ETSI GR NFV-REL 014 V5.1.1 (2023-10)

Network Functions Virtualisation (NFV) Release 5;
Reliability;
Report on evaluating reliability for cloud-native VNFs

Reference

DGR/NFV-REL014

Keywords

cloud-native, reliability

Contents

Intellectual Property Rights

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document studies the reliability evaluation for cloud-native VNFs (as defined in ETSI GS NFV-EVE 011 [i.2]). It identifies key use cases (containerized VNF software modification, containerized VNF scaling out/in) of cloud-native VNF lifecycle management which may impact reliability. In these use cases, possible new functionalities to support the resiliency assurance of these VNF operations are discussed, using Kubernetes® as an example containerization technology. Then possible solutions are presented related to the new functionalities, together with the potential architectural options.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI GR NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[i.2] ETSI GS NFV-EVE 011 "Network Functions Virtualisation (NFV) Release 3; Virtualised Network Function; Specification of the Classification of Cloud Native VNF implementations".

[i.3] ETSI GS NFV-SEC 023: "Network Functions Virtualisation (NFV) Release 4; Security; Container Security Specification".

[i.4] ETSI GR NFV-IFA 029: "Network Functions Virtualisation (NFV) Release 3; Architecture; Report on the Enhancements of the NFV architecture towards "Cloud-native" and "PaaS"".

[i.5] ETSI GS NFV-IFA 010: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Functional requirements specification".

[i.6] ETSI GS NFV-IFA 040: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Requirements for service interfaces and object model for OS container management and orchestration specification".

[i.7] ETSI GS NFV-IFA 036: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Requirements for service interfaces and object model for container cluster management and orchestration specification".

[i.8] Kubernetes® document.

[i.9] Kubernetes® API conventions document.

[i.10] ETSI GS NFV 006: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Architectural Framework Specification".

[i.11] ETSI GS NFV-REL 003: " Network Functions Virtualisation (NFV); Reliability; Report on Models and Features for End-to-End Reliability".

[i.12] ETSI GS NFV-SOL 001: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; NFV descriptors based on TOSCA specification".

[i.13] ETSI GR NFV-IFA 041: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Report on enabling autonomous management in NFV-MANO".

[i.14] ETSI GS NFV-IFA 027: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Performance Measurements Specification".

[i.15] ETSI GS NFV-SOL 018: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; Profiling specification of protocol and data model solutions for OS Container management and orchestration".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI GR NFV 003 [i.1] and the following apply:

NOTE: A term defined in the present document takes precedence over the definition of the same term, if any, in ETSI GR NFV 003 [i.1].

cloud-native VNF: VNF designed to be deployed and managed in a cloud computing environment for efficient operation

NOTE: The present document assumes that cloud-native VNFs have (but are not limited to) the following characteristics:

- dynamic (e.g. with frequent changes);
- scalable;
- fine-granular (e.g. composed of microservices, containerized).

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GR NFV 003 [i.1] apply.

4 Overview and background

4.1 General

The telecom industry is experiencing a transformation towards cloud-native. Cloud-native VNFs may use technologies such as containerized functions, micro-service based architecture, self-management, scalability, etc. The management of VNFs following cloud-native principles is bringing profound changes to the operations and maintenance of telecom cloud-based networks, e.g. the combination of DevOps and Cloud increases the software delivery and efficiency. These new changes introduce new challenges on managing cloud-native VNFs, especially for the non-functional aspects like performance, reliability and security.

Reliability for cloud-native VNFs is really challenging because of their highly dynamic nature. Thus, highly dynamic management is needed due to the nature of cloud-native VNFs. In the scope of ISG NFV, ETSI GS NFV-EVE 011 [i.2] specifies non-functional aspects for cloud-native VNFs, e.g. resiliency, scaling and composition. ETSI GS NFV-SEC 023 [i.3] specifies the security and hardening requirements for VNFs running in a containerized environment. ETSI GR NFV-IFA 029 [i.4] investigates the use cases for application of cloud-native design principles, but it lacks the use cases analysis and has no recommendations on NFV-MANO functional enhancement derived from the use cases.

The present document focuses on the study of reliability aspects for supporting the management of cloud-native VNFs. Clause 5 introduces some cloud-native configuration capabilities and metrics related to reliability. Clause 6 studies a number of use cases for the purpose of deriving corresponding criteria and their associated configuration capabilities and metrics to evaluate the reliability for cloud-native VNFs. Clause 7 further elaborates on the identified criteria and their associated metrics of reliability evaluation. Finally, Clause 8 summarizes the recommendations for normative work in the future.

4.2 Containerized VNFs

In a cloud-native VNF environment, OS-container becomes the recommended technology for the infrastructure services in support of the VNFs, even though VM-based virtualisation is still an option for fulfilling cloud-native objectives. The introduction of OS-containers has an impact on the NFV-MANO architecture, as specified in ETSI GS NFV-IFA 010 [i.5] and ETSI GS NFV-IFA 040 [i.6], namely the introduction of new managed objects and a management function related to OS-container management and orchestration, i.e. MCIO and CISM. A Managed Container Infrastructure Object (MCIO) is a managed object representing the desired and actual state of a containerized workload for the OS-container management and orchestration. As specified in ETSI GS NFV-IFA 040 [i.6], an MCIO requesting compute/storage resources can be mapped to a VNFC.

Kubernetes®, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. ETSI GS NFV-SOL 018 [i.15] profiles the Kubernetes® API as NFV protocol and data model solution for OS container management and orchestration. As defined in ETSI GR NFV-IFA 029 [i.4] and ETSI GS NFV-IFA 040 [i.6], MCIO could be realized as Pod, Deployment, StatefulSet, etc. in Kubernetes®.

Figure 4.2-1 shows the OS-container infrastructure service management architecture, as described in ETSI GS NFV 006 [i.10]. The Container Infrastructure Service (CIS) is responsible for providing the virtualised infrastructure as OS-containers. The Container Infrastructure Service Management (CISM) is responsible for the management of containerized workloads as MCIOs running in the CIS. The Container Cluster Management (CCM) is responsible for the management of CIS clusters.

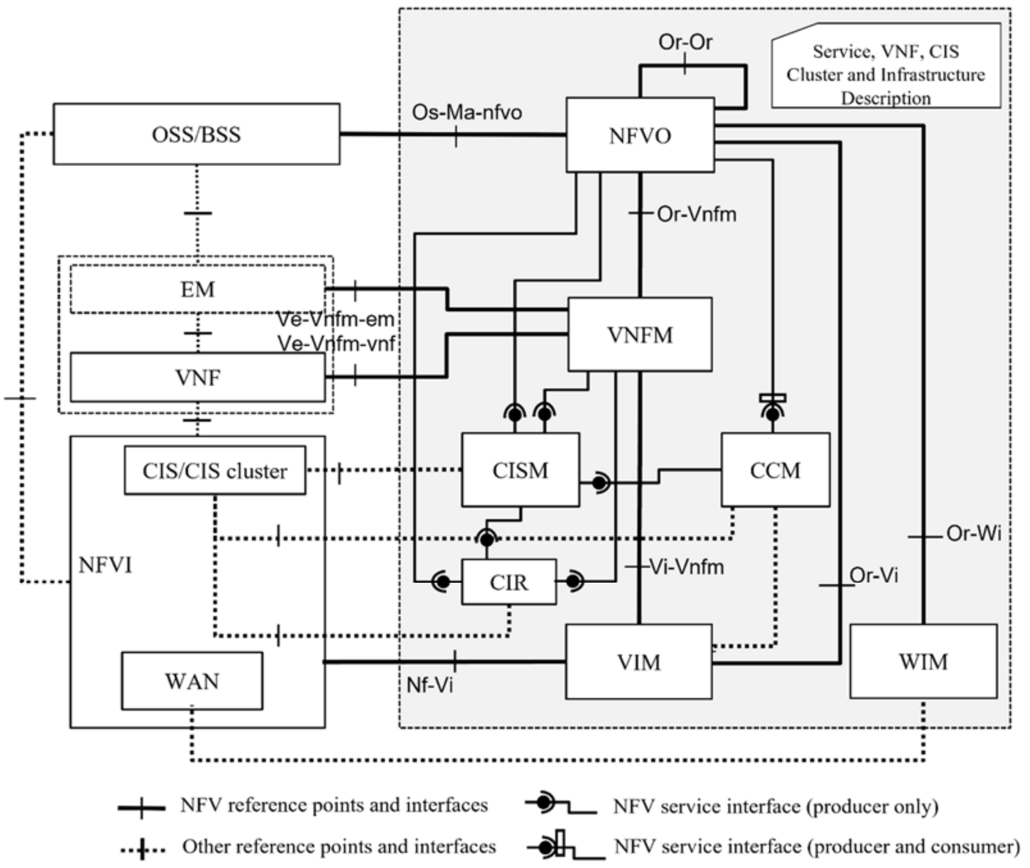
Or-Or
Service, VNF, CIS Cluster and Infrastructure Description
OSS/BSS
Os-Ma-nfvo
NFVO
Or-Vnfm
EM
Ve-Vnfm-em
Ve-Vnfm-vnf
VNFM
VNF
CIS/CIS cluster
CISM
CCM
NFVI
Or-Wi
CIR
Vi-Vnfm
Or-Vi
WAN
Nf-Vi
VIM
WIM
NFV reference points and interfaces
Other reference points and interfaces
NFV service interface (producer only)
NFV service interface (producer and consumer)

Figure 4.2-1: Container-based NFV architectural framework

5 Cloud-native configuration capabilities and metrics related to reliability

5.1 Kubernetes® objects

Kubernetes® objects are persistent managed objects representing the managed container cluster and its different resources such as pods. Kubernetes® objects can describe among others which containerized applications are running (and on which nodes), the resources available to those applications, and the policies applicable to those applications [i.8]. In the context of NFV, Kubernetes objects are MCIOs.

