|  |
| --- |
| O-RAN.WG6.O2DMS-INTERFACE-K8S-PROFILE-R003-v05.00 |
| Technical Specification |
| O-RAN Working Group 6  O2dms Interface Specification: Kubernetes Native API Profile for Containerized NFs |
|  |
|  |

Copyright © 2024 by the O-RAN ALLIANCE e.V.

The copying or incorporation into any other work of part or all of the material available in this specification in any form without the prior written permission of O-RAN ALLIANCE e.V. is prohibited, save that you may print or download extracts of the material of this specification for your personal use, or copy the material of this specification for the purpose of sending to individual third parties for their information provided that you acknowledge O-RAN ALLIANCE as the source of the material and that you inform the third party that these conditions apply to them and that they must comply with them.

O-RAN ALLIANCE e.V., Buschkauler Weg 27, 53347 Alfter, Germany

Register of Associations, Bonn VR 11238, VAT ID DE321720189

# Contents

[Contents 2](#_Toc162445490)

[1 Introduction 3](#_Toc162445491)

[1.1 Scope 3](#_Toc162445492)

[1.2 References 3](#_Toc162445493)

[1.3 Definitions and Abbreviations 3](#_Toc162445494)

[1.3.1 Definitions 3](#_Toc162445495)

[1.3.2 Abbreviations 3](#_Toc162445496)

[2 O2dms Service Overview 5](#_Toc162445497)

[2.1 Overview 5](#_Toc162445498)

[2.2 Supported Use Cases 5](#_Toc162445499)

[3 O2dms Service Definitions 6](#_Toc162445501)

[3.1 General 6](#_Toc162445502)

[3.1.1 The O2dms Services Supported 6](#_Toc162445503)

[3.1.2 Instantiate NF Deployment 6](#_Toc162445504)

[3.1.3 Terminate NF Deployment 8](#_Toc162445505)

[3.1.4 Heal NF Deployment 10](#_Toc162445506)

[3.1.5 Software Upgrade of NF Deployment 12](#_Toc162445507)

[3.1.6 Scale NF Deployment 12](#_Toc162445508)

[4 O2dms Service Definitions for Kubernetes 15](#_Toc162445509)

[4.1 General 15](#_Toc162445510)

[4.2 Referenced Cloud-Native APIs and Data Model Solutions 15](#_Toc162445511)

[4.2.1 Kubernetes API overview 15](#_Toc162445512)

[4.3 Referenced K8s resource objects 16](#_Toc162445513)

[4.3.1 Kubernetes Native Namespace Scoped Resource Objects 17](#_Toc162445514)

[4.3.2 Kubernetes Native Cluster Scoped Resource Objects 30](#_Toc162445515)

[Annex (informative): Change History 31](#_Toc162445516)

# Introduction

## Scope

This Technical Specification has been produced by the O-RAN.org.

The contents of the present document are subject to continuing work within O-RAN WG6 and may change following formal O-RAN approval. Should the O-RAN.org modify the contents of the present document, it will be re-released by O-RAN Alliance with an identifying change of release date and an increase in version number as follows:

Release x.y.z

where:

x the first digit is incremented for all changes of substance, i.e., technical enhancements, corrections, updates, etc. (the initial approved document will have x=01).

y the second digit is incremented when editorial only changes have been incorporated in the document.

z the third digit included only in working versions of the document indicating incremental changes during the editing process.

This document defines O-RAN O-Cloud DMS interface functions and protocols for the O-RAN O2 interface. The document studies the functions conveyed over the interface, including management functions, procedures, operations, and corresponding solutions, and identifies existing standards and industry work that can serve as a basis for O-RAN work.

## References

The following documents contain provisions which, through reference in this text, constitute provisions of this specification (see also <https://www.o-ran.org/specifications>).

1. 3GPP TR 21.905, Vocabulary for 3GPP Specifications
2. O-RAN WG6, Orchestration Use Case and Requirements for O-RAN Virtualized RAN
3. O-RAN WG6, Cloud Architecture and Deployment Scenarios for O-RAN Virtualized RAN
4. O-RAN WG6, O-RAN O2 General Aspects and Principles Specification
5. Kubernetes® API, online <https://kubernetes.io/docs/reference/kubernetes-api/>
6. Kubernetes® API Conventions, online <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md>

NOTE: Kubernetes® and K8s® are registered trademarks of the Linux Foundation, in the United States and other countries. O-RAN is not affiliated with, endorsed, or sponsored by the Linux Foundation.

## Definitions and Abbreviations

### Definitions

### Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1].

DMS Deployment Management Services

API Application Programming Interface

HPA HorizontalPodAutoscaler

SMO Service Management and Orchestration

NFO Network Function Orchestration

# O2dms Service Overview

## Overview

This document deals with the containerized workload lifecycle management services offered over the O2dms interface using the O2dms profile for Kubernetes. The relationship with O2ims services is also described whenever applicable (e.g., provisioning of Kubernetes labels in the O-Cloud Kubernetes cluster by SMO to be able to match them against the O-RAN CNF deployment requirements).

