a cluster and its control plane. This is mainly recommended to reduce management overheads when the tenants belong to the same trust domain, and have the same cluster configuration requirements (including release, add-ons, and so on).

The tenants share the cluster control plane and API, including all add-ons, extensions, CNIs, CSIs, and any custom resources and controllers.

To manage access control, the Kubernetes RBAC must be configured with rules to allow specific tenants to access only the objects within their own namespace, using a Role Object to group the resources within a namespace, and a RoleBinding Object to assign it to a user or a group of users within a namespace.

To prevent or allow network traffic between Pods to belong to different namespaces, NetworkPolicy must also be created.

Resource quotas enable the cluster administrator to allocate CPU and memory to each namespace, limiting the amount of resources the objects belonging to that namespace can consume. This may be configured, in order to ensure that all tenants use no more than the resources that are assigned to them.

By default, the Kubernetes scheduler runs pods belonging to any namespace on any cluster node. If it is required that pods from different tenants are run on different hosts, then affinity rules should be created by using the desired Node Labels on the pod definition. Alternatively, node taints can be used to reserve certain nodes for a predefined tenant.

8.3.2 Hard multitenancy with dedicated Kubernetes clusters per tenant

When tenants do not belong to the same trust domain, or the requirements on the cluster setup and configuration are irreconcilable, hard multitenancy must be implemented by creating multiple Kubernetes clusters for each tenant or group of compatible tenants.

All the default design decisions for Kubernetes clusters apply in this case. No special segregation or capacity management needs to be set up within the clusters.

From an operational perspective, the increased computational and operational overheads and the Cluster LCM (including cluster provisioning automation) are the most impactful aspects.

9 Document Management

A.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
1.0	28/11/2023	New PRD (CR1001).	OIG	Riccardo Gasparetto Stori / Vodafone Gergely Csatary /Nokia Petar Torre / Intel
2.0	xx/01/2025	Major CR Update to Anuket Pieman and Quinnipiac releases	OIG	Riccardo Gasparetto Stori / Vodafone Gergely Csatary /Nokia Petar Torre / Intel Cédric Ollivier / Orange

A.2 Other Information

Type	Description
Document Owner	Networks Group
Editor / Company	Riccardo Gasparetto Stori / Vodafone Gergely Csatary / Nokia Petar Torre / Intel

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.