Most importantly, a Kubernetes® object includes two fields: the spec field and the status field [i.9]. The spec field is set by the user and characterizes the desired status of the entity represented by the Kubernetes® object. The status field shows the current status of the entity represented by the Kubernetes® object as supplied and updated by Kubernetes®. Kubernetes® continually and actively manages the actual status of each entity to match its desired status.

5.2 Management of clusters

5.2.1 Introduction

Current industry solutions for OS container management expect that a cluster of machines (running either in virtual machines or on bare-metal servers) is provided for their use. In the context of NFV, a cluster is called a Container Infrastructure Service (CIS) cluster and is composed of one or multiple CIS cluster nodes. A CIS cluster node, which can be realized as a VM or a bare-metal server, is a compute resource that runs a CIS instance or a CISM instance, or both.

The CCM (CIS Cluster Management) is responsible for the lifecycle management and FCAPS management of the CIS cluster. The CCM consumer can define the essential cluster information, including the description of the CIS cluster nodes, the placement constraints and the affinity or anti-affinity rules, etc., in the CIS Cluster Descriptor (CCD) that is interpreted by the CCM.

The concept of CIS cluster and the functionalities of CCM are detailed in ETSI GS NFV-IFA 036 [i.7].

5.2.2 Relevant configuration capabilities

ETSI GS NFV-IFA 036 [i.7] defines the requirements for CIS cluster management. CCM is responsible for lifecycle management of the CIS cluster, including applying changes to the CIS cluster configuration, scaling the CIS cluster, and modification of CIS cluster software.

CIS cluster configuration attributes include CIS cluster nodes to be used in the CIS cluster, number of CIS cluster nodes, scaling characteristics, placement constraints, cluster networking, and cluster storage. For more details, refer to clause 4.2.4 of ETSI GS NFV-IFA 036 [i.7].

CCM can construct CISM with high availability. For details of how CISM high availability can be achieved, refer to clause 4.2.12 of ETSI GS NFV-IFA 036 [i.7].

5.2.3 Relevant metrics

CCM reports information related to the CIS cluster configuration and CIS cluster status as defined in clause 4.2.4 of ETSI GS NFV-IFA 036 [i.7].

CCM provides performance measurements for CIS cluster nodes, CIS cluster storage, and CIS cluster nodes network, but not at the overall CIS cluster level.

5.3 Management of pods

5.3.1 Overview

Pods are the most basic deployable resources in Kubernetes® which are represented by Kubernetes® objects. Pods contain one or more containers, such as Docker™ containers. When a pod runs multiple containers, the containers share the pod's resources. Pods run on nodes organized in a certain container cluster.

Pods follow a defined lifecycle, starting in the **Pending** phase, moving through the **Running** phase if at least one of its containers is running, or in the process of starting or restarting. Pods complete their lifecycle through either the **Succeeded** or **Failed** phases depending on whether any container in the pod has terminated in a failure. A pod will spend most of its operational life in the Running phase.

In a pod, Kubernetes® tracks the container(s) status and determines what action(s) to take to keep the pod's actual status as desired. Kubernetes® is able to restart containers to handle some failures, e.g. Out-Of-Memory (OOM) failure when a container exceeds its resource limit. Container failures, which cannot be resolved by restart, are escalated to the pod level and cause the pod to be terminated in the Failed phase.

For stateless containerized applications, Kubernetes® provides the **Deployment** object to realize declarative configuration for a set of pods. Users can create a Deployment to roll out a group of identical pods, scale the Deployment to increase or decrease the number of its pods, and update the Deployment which means changing this Deployment's pod template by updating the pods' metadata and spec configurations (see clause 5.3.2), e.g. pods' labels or container images.

For stateful containerized applications, the Kubernetes® object **StatefulSet** is used to manage the roll-out and scaling of a set of pods. A StatefulSet provides guarantees about the ordering and uniqueness of these pods. Like a Deployment, a StatefulSet contains pods that are based on the same container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of its pods. Even though these pods are created from the same spec, they are not interchangeable: each has a persistent identifier that it maintains even across rescheduling [i.8].

Pods in a StatefulSet have a unique identity that is comprised of an ordinal and a stable network identity, and a stable storage. When pods are being deployed, they are created sequentially, which defines a succession. When pods are deleted, they are terminated in reverse order. Pods of a StatefulSet can be updated in a rolling fashion, which means an automated rolling update of their containers, labels, resource requests/limits, and annotations.

The Kubernetes® objects expose some configurable attributes to request resources for running pods and their desired limits. These attributes allow capacity planning, assessment of current or historical scheduling limits, quick identification of workloads that cannot be scheduled due to a lack of resources, and comparison of actual usage to the pod's request.

For more details on pods management, refer to [i.8] and [i.9].

5.3.2 Existing configuration capabilities

The Kubernetes® object representing a pod includes the following configuration capabilities:

- **Metadata:** which contains the identification and description of the pod. Commonly used metadata attributes for a pod are:
    - **Name**.
    - **Namespace:** which configures the namespace to which the pod belongs.
    - **Labels:** which lists the pod's custom labels.
- **Spec:** which is a detailed configuration of the pod's various resources and handling options. The key attributes for the pod's resource configuration are the following:
    - **Containers:** which is used to configure the containers of the pod.
    - **NodeName:** name of the node on which Kubernetes® can schedule this pod.
    - **NodeSelector:** which defines the node labels. Kubernetes® has to schedule this pod to a node with these labels.
    - **Volumes:** which provides the storage volume information of the pod.
    - **RestartPolicy:** which configures the strategy for this pod in case of pod failure.

Kubernetes® provides the capability to configure a single pod; however, it is more common to deploy a set of identical pods as a Deployment or a StatefulSet using pod templates.

For managing a set of stateless pods, the Kubernetes® object Deployment includes the following configuration capabilities:

- **Metadata:** which contains the identification and description of the Deployment. Commonly used metadata attributes for a Deployment are its name and labels.
- **Spec:** which is a detailed configuration of the Deployment's various resources. The key attributes for the Deployment's resource configuration are the following:
    - **Replicas:** which is used to configure how many pod replicas are needed.
    - **Selector:** which defines how the Deployment controller finds which pods it manages. For example, the user can select a label that is defined in the pod template, then the Deployment controller will manage pods with this label.
    - **Template:** which provides template for pods in the Deployment.
    - **Strategy:** which specifies the strategy used to replace pods of the old template by new ones when the Deployment is upgraded. It has two options "**Recreate**" or "**RollingUpdate**".

For a containerized application consisting of a pool of stateless pods, it can be configured in the sub-attribute **maxUnavailable** and **maxSurge** under the RollingUpdate strategy to control the rolling software modification process in Kubernetes®. The configurable attribute maxUnavailable specifies the maximum number of pods that can be unavailable during the software modification process. The value can be an absolute number (for example, 5) or a percentage of desired pods (for example, 10 %), ensuring that the total number of pods available at all times during the software modification process can meet the least availability need of the application. The configurable attribute maxSurge specifies the maximum number of pods that can be created over the desired number of pods. The value can be an absolute number (for example, 5) or a percentage of desired pods (for example, 10 %), ensuring that the total number of pods running at any time during the software modification won't go over the threshold to affect the system performance or occupy too many virtualised resources.

- **ProgressDeadlineSeconds:** which configures the number of seconds Kubernetes® waits for the Deployment to progress before the system reports back that the Deployment has failed.

For managing a set of stateful pods, the Kubernetes® object StatefulSet includes the following configuration capabilities:

- **Metadata:** which contains the identification and description of the StatefulSet, e.g. the name and labels of the StatefulSet.
- **Spec:** which is a detailed configuration of the StatefulSet's various resources. The key attributes for the StatefulSet's resource configuration are the following:
  - **Replicas:** which is used to configure how many pod replicas are needed.
  - **Selector:** which defines how the StatefulSet controller finds which pods it manages. For example, the user can select a label that is defined in the pod template, then the StatefulSet controller will manage pods with this label.
  - **Template:** which provides template for pods in the StatefulSet, including the information for containers' images, ports and volume mounts in the pod.
  - **ServiceName:** which defines the **service** this StatefulSet corresponds to.
  - **VolumeClaimTemplate:** which configures a Persistent Volume Claim (PVC) template so that each pod in the StatefulSet will have its persistent volume stable storage.
  - **PodManagementPolicy:** which allows users to relax the StatefulSet's management of pod ordering while preserving its pod uniqueness and identity guarantees. By default, in a StatefulSet, before a scaling operation is applied, all the predecessors of the new pod need to be in the Running phase; or before a pod is terminated, all of its successors need to be terminated (Succeeded or Failed).
  - **UpdateStrategy:** which configures or disables automated rolling updates for containers, labels, resource request/limits, and annotations of the pods of a StatefulSet. It has two options "**OnDelete**" or "**RollingUpdate**". When the value is "OnDelete", Kubernetes® will not automatically update the pods in the StatefulSet. Users manually delete old pods so that the controller creates new pods that reflect modifications made to the template. The type "RollingUpdate" allows all or a partition of pods to be automatically updated when the StatefulSet's template is updated.