This profile exposes the full set of capabilities of Kubernetes native “API Server” function to the SMO and enables the reuse of Kubernetes native API procedures and data model (Kubernetes resource manifests). As depicted in Figure 2.1-1, the O2dms profile for Kubernetes exposes one O2dms interface instance per one Kubernetes cluster to the SMO and the interface is terminated by the API Server component of Kubernetes native control plane. The address and access credentials required for communication with the Kubernetes API Server component are assumed to be available to the SMO (for example via O2-IMS, from the Kubernetes cluster instantiation time).

Graphical user interface, application

Description automatically generated

**Figure 2.1-1**

## Supported Use Cases

The list of orchestration use-cases captured in [2] needs to be supported using O2dms capabilities for containerized workloads (NFs and other workloads). This version of the O2dms profile for Kubernetes supports the following use cases aligned with [2] based on Kubernetes native APIs and resource objects.

* Instantiation of containerized workloads on Kubernetes cluster: This O2dms capability enables the SMO to instantiate a containerized workload on a target Kubernetes cluster using the Kubernetes native resource manifests (Deployments, StatefulSets, ConfigMaps, etc.). The complete set of Kubernetes native resource objects may be used for workload instantiation including network, compute, storage, and configuration related objects.
* Termination of containerized workloads on Kubernetes cluster: This O2dms capability enables the SMO to terminate a containerized workload on a target Kubernetes cluster by deleting all the Kubernetes native resource manifests related to that workload from the target Kubernetes cluster.
* NF Deployment level healing: This O2dms capability enables the SMO to recover an NF Deployment from failures on a Kubernetes cluster. The SMO may trigger the NF Deployment recovery by creating, modifying, and/or deleting K8s native resource manifests belonging to that NF Deployment.
* Scale NF Deployment on Kubernetes cluster: This O2dms capability enables the SMO to elastically scale the NF Deployment’s service capacity to the service demands. Additional K8s resources may be allocated or deallocated as part of this use case.

# O2dms Service Definitions

## General

This section provides details of the O2dms interface capabilities for containerized workload management exposed to the SMO using the O2dms Kubernetes profile. The O2dms Kubernetes profile exposes a REST based interface to the SMO from the API Server function of Kubernetes native control plane. The protocol stack used by O2dms Kubernetesprofile for secure communication between the SMO and the target Kubernetes cluster is depicted in Figure 3.1-1. This protocol stack enables a secure transfer of the Kubernetes native API objects from the SMO to the target Kubernetes cluster.

A picture containing graphical user interface

Description automatically generated

**Figure 3.1-1: Illustration of the protocol stack used by O2dms Kubernetes profile**

### The O2dms Services Supported

In this specification, the Kubernetes native APIs provide all the required capabilities to the SMO for containerized workload lifecycle management in a Kubernetes cluster. The Kubernetes native APIs are exposed by the API Server function of the Kubernetesnative control plane and is the common denominator for the SMO to manage containerized workloads on Kubernetes based O-Clouds.

The Kubernetes API Sever exposes a REST based interface, offering CRUD (Create, Read, Update, Delete) operations using Kubernetes native resource manifests. Aligned with this REST interface, the ORAN WG6 use cases captured in [2] for lifecycle management of containerized workload are realized by using the API Server’s supported operations (CRUD) and data model (Kubernetes native resource manifests).

### Instantiate NF Deployment

The instantiation use case requires the SMO to start the execution of a containerized workload on a Kubernetes cluster that meets the execution requirements of that workload. In this O2dms profile, the instantiation use case is realized by secured transfer of the Kubernetes native resource objects from the SMO to the API Server function in the target Kubernetes cluster. A containerized network function usually consists of several Kubernetes resource object manifests for different types of resources and a secured transfer of all these manifests is required for successful compeletion of the instantiation use case.

Figure 3.1.2-1 exemplifies the instantiation flow in O2dms Kubernetes profile aligned with the NF instantiation use case in [2].

@startuml

Autonumber

skinparam sequenceArrowThickness 1

skinparam ParticipantPadding 5

skinparam BoxPadding 10

participant User

Box "SMO" #gold

Participant NFO as NFO

end box

Box "Kubernetes® based O-Cloud" #lightseagreen

participant apiserver as "K8s® API Server"

participant worker as "K8s® worker"

End box

Note over NFO, apiserver

PRECONDITIONS:

SMO is active and running normally

K8s® cluster is active and running normally

SMO and K8s® API Server connectivity is established

End Note

== Use case: Instantiate containerized workload on K8s® cluster ==

User -> NFO : Instantiate CNF (Id)