    For the stateful containerized application software modification, the maxUnavailable can be configured for the StatefulSet similar to the Deployment. Besides the maxUnavailable, the sub-attribute **partition** under the RollingUpdate strategy can also be configured to define a set of pods that may be modified and a set of pods that are not modified in this software modification process. This configuration can be useful if it is decided to roll out a subset for software modification validation (known in Kubernetes® as a canary upgrade), or perform a phased roll-out for containerized application software modification depending on the likelihood of meeting the availability requirements of the application.

For more details of the existing pod configurable attributes in Kubernetes®, refer to Annex A.

5.3.3 Existing metrics

For pods management in Kubernetes®, the status of pods summarizes the current situation of the pods in a container cluster. Fields in status for pods are the most recent observations for pods' situation, but they may contain information such as the results of allocations or similar operations which are executed in response to the Kubernetes® object's spec. The key metrics showing the Kubernetes® pod's status are as follows:

- **Phase:** showing which phase this pod is currently in.
- **Conditions:** showing more detailed information per condition type on this pod's readiness:
  - **Type:** there are four condition types: "**PodScheduled**", "**Initialized**", "**ContainersReady**" and "**Ready**". "PodScheduled" shows if the pod has been scheduled to a node. "Initialized" shows if all init containers in this pod have been started successfully. "ContainersReady" shows if all containers in the pod are ready. "Ready" shows if the pod can provide its service. For all four condition types, the status value can be "True", "False" or "Unknown".
  - **LastProbeTime:** the timestamp of the last pod condition detection.
  - **LastTransitionTime:** the timestamp of the last condition transition from one status value to another.
  - **Reason:** a machine-readable reason for the last condition transition.
  - **Message:** a human-readable detailed description of the last condition transition.

Besides, in Kubernetes®, CPU usage and memory usage are two key performance metrics for pod's computing resources:

- **Pod CPU usage:** CPU is reported as the average core usage measured in CPU units. One CPU, in Kubernetes®, means one vCPU/Core for cloud providers, or one hyper-thread on bare-metal processors. The existing metrics include the number of cores and percentage (actual usage compared to the max limit) of CPU usage.
- **Pod memory usage:** Memory is reported as the working set, measured in bytes, at the instant the metric was collected. The existing metrics include the number of bytes and percentage (actual usage compared to the max limit) of memory usage.

For monitoring a set of containerized applications, the key metrics showing the Kubernetes® Deployment and StatefulSet's status are as follows:

- **Replicas:** showing how many pod replicas are in this set of pods.
- **AvailableReplicas:** showing the number of available pod replicas in this set of pods. Available pods are in the phase "Running" and the condition type "Ready".
- **UnavailableReplicas:** showing the number of unavailable pod replicas in this set of pods. Available pods are both in the phase "Running" and in the condition "Ready". All other pods are considered unavailable.
- **Conditions:** showing detailed information per condition type on the readiness of this set of pods:
  - **Type:** there are two condition types: "**Progressing**" and "**Complete**". "Progressing" shows if the Deployment or StatefulSet is now creating pods, scaling or has pods available. "Complete" shows if all of the pods in this set have been updated to the latest version specified by the user, if all new replicas are available and all old replicas have been terminated. For both condition types, the status value can be "True", "False", or "Unknown".
  - **LastTransitionTime:** the timestamp of the last condition transition from one status value to another
  - **Reason:** a machine-readable reason for the last condition transition, e.g. "ProgressDeadlineExceed" for a Deployment.
  - **Message:** a human-readable detailed description of the last condition transition.

For more details of the existing pod metrics in Kubernetes®, refer to Annex A.

6 Use cases

6.1 Introduction

This clause provides use cases related to evaluating reliability for cloud-native VNF software modification and scaling.

6.2 Cloud-native VNF software modification

6.2.1 Overview

In ETSI NFV concepts, the software modification process includes software upgrade, software update, software fallback, and software rollback [i.1]. Cloud-native VNFs are expected to be equipped with appropriate mechanisms to monitor the VNF health and support the general VNF lifecycle. The ability to estimate reliability and availability is very useful for the successful software modification of cloud-native VNFs. For example, the software modification process for VNFs may take different paths depending on the likelihood of meeting the availability requirements of the VNFs. In clause 6.2, the VNF availability evaluation for containerized and VNF software modification is discussed.

Containerized implementation does not bring any inherent change to the overall logic of VNF software modification; however, the CISM (as implemented by Kubernetes®) provides more automatic capabilities for containerized VNF software modification. Similar to the case of VNFs deployed as virtual machines, containerized VNF software modification may be initiated by EM or NFVO (the latter NFVO case assumes that the NFVO has received an *UpdateNS* operation with *updateType = ChangeVnfPkg* from the OSS) using a *ChangeCurrentVnfPackage* operation. As a result of this operation, VNFM requests the update of Kubernetes® managed objects (e.g. *Deployment*, *StatefulSet*) which implies that the new software will be deployed in new containers. Before the request, it would be useful to estimate if the current status of containerized applications managed by Kubernetes® and their configuration would impact the VNF availability during the software modification, and re-configure Kubernetes® (using for example *replicas*, *updateStrategy*) if necessary. Kubernetes® can then follow the configured strategy for the containerized software modification process.

For both stateful and stateless containerized VNFs, Kubernetes® provides several autonomous mechanisms for software modification. However, Kubernetes® itself doesn't have the ability to do the VNF availability calculation without enhancement or external assistance. Therefore, a functional entity needs to evaluate the availability for containerized VNFs before the VNF software modification process is triggered. Clause 7 discusses on where this functional entity could be placed.

Beyond traditional VNF related metrics used for VNF generic OAM, metrics reflecting the container infrastructure status are also useful to be observed for this VNF availability calculation and estimation.

6.2.2 MCIO availability evaluation

The cloud-native technology is supposed to provide container infrastructure status observability for the applications running on it. In practice, container infrastructure status observability is the ability to understand the system using its outputs like metric monitoring, event logging, and tracing. Among these outputs, metric monitoring allows the management system to have a general understanding of the container infrastructure service health, event logging can provide real-time event logs generated during the execution of the applications that can explain the running status of the observed object system in detail, while tracing is request-oriented with abnormal point information useful in the case of debugging for containerized VNF software modification.

As one of the 3 key aspects (metric monitoring, event logging and tracing) in the cloud-native observability practice, container infrastructure related metrics are the focus of the present use case for availability evaluation for containerized VNF software modification. As mentioned in clause 5, Kubernetes® provides a set of pod and cluster metrics for monitoring the status of containerized implementation.

Beyond generic metrics which Kubernetes® gathered from its own metrics server, custom metrics can also be monitored by another server, an aggregated custom metrics server for carrier-grade telco cloud requirements, e.g. the network flow rate for different pods inside a certain Container Infrastructure Service (CIS) instance (also known as a Kubernetes® node). These container infrastructure observability improvements, especially the Kubernetes® pod related metrics, can help stakeholders to ensure the availability for containerized VNFCs.

As described in clause 4.2, an MCIO can be mapped to a VNFC and an MCIO instance could be realized as a Pod, a Deployment, or a StatefulSet, etc. in Kubernetes®. The following shows an example of pod availability calculation based on pod metrics.

According to ETSI GS NFV-REL 003 [i.11], the reliability and availability of a complex system such as an NFV deployment can be modelled by breaking it down into its constituent components, of which the reliability and availability are known. For repairable components, this can be expressed using cycles of uninterrupted working intervals (uptime), followed by repair periods after a failure has occurred (downtime). The average length of the first intervals is usually called the Mean Time Between Failures (MTBF), while the average length of the second intervals is the Mean Time To Repair (MTTR). The MTBF and the MTTR of a pod, together with infrastructure availability, can be used to calculate the availability of a single pod as:

$$A_{MCIO} = A_{pod} = A_{Infrastructure} * MTBF_{pod} / (MTBF_{pod} + MTTR_{pod}) \qquad (1)$$

6.2.3 Containerized VNF availability evaluation

As discussed in clause 6.2.2, the availability of a MCIO instance can be measured according to related metrics. The evaluation of the availability of a MCIO instance is performed as the first step for calculating the containerized VNF availability. Based on the availability of a MCIO instance, the robustness of the containerized VNF could be evaluated based on the topology model of MCIOs (e.g. containerized instances of the same VNFC with active-active model). Based on these models, the analysing function can provide the availability estimation of the containerized VNF to give a green light for its software modification.

The following shows an example of containerized VNF-internal redundancy model and VNF availability calculation.

In order to meet the VNF availability requirement, the analysing function needs to continually estimate the availability of a VNF instance. To achieve this, the function responsible for the evaluation needs to have knowledge of the topology of the constituent VNFCs and the redundancy models that are used. The topology of the constituent VNFCs can be found in the VNFD which is used by the VNFM for the software modification.

Even though the methods to gather this information are not standardized, it is possible to give the following guidance on how the availability of a VNF instance can be estimated.

The availability of a VNF which is composed of x VNFCs in series can be calculated as:

$$A_{VNF} = A_{VNFC_1} * A_{VNFC_2} * \dots * A_{VNFC_x} \qquad (2)$$

As described in clause 4.2, an MCIO can be mapped to a VNFC. For a pool of MCIO instances (instances of the same VNFC) with redundancy model RM(), the availability of a pool of n MCIO instances is calculated as:

$$A_{VNFC} = RM(A_{MCIOinstance_1}, A_{MCIOinstance_2}, \dots, A_{MCIOinstance_n}) \qquad (3)$$

For the simplest example pool of MCIO instances with active-active redundancy (where at least one of these MCIO instances is available, this VNFC can be regarded as available), the availability of this VNFC is calculated as:

$$A_{VNFC} = 1 - (1 - A_{MCIOinstance_1}) * (1 - A_{MCIOinstance_2}) * \dots * (1 - A_{MCIOinstance_n}) \qquad (4)$$

6.2.4 Impact of the availability evaluation

Different VNFs have different availability requirements to fulfil. For each containerized VNF, the CISM will continuously monitor the CaaS metrics as input for availability calculation. Based on this input, the analysing function is supposed to perform the VNF availability evaluation when the software modification is initiated by EM/NFVO. The software modification can be performed at a certain time when the analysing function confirms that the VNF availability requirements can be met during the software modification. If the calculation finds that the VNF availability requirements cannot be met, CISM needs to take management actions accordingly (e.g. allocate more resources) to reach these requirements. According to the analysis output, the analysing function can provide Kubernetes® with the appropriate configuration parameters for this software modification (these configuration parameters can be carried in Helm® charts and Kubernetes® manifests which can be delivered by NFVO/VNFM to the CISM).

Even though it is not possible to directly calculate the availability of pods containing the updated software, it is possible to perform some estimation based on previous experience with pods which contain the older software (refer to ETSI GS NFV-REL 003 [i.11], clause 5.3). In particular, it is assumed that the failure rate may typically be higher after a software modification, which implies that the MTBF is shorter, and thereby the availability is reduced. In practical terms, this means that software modification introduces some risks to VNF availability, and proactive measures may be needed to mitigate these risks.

The following shows an example of how the maxUnavailable attribute in Kubernetes® may need to be changed to ensure VNF availability after VNF software modification. This simple example is intended to demonstrate how to calculate a Kubernetes® configurable attribute, it is not intended to be a complete realistic example.

If a pool of 5 MCIO instances of the same VNFC are operating with active-active redundancy, the necessary value of maxUnavailable attribute in Kubernetes® can be calculated to meet the VNFC availability requirement. As an example to demonstrate the use of the formulae in clause 6.2.3, it is assumed that each MCIO instance hosting the new software has an availability of 99 %. The formulae can be used to calculate the availability of the containerized VNFC, and thereby the minimum viable MCIO pool size. In the example of 5 MCIO instances, to reach a VNFC availability level of 99,99 %, the pool contains at least 2 active MCIO instances, thus the maxUnavailable attribute is set to 3 in this case. Similarly, to reach a VNFC availability level of 99,999 %, the pool contains at least 3 active MCIO instances, thus the maxUnavailable attribute is set to 2 in this case.

6.2.5 Resiliency assurance for containerized VNF software modification

6.2.5.1 Actors and roles

Table 6.2.5.1-1 describes actors and roles of the resiliency assurance for containerized VNF software modification initiated by NFVO.

Table 6.2.5.1-1: Resiliency assurance for containerized VNF software modification actors and roles

#	Role	Description
1	NFVO	NFV Orchestrator managing the NS instance.
2	VNFM	VNF Manager managing one or more VNF instances of the NS instance.
3	CISM	CIS Manager managing one or more container infrastructure services.
4	AEAF	Availability Estimation and Analysis Function responsible for performing the VNF availability evaluation/estimation. The function might or might not be part of NFV-MANO.

6.2.5.2 Pre-conditions

Table 6.2.5.2-1 describes the use case pre-conditions.

Table 6.2.5.2-1: Resiliency assurance for containerized VNF software modification pre-conditions

#	Pre-condition	Additional description
1	The NS has been instantiated according to the relevant NSD and is working normally.	The containerized infrastructure is managed using Kubernetes®.
2	The NS instance provides its services according to the requested availability characteristics.	
3	The AEAF is equipped with models and algorithms and is ready to perform availability estimation for the containerized VNFs.	
4	The NFVO has received an *UpdateNS* operation with *updateType = ChangeVnfPkg* from the OSS and decides to initiate software modification for one containerized VNF.	

6.2.5.3 Post-conditions

Table 6.2.5.3-1 describes the use case post-conditions.

Table 6.2.5.3-1: Resiliency assurance for containerized VNF software modification post-conditions

#	Post-condition	Additional description
1	The containerized VNF is equipped with the new version of software.	
2	The NS instance continues to provide its services according to the requested availability characteristics.	

6.2.5.4 Flow description

Table 6.2.5.4-1 describes the use case flow.

Table 6.2.5.4-1: Flow description of resiliency assurance for containerized VNF software modification

#	Actor/Role	Action/Description
Begins when	NFVO → VNFM	The NFVO sends a *ChangeCurrentVnfPackage* request to the VNFM. See notes 1 and 2.
Step 1	VNFM	VNFM determines which VNFCs will be impacted by the software modification of the VNF. Impacts may be addition of VNFC(s), update of VNFC(s), or removal of VNFC(s).
Step 2	VNFM → AEAF	VNFM requests AEAF to perform the availability evaluation for the proposed VNFC(s) software modification.
Step 3	AEAF	AEAF reads the VNFD to fetch information (for example DeploymentFlavour) which is needed to infer the expected availability of the new VNFC(s).
		Step 4 to Step 5 are repeated for each impacted VNFC.
Step 4	AEAF → CISM	AEAF fetches information from CISM related to the VNFC, for example number of MCIOs and status of these MCIOs.
Step 5	AEAF	AEAF estimates the availability of the VNFC as a result of the software modification. See note 3.
Step 6	AEAF → VNFM	Taking into consideration the estimation of availability for each impacted VNFC, AEAF responds to VNFM with the analysis result. This analysis result may contain a recommendation to change Kubernetes® configurable attributes before the software modification can proceed.
Step 7 (optional)	VNFM → CISM	If the analysis result recommends to change Kubernetes® configurable attributes, VNFM sends updated attribute values to CISM according to the recommendations received from AEAF. Otherwise, there is no need to change Kubernetes® configurable attributes.
Step 8	VNFM → CISM	VNFM requests the CISM to update the VNFC(s) to the new software.
Step 9	CISM → VNFM	CISM informs VNFM that the update is complete.
Step 10	VNFM → NFVO	VNFM informs NFVO that the VNF software modification is complete.
Ends when	NFVO	NFVO has been informed that the VNF software modification is complete.
NOTE 1: This is an optimistic flow, fault conditions are not considered. NOTE 2: Aspects of service continuity during and after upgrade are not the core subject of this use case and are not described in this flow. NOTE 3: Refer to clause 7 for discussion of how this can be achieved.		

6.3 Cloud-native VNF scaling

6.3.1 Overview

NFV-MANO orchestrates the scaling in and out of VNFs. In relation to scaling operations for a cloud-native VNF, it is valuable to estimate the VNF availability for the following reasons.

A VNF scaling out operation may not only be triggered to meet current and future traffic, but also to meet the VNF availability requirements.

In practice, the purpose of scaling in a VNF is to avoid resource over-allocation through reducing unnecessary service capacity. In preparation for a scaling in operation, an evaluation of VNF availability can be used to predict if this operation may violate the availability requirements.

The Availability Estimation and Analysis Function can help decide if scaling operations are appropriate at a specific time.

Also, as part of any scaling operation, NFVO/VNFM is expected to supply CISM with the appropriate Kubernetes® configurable attributes. These attributes indicate the expected number of MCIOs that are provisioned to satisfy the availability requirements.

6.3.2 Containerized VNF scaling out

6.3.2.1 Introduction

A VNF scaling out may be necessary to reach the desired availability requirements for the VNF. The CISM will continuously monitor the CaaS metrics as input for availability evaluation for each containerized VNF (see clause 6.2 for details). The Availability Estimation and Analysis Function (AEAF) is proposed to use the information from the CISM for performing the VNF availability evaluation, and recommending to VNFM if a scaling out can be triggered.

This use case will explore how VNF availability evaluation can be used in cloud-native VNF scaling out.

6.3.2.2 Actors and roles

Table 6.3.2.2-1 describes the actors and roles of VNF availability evaluation for a containerized VNF scaling out operation.

Table 6.3.2.2-1: Actors and roles of VNF availability evaluation for a containerized VNF scaling out

#	Role	Description
1	VNFM	VNF Manager managing one or more VNF instances of the NS instance.
2	CISM	CIS Manager managing one or more container infrastructure services.
3	AEAF	Availability Estimation and Analysis Function responsible for performing the VNF availability evaluation/estimation. The function might or might not be part of NFV-MANO.

6.3.2.3 Pre-conditions

Table 6.3.2.3-1 describes the use case pre-conditions.

Table 6.3.2.3-1: Pre-conditions of VNF availability evaluation for a containerized VNF scaling out

#	Pre-condition	Additional description
1	The VNF is working normally.	
2	The AEAF is equipped with models and algorithms and is ready to perform availability estimation for the containerized VNFs.	
3	The VNFM intends to initiate a scaling out operation for a containerized VNF (for example, to tackle a surge of traffic).	

6.3.2.4 Post-conditions

Table 6.3.2.4-1 describes the use case post-conditions.

Table 6.3.2.4-1: Post-conditions of VNF availability evaluation for a containerized VNF scaling out

#	Post-condition	Additional description
1	VNFM has been informed of the VNF availability evaluation results.	If the analysis results show that the availability requirement of the VNF cannot be met after scaling out, AEAF may recommend that the scaling be increased further.

6.3.2.5 Flow description

Table 6.3.2.5-1 describes the use case flow.