Note over NFO

SMO gets the deployment artifacts for

containerized workload (e.g., ASD, Helm Charts)

from a repository

end Note

Note over NFO

If unspecified, the SMO performs homing to select

a suitable K8s® cluster and adds information in

the inventory for the workload being deployed

end Note

Note over NFO

SMO generates parameterized K8s® manifests

for the containerized workload

end Note

loop Until all K8S resource manifests are sent to K8s® cluster

NFO -> apiserver : \*\*[O2dms]\*\* <<HTTP POST>> Create K8s® resource object

apiserver -> apiserver: Create K8s® resource

apiserver -> NFO : \*\*[O2dms]\*\* <<HTTP 201>> CREATED

end

Note over apiserver

K8s® API Server notifies the worker

node(s) to start workload execution

end note

Note over worker

K8s® worker fetches the container

images and starts their execution

end note

note over NFO

SMO acquires the workload instantiation status

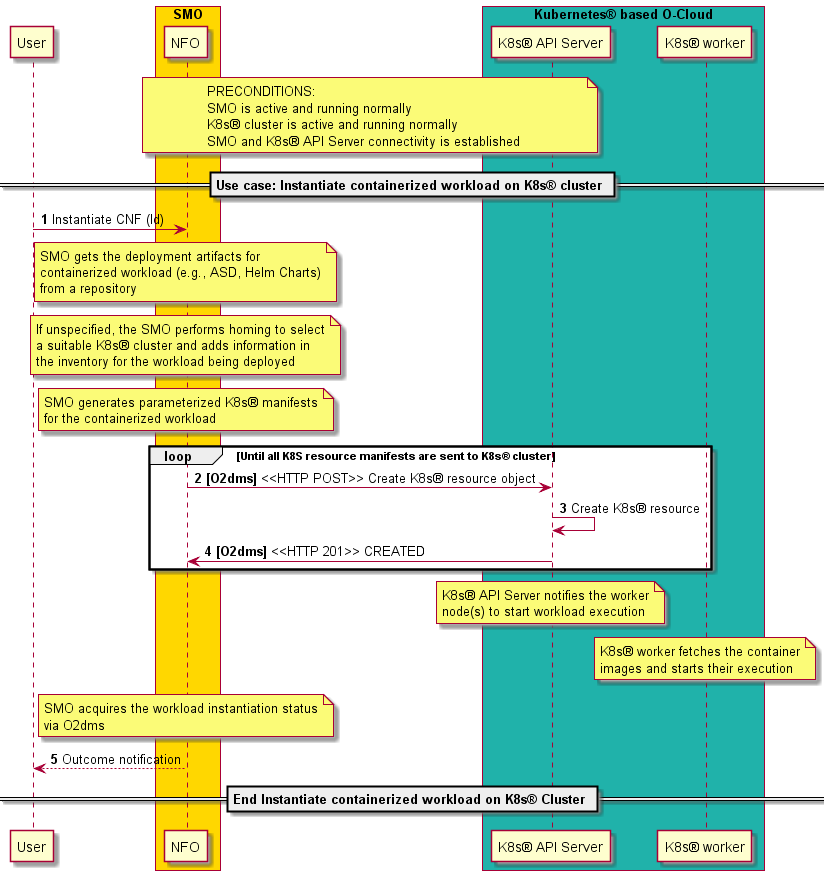
via O2dms

end note

NFO --> User : Outcome notification

== End Instantiate containerized workload on K8s® Cluster ==

@enduml



**Figure 3.1.2-1: NF Deployment Instantiation**

**Table 3.1.2-1: Instantiation Use Case**

|  |  |
| --- | --- |
| Use Case Stage | Evolution/Specification |
| Use Case | Instantiation of containerized workload on a selected Kubernetes cluster |
| Goal | The goal of this use case is to instantiate a containerized workload on a Kubernetes cluster that has the required resources to host the workload |
| Actors and roles | * SMO: The SMO initiates the instantiation process * Kubernetes API Server: O2dms termination point in Kubernetes cluster |
| Preconditions | * SMO is active and running normally * Selected Kubernetes cluster is running normally * Kubernetes API server is accessible via O2dms interface |
| Begins when | The SMO decides to instantiate a containerized workload on a Kubernetes cluster |
| Step 1 (M) | A request to instantiate a new containerized workload is received by the SMO with an identifier for that workload. (NOTE 1, NOTE 2, NOTE 3) |
| Step 2 (M) | The SMO sends the Kubernetes native resource manifests to the API Server function in the chosen Kubernetes cluster using O2dms interface by making an HTTP POST call to the respective resource URL with the Kubernetes resource manifest as payload. |
| Step 3 (M) | The API Server function authenticates the request and validates the resource manifests and creates the respective resource objects in Kubernetes cluster. (NOTE 4, NOTE 5) |
| Step 4 (M) | The API Server responds to the SMO by acknowledging the creation of the requested resource objects in the Kubernetes cluster. (NOTE 6) |
| Step 5 (O) | The SMO notifies the user about the outcome of workload instantiation request |
| Ends when | This use case ends when the containerized workload is up and running in the chosen Kubernetes cluster. |
| NOTE 1: The identifier provided is used by the SMO to get related artifacts for that workload. Examples of such artifacts includes workload descriptors (e.g., ASD and HelmCharts).  NOTE 2: If the cluster is not user selected, the SMO performs homing by matching the requirements of the containerized workload with the resources available on the Kubernetes cluster. The SMO adds information about the workload being deployed in the SMO inventory.  NOTE 3: The SMO generates parameterized Kubernetes native resource manifests from the workload artifacts using SMO internal workload deployment toolchain (e.g., Helm Client).  NOTE 4: The creation of a Kubernetes resource object results in an internal notification to the worker nodes that the deployed containerized workload needs to be instantiated.  NOTE 5: A worker node responsible for the workload execution fetches the container images from the image registery specified in the resource manifests files and starts it execution by provisioning the required number of resources.  NOTE 6: There may be a delay between the Success response from the API Server to the SMO and the actual execution of the containerized workload on the Kubernetes cluster due to several reasons including, for example, the time it takes to pull the container images from the registry, the allocation of the resources to the container and the initialization of the application inside the containers. The successful workload instantiation notification over O2dms may not represent an active workload state at the service level. Depending on the workload type (e.g., NF, xAPP, rAPP), the SMO may be required, after the O2dms success notification, to configure the deployed workload (e.g., via O1 interface) to make it available at the service level. | |