Table 6.3.2.5-1: Flow description of VNF availability evaluation for a containerized VNF scaling out

#	Actor/Role	Action/Description
Begins when	VNFM	VNFM proposes a scaling out operation for a containerized VNF and derives a list of impacted VNFCs. See notes 1 and 2.
Step 1	VNFM → AEAF	VNFM requests AEAF to perform the availability estimation for the proposed addition of VNFC(s).
Step 2	AEAF	AEAF reads the VNFD to fetch information needed to infer the availability of the impacted VNFC(s).
		Step 3 to Step 4 are repeated for each impacted VNFC.
Step 3	AEAF → CISM	AEAF fetches information from CISM related to the VNFC, for example number of MCIOs and status of these MCIOs.
Step 4	AEAF	AEAF estimates the availability of the VNFC as a result of the scaling out. See note 3.
Step 5	AEAF → VNFM	AEAF responds to VNFM with the analysis result. This analysis result may contain a recommendation that the scaling be increased further.
Ends when	VNFM	VNFM has been informed of the availability estimation results related to the proposed scaling out operation which optionally include recommendation by AEAF.
NOTE 1: This is an optimistic flow, fault conditions are not considered. NOTE 2: Aspect of VNF availability evaluation after the scaling out operation (to confirm if the VNF provides its services according to the availability requirement as desired) is not the core subject of this use case and is not described in this flow. NOTE 3: Refer to clause 7 for discussion of how this can be achieved.		

6.3.3 Containerized VNF scaling in

6.3.3.1 Introduction

Typically, scaling in of a VNF is initiated as a result of resource over-allocation. This scaling in is implemented by removing VNFC instances that are deemed to be unnecessary. During any VNF scaling in, there is a risk for VNF availability because redundancy is typically reduced as the number of VNFC instances reduces. Therefore, availability estimation before the scaling in operation is needed. This scaling in operation is to be cancelled if the Availability Estimation and Analysis Function (AEAF) estimates that the desired availability requirement of the VNF cannot be met after the operation.

This use case will explore how VNF availability evaluation can be used in cloud-native VNF scaling in.

6.3.3.2 Actors and roles

Table 6.3.3.2-1 describes the actors and roles of the VNF availability evaluation for a containerized VNF scaling in operation.

Table 6.3.3.2-1: Actors and roles of VNF availability evaluation for a containerized VNF scaling in

#	Role	Description
1	VNFM	VNF Manager managing one or more VNF instances of the NS instance.
2	CISM	CIS Manager managing one or more container infrastructure services.
3	AEAF	Availability Estimation and Analysis Function responsible for performing the VNF availability evaluation/estimation. The function might or might not be part of NFV-MANO.

6.3.3.3 Pre-conditions

Table 6.3.3.3-1 describes the use case pre-conditions.

Table 6.3.3.3-1: Pre-conditions of VNF availability evaluation for a containerized VNF scaling in

#	Pre-condition	Additional description
1	The VNF is working normally.	The container infrastructure is managed using Kubernetes® (CISM).
2	The AEAF is equipped with models and algorithms and is ready to perform availability estimation for the containerized VNFs.	
3	The VNFM is going to initiate a scaling in operation for a containerized VNF (for example, to optimize resource over-allocation).	

6.3.3.4 Post-conditions

Table 6.3.3.4-1 describes the use case post-conditions.

Table 6.3.3.4-1: Post-conditions of VNF availability evaluation for a containerized VNF scaling in

#	Post-condition	Additional description
1	VNFM has been informed of the VNF availability estimation results.	If the analysis result shows that the desired availability requirement of the VNF cannot be met after scaling in, it can be recommended by AEAF to cancel this operation.

6.3.3.5 Flow description

Table 6.3.3.5-1 describes the use case flow.

Table 6.3.3.5-1: Flow description of VNF availability evaluation for a containerized VNF scaling in

#	Actor/Role	Action/Description
Begins when	VNFM	VNFM proposes a scaling in operation for a containerized VNF and derives a list of which VNFCs will be impacted. See notes 1 and 2.
Step 1	VNFM → AEAF	VNFM requests AEAF to perform the availability estimation for the proposed removal of VNFC instances.
Step 2	AEAF	AEAF reads the VNFD to fetch information which is needed to infer the expected availability of the impacted VNFCs.
		Step 3 to Step 4 are repeated for each impacted VNFC.
Step 3	AEAF → CISM	AEAF fetches information from CISM related to the VNFC, for example number of MCIOs and status of these MCIOs.
Step 4	AEAF	AEAF estimates the availability of the impacted VNFC as a result of the scaling in. See note 3.
Step 5	AEAF → VNFM	AEAF responds to VNFM with the analysis result. This analysis result may contain a recommendation to cancel the scaling in operation.
Ends when	VNFM	VNFM has been informed of the availability estimation results related to the proposed scaling in operation which optionally include recommendation by AEAF.
NOTE 1: This is an optimistic flow, fault conditions are not considered. NOTE 2: Aspects of VNF availability evaluation during and after scaling in operation are not the core subject of this use case and are not described in this flow. NOTE 3: Refer to clause 7 for discussion of how this can be achieved.		

7 Functionalities of AEAF

7.1 Evaluation request

As part of the use cases for containerized VNF software modification and scaling, VNFM determines which VNFCs of a VNF will be impacted and expresses the proposed changes in the evaluation request sent to the AEAF. The evaluation request also contains a reference to the Helm chart and Kubernetes® manifests where AEAF can find the related Kubernetes® configuration parameters for the proposed containerized VNF software modification or scaling.

For detailed specification of Helm chart as TOSCA-based NFV descriptors for containerized implementations, refer to the clause 6 of ETSI GS NFV-SOL 001 [i.12].

7.2 Evaluation methods

7.2.1 Introduction

The availability evaluation of cloud-native VNFs depends on the collection of indicators across the application, Container Infrastructure Service (CIS) and related infrastructure.

For cloud-native VNFs, ETSI GS NFV-IFA 027 [i.14] has provided some typical performance measurement metrics. Related metrics collected by VNFM can be used to observe the status of VNFs, VNFCs, internal and external connection points of VNF instances.

The container infrastructure service related metrics monitored by Kubernetes® are given in clause 5. Some of these metrics are useful for measuring the availability of MCIOs.

NOTE: Kubernetes® does not explicitly provide the metrics of CIS clusters and nodes, but focuses on those related to pods status. For measuring the availability of nodes and clusters which provide the environment hosting the containerized implementations, ETSI GS NFV-IFA 027 [i.14] can be used to provide some metrics related to virtualised compute, storage and network resources.

7.2.2 Evaluation methods classified by measured object

7.2.2.1 VNF

A VNF may be composed of one or multiple VNF Components (VNFC). For combining these VNFCs, different dependencies are possible, such as parallel, serial and a mix of these two. For related calculation methods, refer to the clause 5.2.2 of ETSI GS NFV-REL 003 [i.11].

7.2.2.2 VNFC

A VNFC may be composed of a pool of VNFC instances. A need for redundancy implies that the size of the pool is greater than the minimum pool size needed to provide the desired service.

In this case, if at least *n* ($n \geq 1$) instances of the same VNFC are needed to provide the desired service (for capacity or other reason), to ensure this VNFC could provide its service satisfactorily, a pool of *m* ($m \geq n$) active instances of this VNFC is needed.

If the availability of each VNFC instance is $A_{VNFCinstance}$, the availability of this VNFC may be calculated (according to the binomial distribution) as:

$$A_{VNFC} = \sum_{i=n}^{m} \binom{m}{i} A_{VNFCinstance}{}^{i} * (1 - A_{VNFCinstance})^{m-i} \qquad (5)$$

which provides the probability that *n* or more instances (within this *m*-size pool) of this VNFC are providing the desired service at a sampling time point during this VNFC's working period.

As described in clause 4.2, an MCIO can be mapped to a VNFC and an MCIO instance could be a Pod, a Deployment, or a StatefulSet, etc. in Kubernetes®.

As Kubernetes® natively only supports active-active redundancy mechanisms (see note) for VNFCs, all Managed Container Infrastructure Objects (MCIO) are considered as active from the Kubernetes® perspective.

NOTE: To achieve active-standby for containerized VNFCs, it is possible to add additional enhancements acting as the traffic (or service requests) entry point for Kubernetes managed workloads to achieve active-standby for containerized applications. These additional enhancements are out of the scope of the present document.

Therefore, for containerized implementation, the availability of a VNFC instance could be one of the following:

$$A_{VNFCinstance} = A_{MCIOinstance} = A_{pod} \quad (6a)$$

$$A_{VNFCinstance} = A_{MCIOinstance} = A_{deployment} \quad (6b)$$

$$A_{VNFCinstance} = A_{MCIOinstance} = A_{statefulset} \quad (6c)$$

7.2.2.3 Pod, Deployment and StatefulSet

To estimate the availability of a pod, AEAF needs to estimate the MTBF and MTTR for this pod. This can be based on historical information for similar pods, which may be collected from laboratory environments or from live network environments. Similarly, the availability of pod groups (Kubernetes® Deployments or Kubernetes® StatefulSets) may be estimated based on historical information for similar pod groups.

Kubernetes® provides its log system which is particularly useful for debugging pod problems and monitoring cluster activity. However, the native functionality provided by Kubernetes® is usually not enough for a complete historical information solution. But Kubernetes® can be integrated with cluster and cross-cluster monitoring and logging solutions, which provide separate backend to collect, store, and analyse this historical information.

When a single pod's failure is detected by Kubernetes®, it will be terminated and replaced by a new pod providing the desired service. When failure of a pod (or pods) leads to a group of pods providing unavailable or degraded service, the healing/replacement of this pod group will be executed by Kubernetes®. Figure 7.2.2.3-1 shows the restoration sequence for a failed pod group. Figure 7.2.2.3-2 shows the restoration sequence for a degraded pod group.

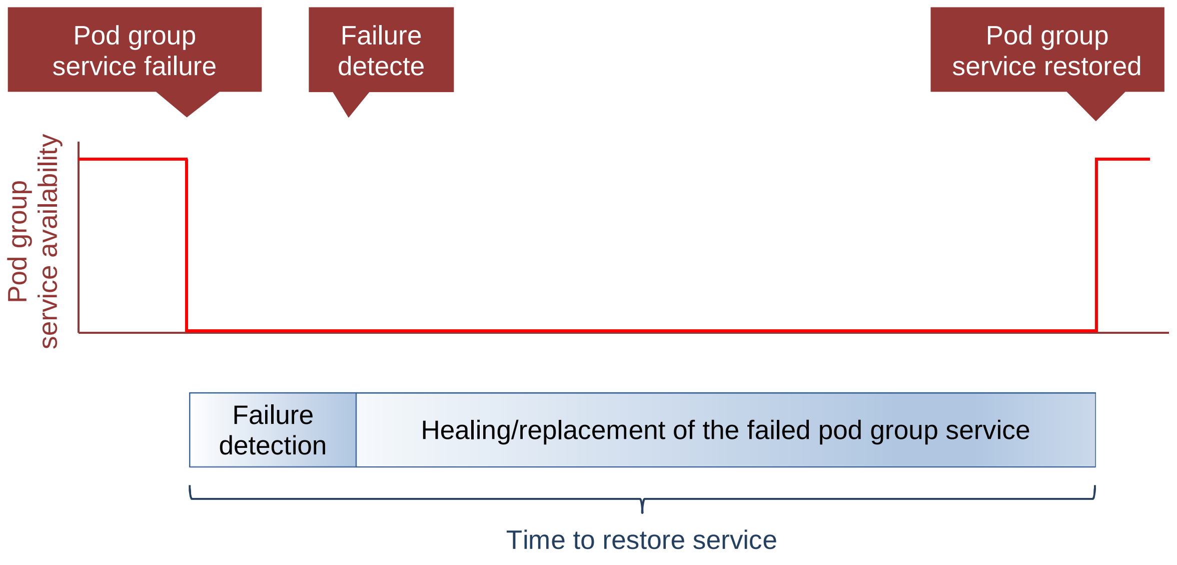
Pod group service failure
Failure detecte
Pod group service restored
Pod group service availability
Failure detection
Healing/replacement of the failed pod group service
Time to restore service

Figure 7.2.2.3-1: Example of restoration sequence for a failed pod group service

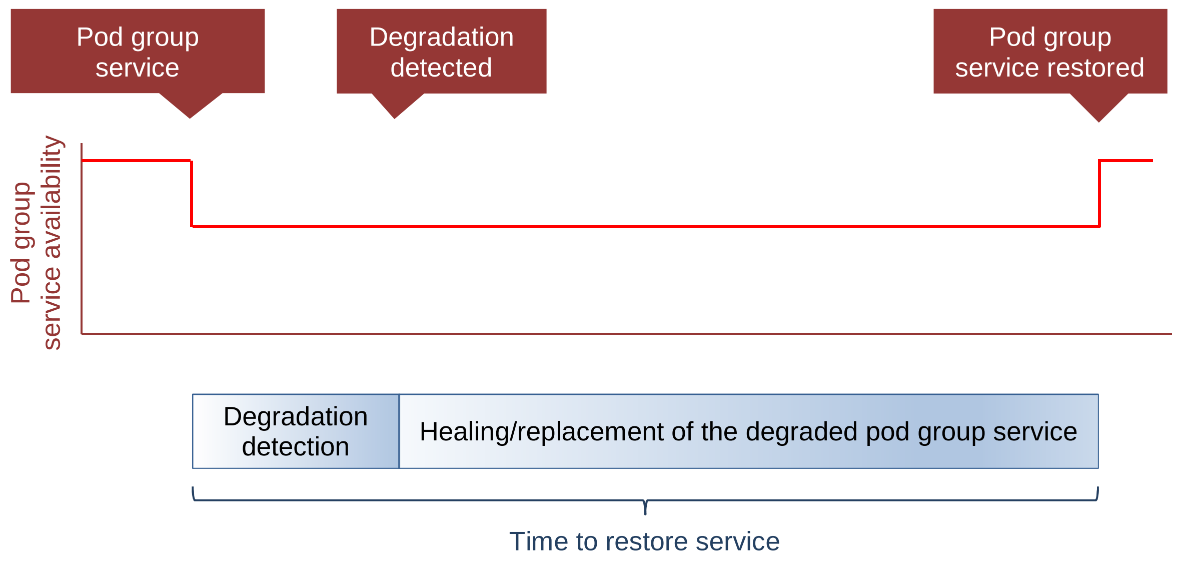
Pod group service
Degradation detected
Pod group service restored
Pod group service availability
Degradation detection
Healing/replacement of the degraded pod group service
Time to restore service

Figure 7.2.2.3-2: Example of restoration sequence for a degraded pod group service

Taking the Figure 7.2.2.3-1 as an example, one can see that the MTTR is not only affected by the time to respond to a failure, but also the non-zero time needed to detect that a failure has occurred. In a live network, it is not possible to measure this time, because the management system is not aware of the failure itself, but is only informed when a failure has been detected. However, to gain a good estimation of the failure detection delay, it is possible to deliberately inject faults in a laboratory environment and measure the typical delay until the management system detects the fault. Multiple classes of faults may need to be injected (for example connectivity, storage, processor) because each class may have a different typical delay until detection.

The typical time from failure detection to service restoration can be measured by observing historical data for similar pods (or pod groups), especially the Kubernetes® state transitions.

Assuming complete availability of the infrastructure used to deploy the pod (see Formula (1)), AEAF may estimate the availability of a single pod as:

$$A_{pod} = \frac{MTBF_{pod}}{MTBF_{pod}+MTTR_{pod}} \quad (7)$$

$MTBF_{pod}$ may be estimated by historical analysis of similar pods.

AEAF may estimate the MTTR of a single pod as:

$$MTTR_{pod} = DetectionTime_{pod} + RestorationTime_{pod} \quad (8)$$

As mentioned above, $DetectionTime_{pod}$ cannot be calculated based on information from the live network, but it may be estimated based on offline information.

AEAF may estimate the RestorationTime of a single pod as:

$$RestorationTime_{pod} = Tready_{pod} - Tfailed_{pod} \quad (9)$$

Where $Tready_{pod}$ is the time Kubernetes® sets the attribute kube_pod_status_ready = true, and $Tfailed_{pod}$ is the time Kubernetes® sets the attribute kube_pod_status_ready = false.

For a Deployment, it is a more realistic scenario that service is degraded, but has not completely failed. Thus, Figure 7.2.2.3-2 is more suitable for this situation. In this case, AEAF may estimate the availability ($A_{deployment}$), MTTR and MTBF of a Deployment the same way as for the estimation of pod availability, refer to the related earlier text and formulae (7) and (8).

AEAF may estimate the RestorationTime of a Deployment as:

$$RestorationTime_{dep} = Tready_{dep} - Tdegraded_{dep} \quad (10)$$

Where Tready$_{dep}$ is the time Kubernetes® sets the attribute kube_pod_status_ready = true for all pods of this Deployment, and Tfailed$_{dep}$ is the time Kubernetes® sets the attribute kube_pod_status_ready = false for at least one pod of this Deployment.

A StatefulSet provides guarantees about the ordering and uniqueness of a set of stateful pods. If at least one pod of this StatefulSet failed, the StatefulSet is assumed to be unavailable. Therefore, Figure 7.2.2.3-1 is more suitable for this situation. In this case, AEAF may estimate the availability ($A_{statefulset}$), MTTR and MTBF of a StatefulSet the same way as for the estimation of pod availability, refer to the related earlier text and formulae (7) and (8).

AEAF may estimate the RestorationTime of a StatefulSet as:

$$RestorationTime_{ss} = Tready_{ss} - Tfailed_{ss} \tag{11}$$

Where Tready$_{ss}$ is the time Kubernetes® sets the attribute kube_pod_status_ready = true for all pods of this StatefulSet, and Tfailed$_{ss}$ is the time Kubernetes® sets the attribute kube_pod_status_ready = false for at least one pod of this StatefulSet.

7.2.2.4 Node

To estimate the availability of a node, AEAF needs to estimate the MTBF and MTTR for this node. This can be based on vendor information and historical information for similar nodes, which may be collected from laboratory environments or from live network environments. Figure 7.2.2.4-1 shows the restoration sequence for a failed node.