### Terminate NF Deployment

The termination use case requires the SMO to stop the execution of a containerized workload on a Kubernetes cluster and release all resources that are being used by that workload. In this O2dms profile, the termination use case is realized by deleting the Kubernetes native resource objects from the Kubernetes cluster that is running the workload. Figure 3.1.3-1 exemplifies the termination use case flow in O2dms Kubernetes profile aligned with the NF termination use case in [2].

@startuml

Autonumber

skinparam sequenceArrowThickness 1

skinparam ParticipantPadding 5

skinparam BoxPadding 10

participant User

Box "SMO" #gold

Participant NFO as NFO

end box

Box "Kubernetes® based O-Cloud" #lightseagreen

participant apiserver as "K8s® API Server"

participant worker as "K8s® worker"

End box

Note over NFO, apiserver

PRECONDITIONS:

SMO is active and running normally

K8s® cluster is active and running normally

SMO and K8s® API Server connectivity is established

End Note

== Use Case: Terminate containerized workload on K8s® cluster ==

User -> NFO : Terminate CNF (ID)

Note over NFO

The SMO determines the K8s® cluster

running the containerized workload

The SMO determines the K8s® resources

installed in the cluster for that workload

end note

loop Until all K8s® resources are deleted from the cluster

NFO -> apiserver : \*\*[O2dms]\*\* <<HTTP DELETE>> Delete K8s® resource object

apiserver -> apiserver: Delete K8s® resource

apiserver -> NFO : \*\*[O2dms]\*\* <<HTTP 200>> OK

end

Note over apiserver

K8s® API Server notifies the worker

node(s) to terminate workload execution

end note

Note over worker

K8s® worker stops the container

process and releases the resources

end note

note over NFO

SMO acquires the workload termination status

via O2dms

end note

NFO --> User : Outcome notification

== End Terminate containerized workload on K8s® cluster ==

@enduml

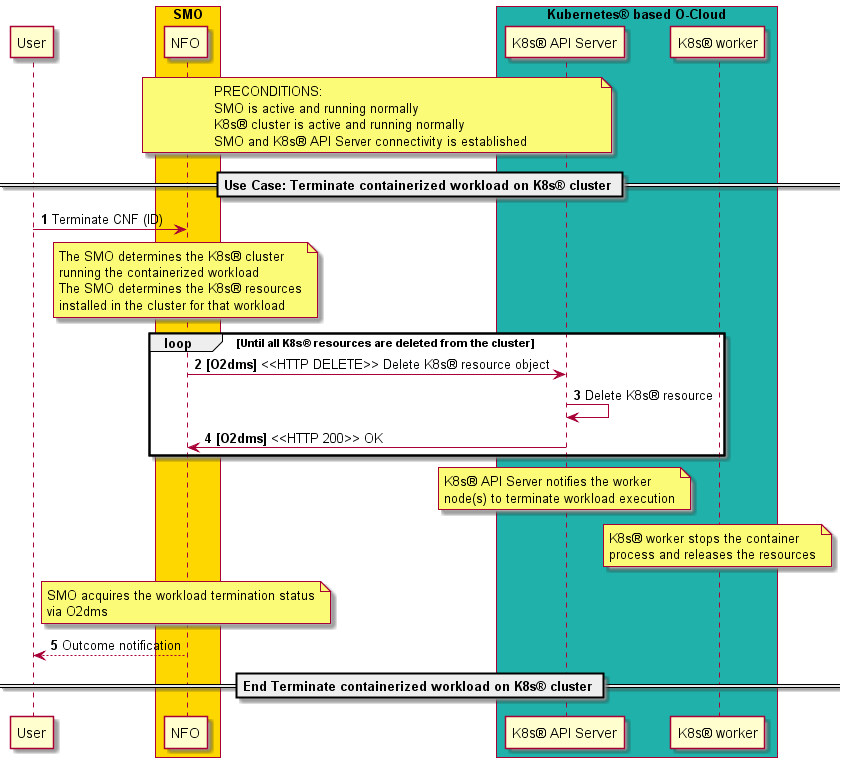


Figure 3.1.3-1: NF Deployment Termination

**Table 3.1.3-1: Termination Use Case**

|  |  |
| --- | --- |
| Use Case Stage | Evolution/Specification |
| Use Case | Termination of containerized workload on a selected Kubernetes cluster |
| Goal | The goal of this use case is to terminate a containerized workload on a Kubernetes cluster that is hosting it and release all resources used by that workload |
| Actors and roles | * SMO: The SMO initiates the deletion process * Kubernetes API Server: O2dms termination point in Kubernetes cluster |
| Preconditions | * SMO is active and running normally * Selected Kubernetes cluster is running normally * Kubernetes API server is accessible via O2dms interface |
| Begins when | The SMO decides to terminate a containerized workload on a Kubernetes cluster |
| Step 1 (M) | A request to terminate a new containerized workload is received by the SMO with an identifier for that workload. (NOTE 1, NOTE 2) |
| Step 2 (M) | The SMO requests the deletion of all Kubernetes resource objects representing the containerized workload from the API Server function in the target Kubernetes cluster by sending HTTP DELETE request(s) to the respective resource URL. |
| Step 3 (M) | The API Server function authenticates the request and deletes the requested resource objects from the Kubernetes cluster. (NOTE 3, NOTE 4) |
| Step 4 (M) | The API Server responds to the SMO by informing it about the resource objects that have been deleted from Kubernetes cluster. (NOTE 5) |