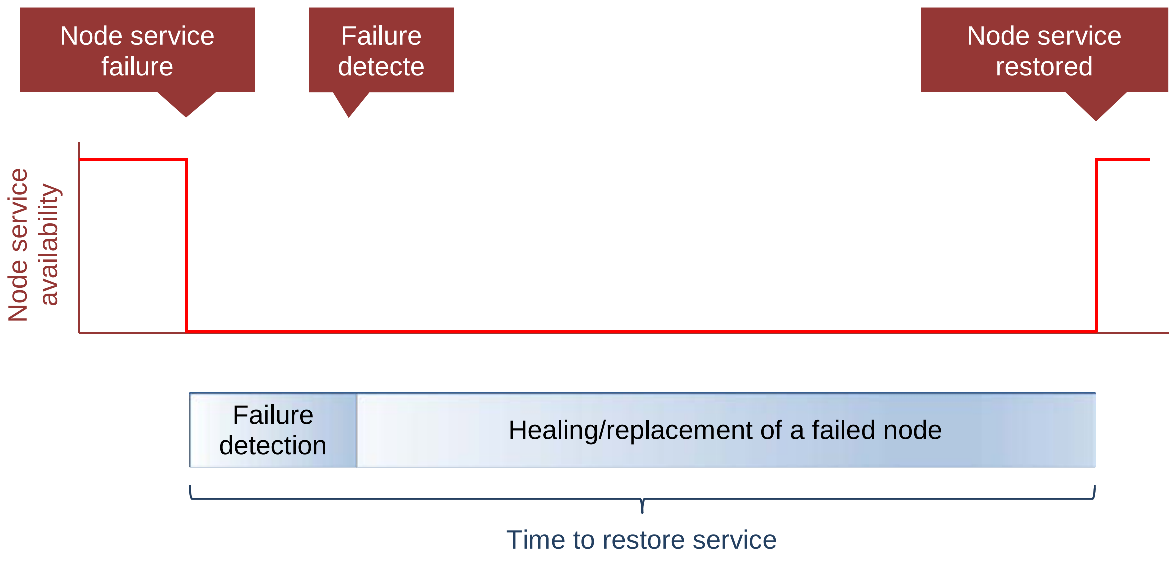

Figure 7.2.2.4-1: Example of restoration sequence for a failed node

The typical time from failure detection to service restoration can be measured by observing historical data for similar nodes.

AEAF may estimate the availability of a single node as:

$$A_{node} = \frac{MTBF_{node}}{MTBF_{node}+MTTR_{node}} \tag{12}$$

The vendor may provide an estimation of MTBF$_{node}$, or it may be estimated by historical analysis of similar nodes.

AEAF may estimate the MTTR of a single node as:

$$MTTR_{node} = DetectionTime_{node} + RestorationTime_{node} \tag{13}$$

Similar to the description for pods, the DetectionTime$_{node}$ cannot be calculated based on information from the live network, it may be estimated based on offline information gathered by fault injection experiments, etc.

AEAF may estimate the RestorationTime of a single node as:

$$RestorationTime_{node} = Tready_{node} - Tfailed_{node} \tag{14}$$

Where $\text{Tready}_{\text{node}}$ is the time Kubernetes® sets the attribute kube_node_status_ready = true for this node, and $\text{Tfailed}_{\text{node}}$ is the time Kubernetes® sets the attribute kube_node_status_ready = false for this node.

7.2.2.5 CIS cluster

According to the CIS cluster fault management service interface requirement, when a CIS cluster service failure is detected by CCM, CCM enables its consumers to collect CIS cluster fault information [i.7]. It is proposed that the NFV-MANO provides the healing/replacement of a failed CIS cluster.

Figure 7.2.2.5-1 shows the restoration sequence for a failed CIS cluster.

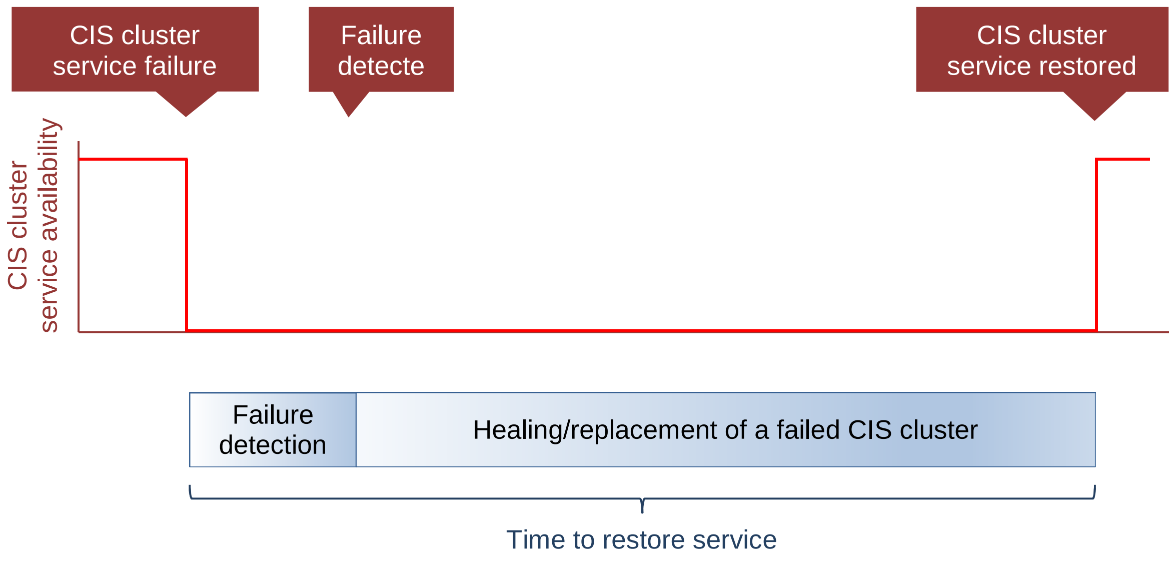
CIS cluster service failure
Failure detecte
CIS cluster service restored
CIS cluster service availability
Failure detection
Healing/replacement of a failed CIS cluster
Time to restore service

Figure 7.2.2.5-1: Example of restoration sequence for a failed CIS cluster

The typical time from failure detection to service restoration can be measured by observing historical data for similar CIS clusters.

AEAF may estimate the availability of a CIS cluster as:

$$A_{cluster} = \frac{MTBF_{cluster}}{MTBF_{cluster}+MTTR_{cluster}} \qquad (15)$$

$\text{MTBF}_{\text{cluster}}$ may be estimated by historical analysis of similar clusters.

AEAF may estimate the MTTR of a CIS cluster as:

$$MTTR_{cluster} = DetectionTime_{cluster} + RestorationTime_{cluster} \qquad (16)$$

Similar to the description for nodes, the $\text{DetectionTime}_{\text{cluster}}$ cannot be calculated based on information from the live network, it may be estimated based on offline information gathered by fault injection experiments, etc.

AEAF may estimate the RestorationTime of a CIS cluster as:

$$RestorationTime_{cluster} = Tready_{cluster} - Tfailed_{cluster} \qquad (17)$$

Where $\text{Tready}_{\text{cluster}}$ is the time CCM sends notifications in event of the clearance of a fault for a CIS cluster, and $\text{Tfailed}_{\text{cluster}}$ is the time CCM sends notifications in event of a fault detected for a CIS cluster.

7.3 Evaluation outputs

Generally, the typical output of the evaluation contains an indicator showing the result on whether the desired availability requirement of the VNF can or cannot be met after the proposed operation.

In addition, for the containerized VNF software modification use case, this evaluation output may contain a recommendation to change Kubernetes® configurable attributes before the software modification can proceed. For the containerized VNF scaling out use case, if AEAF estimates that the availability requirement of the VNF cannot be met after scaling out, this analysis result may contain a recommendation that the scaling be increased further.

AEAF's evaluation outputs related to the reliability and availability of NFV managed objects can be used by VNFM and other functional entities to determine appropriate management actions in cloud-native VNF deployment, operation and maintenance.

7.4 Potential architectural options related to AEAF

Based on the use cases and AEAF functionality, the following architectural options are described for AEAF:

1) The AEAF resides in the VNFM.

2) The AEAF is part of the Management Data Analytics (MDA) function.

3) The AEAF is a new functional block within NFV-MANO.

4) The AEAF lies outside NFV-MANO.

Option #1: The AEAF resides in the VNFM

With this architectural option, the AEAF is co-located with the VNFM. In this scenario, there is no need to standardize the interface between VNFM and AEAF.

Option #2: The AEAF is part of the MDA function

With this architectural option, the AEAF is one function of the set of MDA functions. ETSI GR NFV-IFA 041 [i.13] specifies that MDA can provide the capability of processing and analysing the raw data related to network and service events and status to provide analytics report to enable necessary actions for network operations. For potential options for integrating MDA functions with NFV-MANO, refer to clause 7.2.2 in ETSI GR NFV-IFA 041 [i.13]. In this scenario, the interface for the analysis request and response process between VNFM and AEAF is described in clause 7.2.2 in ETSI GR NFV-IFA 041 [i.13].

Option #3: The AEAF is a new functional block within NFV-MANO

With this architectural option, the AEAF is represented by its own NFV-MANO functional block. In this scenario, a new interface between VNFM and AEAF needs to be defined.

Option #4: The AEAF lies outside NFV-MANO

With this architectural option, the AEAF is a functional entity external to NFV-MANO. In this scenario, requirements on the new interface between VNFM and AEAF may be defined, but the definition of the interface is out of the scope of ETSI ISG NFV.

8 Recommendations related to AEAF

This clause provides recommendations for NFV-MANO which are derived from the use cases discussed in clause 6. The recommendations are made from a reliability and availability point of view.

The term "Availability Estimation and Analysis Function" is used for the purpose to describe the functionality and to address the entity that is providing that functionality. No assumption is made about what entity or entities can play such a role. This functionality may be a standalone function or may be part of another function, for example Management Data Analytics Function (MDAF).

Table 8-1 provides recommendations related to Availability Estimation and Analysis Function.