| Step 5 (O) | The SMO notifies the user about the outcome of workload termination request |
| Ends when | This use case ends when a containerized workload running in the chosen Kubernetes cluster is terminated and all resources used by that workload are released. |
| NOTE 1: The SMO may use the provided workload identifier for determining the Kubernetes cluster running that workload or may have this information available to it a priori (e.g., from inventory).  NOTE 2: The SMO determines all the Kubernetes resources manifests that collectively represent the containerized workload in the target Kubernetes cluster.  NOTE 3: The deletion of a Kubernetes resource object results in an internal notification to the worker nodes to release any resources held by that workload.  NOTE 4: The Kubernetes worker nodes stop the execution of workloads automatically for which the resource objects no longer exist in the Kubernetes cluster and release the resources.  NOTE 5: There may be a delay between the Success response from the API Server to the SMO and the actual termination of workload and release of resources in the Kubernetes cluster. The SMO may query the status of the terminated workload via O2dms before notifying the SMO user of operation outcome. | |

### Heal NF Deployment

Recovery of an NF Deployment from failures is documented in [2] clause 3.6.2, which describes an NF Deployment level healing use case based on either auto-healing support in the O-Cloud platform or SMO triggered healing over O2dms interface. Based on these healing triggers, the use case describes corrective actions that can be taken for NF Deployment healing in the O-Cloud such as restarting, replacing and/or reallocating O-Cloud resources belonging to the NF Deployment. This section describes the NF Deployment level healing use case for healing containerized workloads on a K8s cluster using O2dms K8s profile.

#### Auto Healing of NF Deployment in a Kubernetes cluster

Auto-healing of NF Deployment refers to the intrinsic capabilities of the O-Cloud platform to monitor the workloads and initiate corrective actions automatically when runtime errors/faults are detected by the O-Cloud platform. The Kubernetes platform provides auto-healing features based on continuous monitoring of the running containerized workloads (e.g., Pods) using liveness checks at the application layer. A workload designed for Kubernetes is expected to pass these liveness checks to be considered as healthy workload running in the K8s cluster. If the liveness checks fail for a workload, the Kubernetes control plane automatically replaces the failing workload with a new copy.

When the restart and replacement mechanics of Kubernetes auto-healing process are not sufficient for the full recovery of the NF Deployment, the SMO may trigger NF Deployment healing using O2dms interface.

#### SMO triggered Healing of NF Deployment in a Kubernetes cluster

Based on monitoring of the NF workloads using O1 and/or O2 interface, the SMO may detect the need and decide to heal an NF Deployment using O2dms interface. This SMO triggered healing over O2dms may occur independently from any auto-healing features running in the O-Cloud platform for healing NF workloads. In the O2dms K8s profile, the SMO triggered NF Deployment healing is realized by creating, modifying and/or deleting K8s native resource objects belonging to the NF Deployment facing failures. Figure 3.1.4-1 exemplifies the SMO triggered healing of NF Deployment using the O2dms K8s profile.

@startuml

Autonumber

Box "SMO" #gold

Participant NFO as NFO

end box

Box "Kubernetes based O-Cloud" #lightseagreen

participant apiserver as "K8s API Server"

End box

== Use case: SMO triggered NF Deployment Healing on a K8s® cluster ==

Note over NFO

The SMO actively monitors the NF deployments

for failure detection using O1/O2 interface

end Note

NFO -> NFO: SMO detects or receives a NF healing trigger

Note over NFO

SMO identifies the resources to be healed

and generates updated K8s resource manifests

end Note

group opt [K8s resources to be created]

loop Until all required K8s resources are created

NFO -> apiserver : \*\*[O2dms]\*\* <<HTTP POST>> Create K8s resource object

apiserver -> apiserver: Create K8s resource

apiserver -> NFO : \*\*[O2dms]\*\* <<HTTP 201>> CREATED

end

end

group opt [K8s resources to be modified]

loop Until all required K8s resources are updated

NFO -> apiserver : \*\*[O2dms]\*\* <<HTTP PUT/PATCH>> Update K8s resource object

apiserver -> apiserver: Update K8s resource

apiserver -> NFO : \*\*[O2dms]\*\* <<HTTP 200>> OK

end

end

group opt [K8s resources to be deleted]

loop Until all required K8s resource are deleted

NFO -> apiserver : \*\*[O2dms]\*\* <<HTTP DELETE>> Delete K8s resource object

apiserver -> apiserver: Delete K8s resource

apiserver -> NFO : \*\*[O2dms]\*\* <<HTTP 200>> OK

end

end

Note over apiserver

K8s API Server notifies the worker nodes to process the

updated resources and heal the NF deployment workload

end note

note over NFO

SMO acquires the workload status via O2dms

end note

== End: SMO triggered NF Deployment Healing on a K8s® Cluster ==

@enduml

Timeline

Description automatically generated

Figure 3.1.4-1 NF Deployment Healing

Table 3.1.4‑1 NF Deployment Healing Use Case

|  |  |
| --- | --- |
| Use Case Stage | Evolution/Specification |
| Use Case | Healing of containerized workload on a Kubernetes cluster |
| Goal | The goal of this use case is to heal a containerized workload by the SMO using O2dms K8s profile |
| Actors and roles | * SMO: The SMO initiates the healing process * Kubernetes API Server: O2dms termination point in Kubernetes cluster |
| Preconditions | The NF Deployment has been previously instantiated by the SMO |
| Begins when | The SMO decides to heal a containerized workload on a Kubernetes cluster |
| Step 1 | The SMO detects the need to heal a containerized workload based on NF deployment monitoring using O1 and/or O2 interface. (NOTE) |
| Step 2, 3, 4 | If the SMO decides to heal the NF Deployment by creating additional K8s resources, it shall send the new K8s resource objects using HTTP CREATE request(s) to the K8s API server (2). The K8s API server will create the new resources (3) and informs the SMO about the resource creation (4). |
| Step 5, 6, 7 | If the SMO decides to heal the NF Deployment by modifying its K8s resources, it shall send updated K8s resource objects using HTTP PUT/PATCH request(s) to the K8s API server (5). The K8s API server will update the NF Deployment resources (6) and informs the SMO about the resource updates (7). |
| Step 8, 9, 10 | If the SMO decides to heal the NF Deployment by deleting some of the K8s resources belong to the NF deployment, it shall delete those K8s resource objects using HTTP DELETE request(s) to the K8s API server (8). The K8s API server will delete the requested resources (9) and informs the SMO about the resource deletion (10). |
| Ends when | This use case ends when the NF Deployment running in a Kubernetes cluster is recovered from failures. |
| NOTE: The SMO determines the K8s resources belonging to the NF workload requiring updates for healing the NF Deployment. | |

### Software Upgrade of NF Deployment

This section describes the use case for SMO managed software upgrade of an NF Deployment and its realization based on O2dms K8s API profile.

#### Build-and-Replace Software Upgrade

This section illustrates a build-and-replace NF Deployment upgrade approach in which an NF Deployment using old software version is replaced by a new and independently deployed instance using newer software version. This approach utilizes a graceful or soft upgrade where the old NF Deployment is not removed from a K8s cluster until this can be done without disrupting existing traffic that the NF Deployment supports.

Upon receiving a request to upgrade software of a particular NF Deployment, the SMO instantiates a new NF Deployment according to the procedure described in section 3.1.2 (Instantiate NF Deployment). The new instance is created independently and uses newer software version of the NF Deployment. If an exception occurs during the instantiation step that causes the new NF Deployment not to be created successfully, the operation is rolled back and further actions aborted i.e., the new NF Deployment instance is removed, and the old NF Deployment continues to provide service as before.

After the new NF Deployment instantiation is successful, the SMO may trigger further actions e.g., O1 configuration of parts of the NF related to the NF Deployment. Once the new NF Deployment is up and running, the old NF Deployment is terminated according to the procedure described in section 3.1.3 (Terminate NF Deployment).

### Scale NF Deployment

#### Overview

The Scale NF Deployment use case concerns allocation of additional or deallocation of existing K8s resources to/from an NF Deployment. This enables the SMO to increase or decrease the NF Deployment’s service capacity to the required service level. The allocation or deallocation of resources for scaling and subsequent change in NF Deployment service level happens within the bounds of resources available in the K8s cluster running that NF Deployment. In the O2dms K8s profile, Scale NF Deployment may be realized either by SMO managed scaling or by automatic scaling in the K8s cluster. These approaches enable different types of scaling such as scale-out, scale-in, scale-up, and scale-down of NF Deployment.