Table 8-1: Recommendations related to Availability Estimation and Analysis Function

Identifier	Recommendation description	Reference
Aeaf.001	It is recommended that an Availability Estimation and Analysis Function is provided to support VNF availability estimation for the containerized VNF software modification operation.	Clause 6.2.5
Aeaf.002	It is recommended that an Availability Estimation and Analysis Function is provided to support VNF availability estimation for the containerized VNF scaling out operation.	Clause 6.3.2
Aeaf.003	It is recommended that an Availability Estimation and Analysis Function is provided to support VNF availability estimation for the containerized VNF scaling in operation.	Clause 6.3.3
Aeaf.004	It is recommended that the Availability Estimation and Analysis Function is able to read the VNFD of the relevant VNF to fetch information needed to infer the availability of the VNFC(s) which will be impacted by a lifecycle management operation.	Clause 6
Aeaf.005	It is recommended that the Availability Estimation and Analysis Function is able to fetch information from CISM related to the VNFC(s) impacted by a lifecycle management operation, for example number of MCIOs and status of these MCIOs.	Clause 6
Aeaf.006	It is recommended that the Availability Estimation and Analysis Function is able to provide an analysis result for the containerized VNF software modification operation, which may contain a recommendation to change Kubernetes® configurable attributes before the software modification can proceed.	Clause 6.2.5
Aeaf.007	It is recommended that the Availability Estimation and Analysis Function is able to provide an analysis result for the containerized VNF scaling out operation, which may contain a recommendation that the scaling out level be increased appropriately.	Clause 6.3.2
Aeaf.008	It is recommended that the Availability Estimation and Analysis Function is able to provide an analysis result for the containerized VNF scaling in operation, which may contain a recommendation to cancel this scaling in operation.	Clause 6.3.3

9 Conclusion

The present document has studied the reliability evaluation for cloud-native VNFs. Cloud-native configuration capabilities and metrics related to reliability were introduced in clause 5. In clause 6, three use cases were developed and analysed with the introduction of the Availability Estimation and Analysis Function (AEAF) and how this function could support VNF availability estimation for the resiliency assurance of the related operations:

- Containerized VNF software modification.
- Containerized VNF scaling out.
- Containerized VNF scaling in.

Although these use cases are representative from the reliability perspective of the lifecycle management of cloud-native VNFs, there may be more aspects and use cases which could be considered.

Evaluation requests, methods, and outputs related to the functionalities of Availability Estimation and Analysis Function were presented in clause 7, together with the potential architectural options for this function within or outside NFV-MANO.

From the three use cases and the functionality discussion, a set of recommendations were derived in clause 8 to provide new functionalities and their different aspects in an NFV environment. Implementing these recommendations would significantly improve the reliability and availability of cloud-native VNFs for the use cases tackled in the present document.

Annex A: Existing pod metrics and configurable attributes in Kubernetes®

Table A-1 shows the existing pod metrics and configurable attributes in Kubernetes®.

Table A-1: Kubernetes® pod metrics and configurable attributes

Metric name	Description	Type
kube_pod_annotations	Kubernetes® annotations converted to Prometheus labels	Configurable attributes
kube_pod_info	Information about pod	Configurable attributes
kube_pod_ips	Pod IP addresses	Configurable attributes
kube_pod_start_time	Start time in Unix timestamp for a pod	Metrics
kube_pod_completion_time	Completion time in Unix timestamp for a pod	Metrics
kube_pod_owner	Information about the pod's owner	Configurable attributes
kube_pod_labels	Kubernetes® labels converted to Prometheus labels	Configurable attributes
kube_pod_nodeselectors	Indicates the pod nodeSelectors	Configurable attributes
kube_pod_status_phase	Pods current phase	Metrics
kube_pod_status_ready	Indicates whether the pod is ready to serve requests	Metrics
kube_pod_status_scheduled	Indicates the status of the scheduling process for the pod	Metrics
kube_pod_container_info	Information about a container in a pod	Configurable attributes
kube_pod_container_status_waiting	Indicates whether the container is currently in waiting state	Metrics
kube_pod_container_status_waiting_reason	Indicates the reason the container is currently in waiting state	Metrics
kube_pod_container_status_running	Indicates whether the container is currently in running state	Metrics
kube_pod_container_state_started	Start time in Unix timestamp for a pod container	Metrics
kube_pod_container_status_terminated	Indicates whether the container is currently in terminated state	Metrics
kube_pod_container_status_terminated_reason	Indicates the reason the container is currently in terminated state	Metrics
kube_pod_container_status_last_terminated_reason	Indicates the last reason the container was in terminated state	Metrics
kube_pod_container_status_ready	Indicates whether the containers readiness check succeeded	Metrics
kube_pod_container_status_restarts_total	Number of container restarts per container	Metrics
kube_pod_container_resource_requests	Number of requested request resource by a container	Configurable attributes
kube_pod_container_resource_limits	Number of requested limit resource by a container	Configurable attributes
kube_pod_overhead_cpu_cores	Pod overhead in regards to CPU cores associated with running a pod	Metrics
kube_pod_overhead_memory_bytes	Pod overhead in regards to memory associated with running a pod	Metrics
kube_pod_runtimeclass_name_info	Runtimeclass associated with the pod	Configurable attributes
kube_pod_created	Unix creation timestamp	Metrics
kube_pod_deletion_timestamp	Unix deletion timestamp	Metrics
kube_pod_restart_policy	Indicates the restart policy in use by this pod	Configurable attributes
kube_pod_init_container_info	Information about an init container in a pod	Configurable attributes

Metric name	Description	Type
kube_pod_init_container_status_waiting	Indicates whether the init container is currently in waiting state	Metrics
kube_pod_init_container_status_waiting_reason	Indicates the reason the init container is currently in waiting state	Metrics
kube_pod_init_container_status_running	Indicates whether the init container is currently in running state	Metrics
kube_pod_init_container_status_terminated	Indicates whether the init container is currently in terminated state	Metrics
kube_pod_init_container_status_terminated_reason	Indicates the reason the init container is currently in terminated state	Metrics
kube_pod_init_container_status_last_terminated_reason	Indicates the last reason the init container was in terminated state	Metrics
kube_pod_init_container_status_ready	Indicates whether the init container readiness check succeeded	Metrics
kube_pod_init_container_status_restarts_total	Number of restarts for the init container	Metrics
kube_pod_init_container_resource_limits	Number of CPU cores requested limit by an init container	Configurable attributes
kube_pod_init_container_resource_requests	Number of CPU cores requested by an init container	Metrics
kube_pod_spec_volumes_persistentvolumeclaims_info	Information about persistentvolumeclaim volumes in a pod	Configurable attributes
kube_pod_spec_volumes_persistentvolumeclaims_readonly	Indicates whether a persistentvolumeclaim is mounted read only	Configurable attributes
kube_pod_status_reason	Pod status reasons	Metrics
kube_pod_status_scheduled_time	Unix timestamp when pod moved into scheduled status	Metrics
kube_pod_status_unschedulable	Indicates the unschedulable status for the pod	Metrics
kube_pod_tolerations	Information about the pod tolerations	Configurable attributes

Annex B: Change History

Date	Version	Information about changes
October 2021	0.0.1	Initial draft version including the skeleton of the GR and scope clause: NFVREL(21)000141r1, NFVREL(21)000142r2
March 2022	0.0.2	Early draft version including the following approved contributions: NFVREL(21)000149r2, NFVREL(21)000165r3, NFVREL(22)000024
July 2022	0.0.3	Early draft version including the following approved contributions: NFVREL(22)000027r2, NFVREL(22)000038r2, NFVREL(22)000039r4, NFVREL(22)000045r3, NFVREL(22)000063r1
January 2023	0.0.4	Draft version including the following approved contributions: NFVREL(23)000005r4, NFVREL(23)000002r2, NFVREL(22)000136, NFVREL(22)000125r10, NFVREL(22)000096r8, NFVREL(22)000101, NFVREL(22)000092r2, NFVREL(22)000064r1
March 2023	0.0.5	Draft version including the following approved contributions: NFVREL(23)000014r2, NFVREL(23)000019r2, NFVREL(23)000020r1, NFVREL(23)000021r1, NFVREL(23)000028r5, NFVREL(23)000029r6
May 2023	0.0.6	Draft version including the following approved contributions: NFVREL(23)000042, NFVREL(23)000043r1, NFVREL(23)000044r1, NFVREL(23)000046r1, NFVREL(23)000049r1, NFVREL(23)000052r1, and the following editorial clean-ups (a) editorial clean-up in clause 7 related to ordering the equation numbers from equation No.5 to No.17 correctly; (b) editorial clean-up in clause 2.2 related to the Editor's Note on confirming that Ref [i,7] has been published; (c) editorial clean-up in clause 2.2 related to adding the Ref [i,14] ETSI GS NFV-IFA 027, which is cited in the main text clause 7.2.1, but not appears in the list of references.
June 2023	0.1.0	Stable draft version including the following approved contributions: NFVREL(23)000062r2, NFVREL(23)000063r2, NFVREL(23)000064r1, NFVREL(23)000066r1 , NFVREL(23)000067; and the following editorial clean-ups: a) Delete empty Annex B since it is just a sample format provided by ETSI and never used; b) Re-number clause 4.3 to 4.2 since clause 4.2 is deleted in NFVREL(23)000062r2; c) Remove duplicated text in clause 6.2.2.8 step 6 ("AEAF responds to VNFM with the analysis result.") since the previous sentence already covers this text. This remove is as agreed in REL#450 but missed in contribution NFVREL(23)000062's revision r2.
August 2023	0.2.0	Revised stable draft version including the following approved contributions: NFVREL(23)000076r1, NFVREL(23)000077r1, NFVREL(23)000078r1, NFVREL(23)000080, modify all Kubernetes to Kubernetes®, add that active-standby for containerized workload is out of the scope of the present document, amend the scope to add a reference to Kubernetes®, and editorial clean-ups according to NFVREL(23)000074 comments as rapporteur actions.

History

Document history		
V5.1.1	October 2023	Publication