#### SMO managed scale

The SMO may initiate scaling of NF Deployment by creating new or updating the existing K8s resources belonging to that NF Deployment in the K8s cluster. The SMO may explicitly set parameters for K8s workload resources of the NF Deployment (e.g., Deployments, StatefulSets) or manage K8s HorizontalPodAutoscaler (HPA) resource (see clause 4.3.1.4) for auto-scaling. These approaches to scaling may manifest in the following ways:

* SMO setting min/max replica count values for the HPA resource belonging to an NF Deployment. See clause 3.1.6.3 for HPA resource.
* SMO setting explicit replica counts values for K8s workload resource e.g., Deployments, Statefulsets.
* SMO creating new or updating existing NF Deployment resources for which K8s auto-scaling does not apply e.g., storage, networking etc.

Scale NF Deployment in a K8s cluster may result in allocation/deallocation of K8s resources related to compute, storage, networking, acceleration, etc. As part of the SMO managed scaling, K8s resources belonging to one or multiple DeploymentItems in the NF Deployment ASD package may be updated (e.g., executing a “helm upgrade” with updated lifecycle parameter values for one or multiple helm charts in the ASD package).

Figure 3.1.6.2-1 exemplifies the SMO managed scaling of NF Deployment using the O2dms K8s profile.

@startuml

Autonumber

Box "SMO" #gold

Participant NFO as "NFO"

end box

Box "Kubernetes based O-Cloud"#lightseagreen

participant DMS as "K8s API Server"

End box

== SMO triggered Scale NF Deployment ==

NFO -> NFO: Trigger to Scale NF Deployment

Note over NFO

SMO generates new or updates the existing

NF Deployment resources in the K8s cluster

end note

alt K8s resources to be created

loop Until all required K8s resources are created

NFO -> DMS: [O2dms] <<HTTP POST>> Create K8s resource object

DMS -> DMS: Create K8s resource

DMS -> NFO: <<HTTP 200>> OK

end loop

else K8s resources to be updated

loop Until all required K8s resources are updated

NFO -> DMS: [O2dms] <<HTTP PUT/PATCH>> Update K8s resource object

DMS -> DMS: Update K8s resource

DMS -> NFO: <<HTTP 200>> OK

end loop

end alt

Note over DMS

K8s API Server notifies the worker nodes in the

cluster to scale the NF Deployment resources

End note

== End of SMO triggered Scale NF Deployment ==

@enduml

A screenshot of a computer program

Description automatically generated

Figure 3.1.6.2-1 Scale NF Deployment

Table 3.1.6.2‑ Scale NF Deployment Use Case

|  |  |
| --- | --- |
| Use Case Stage | Evolution/Specification |
| Use Case | Scale NF Deployment on a Kubernetes cluster to desired service level. |
| Goal | The goal of this use case is to scale an NF Deployment to a desired service level by the SMO using O2dms K8s profile. |
| Actors and roles | * SMO: The SMO initiates the scaling of NF Deployment. * Kubernetes API Server: O2dms termination point in Kubernetes cluster. |
| Preconditions | The NF Deployment has been previously instantiated by the SMO. |
| Begins when | The SMO decides to scale an NF Deployment on a K8s cluster. |
| Step 1 | The SMO receives a trigger to scale an NF Deployment to a new service level. |
| (Alt choice) The SMO decides to scale the NF Deployment by creating additional K8s resources. The additional resources may belong to one or multiple of the DeploymentItems specified in the ASD package of the NF Deployment. | |
| Step 2 (M) | The SMO sends new K8s native resource objects using HTTP POST request(s) to the K8s API server. |
| Step 3 (M) | The K8s API server creates the new K8s resources for the NF Deployment. |
| Step 4 (M) | The K8s API server informs the SMO about resource creation. |
| (Alt choice) The SMO decides to scale an NF Deployment by updating/modifying its existing resources in the K8s cluster. | |
| Step 5 (M) | The SMO sends updated K8s resource objects using HTTP PUT/PATCH request(s) to the K8s API server. |
| Step 6 (M) | The K8s API server updates the NF Deployment resources for scaling the workload. |
| Step 7 (M) | The K8s API server informs the SMO about the resource updates. |
| Ends when | This use case ends when the NF Deployment running in a Kubernetes cluster is scaled to the desired service level. |

#### Kubernetes sutoscaling

K8s inherently supports automatic scaling of workloads based on autoscaling triggers specified for K8s workload resources such as K8s Deployments and StatefulSets. The K8s HorizontalPodAutoscaler resource (see clause 4.3.1.4) may be used to specify minimum and maximum replica count for K8s Pods belonging to an NF Deployment. The HorizontalPodAutoscaler resources may be created as part of NF Deployment instantiation process or as part of a later update.

To scale an NF Deployment and increase/decrease its service capacity, the K8s cluster running the NF Deployment will automatically create new K8s Pods or reduce the number of existing Pods in the K8s cluster. These changes in the number of Pods are governed by specified minimum and maximum replica count numbers in the HPA resource and resource availability in the K8s cluster. Autoscaling of the NF Deployment resources is performed by K8s within the bounds of the HPA resources. The criteria based on which Kubernetes decides when to invoke automatic scaling or how the autoscaling is achieved are outside the scope of the present document.

# O2dms Service Definitions for Kubernetes

## General

The O2dms profile for Kubernetes is based on the usage of Kubernetes native APIs and data model as O2dms interface. Kubernetes offers a native REST based, secured interface with a large set of cluster-scoped and namespace-scoped resource objects as data model. This chapter provides the details of these K8s native APIs and data model.

## Referenced Cloud-Native APIs and Data Model Solutions

### Kubernetes API overview

At high-level, the Kubernetes API [4] structure has a grouping of the managed K8s resource objects which represent K8s resource categories. The generic concepts are introduced regarding the data model of the managed K8s resource objects in Kubernetes.

#### API Structure

The Kubernetes API [4] managed objects represent a concrete instance of a K8s resource type on the Kubernetes cluster. Kubernetes leverages standard RESTful terminology to describe the API concepts:

* A K8s resource is a single instance of a K8s resource type (Note: Throughout this document, to distinguish a "resource" in context of Kubernetes from the "resources" used in a different O-RAN context, this will now be called a "K8s resource")
* A K8s resource type is the name used in the URL
* All K8s resource types have a representation in JSON (their object schema) which is called a kind
* A list of instances of a K8s resource type is known as a collection

All K8s resource types are either scoped by the Kubernetes cluster (e.g., /apis/GROUP/VERSION/\*) or to a namespace (e.g., /apis/GROUP/VERSION/namespaces/NAMESPACE/\*).

##### API Verbs

API verbs GET, CREATE, UPDATE, PATCH, DELETE and PROXY support single K8s resources only. These verbs with single K8s resource support have no support for submitting multiple K8s resources together in an ordered or unordered list or transaction. API verbs LIST and WATCH support getting multiple K8s resources, and DELETECOLLECTION supports deleting multiple K8s resources.

##### K8s resource Objects

The Kubernetes API [4] supports read and write operations on the K8s resource objects via a Kubernetes API endpoint. Kubernetes differentiates the following categories of K8s resource objects managed via their APIs:

* Workloads: objects used to manage and run OS containers on the Kubernetes cluster.
* Discovery & Loadbalancing: objects used to inter-connect the workloads into externally accessible, load-balanced services.
* Configuration & Storage: objects used to inject initialization data into the containerized applications, and to persist data that is external to the OS containers.
* Cluster: objects define how the Kubernetes cluster itself is configured.
* Metadata: objects used to configure the behavior of other K8s resources within the Kubernetes cluster.

Note: There are some Kubernetes resource objects (e.g., ClusterRoles, CustomResourceDefinitions, StorageClasses etc.) which the SMO may manage separately from the containerized workload resource management. These are FFS.

#### Data model concepts

The K8s resource objects are modelled with individual object schemas. All K8s resource objects typically have 3 components:

* **K8s resource ObjectMeta:** The metadata about the K8s resource object, such as its name, type, api version, annotations, and labels. This schema, which is common to all K8s resource types, contains fields that maybe updated both by the SMO and the Kuberneters cluster’s internal control plane.
* **K8s resourceSpec:** It is defined by the SMO and describes the desired state of system concerning the K8s resource object. Specified when creating or modifying a K8s resource object is requested.
* **K8s resourceStatus:** Provided by the Kubernetes cluster’s internal control plane and represents the current state of the system concerning the K8s resource object.

#### O2dms Procedures

The following are the list of Kubernetes procedures supported by the O2dms:

* **CREATE:** The CREATE procedure enables the SMO to instantiate a workload on the target Kubernetes cluster. This procedure requires a namespace and a specified K8s resource type.
* **READ:**
  + **GET:** The GET procedure enables the SMO to retrieve and read a specified workload from a target Kubernetes cluster. This procedure requires a namespace, a specified K8s resource type, and the name of the K8s resource.
  + **LIST:** The LIST procedure enables the SMO to list or watch a specified workload type from a target Kubernetes cluster. This procedure requires a namespace and a specified K8s resource type.
* **UPDATE:**
  + **PATCH:** The PATCH procedure enables the SMO to partially update a workload in a Kubernetes cluster. This procedure requires a namespace, a specified K8s resource type, and the name of the K8s resource.
  + **REPLACE:** The REPLACE procedure enables the SMO to replace a workload in a target Kubernetes cluster. This procedure requires a namespace, a specified K8s resource type, the name of the K8s resource.
* **DELETE:** The DELETE procedure enables the SMO to terminate a workload(s) in a target Kubernetes cluster. This procedure requires a namespace, a specified K8s resource type, and the name of the K8s resource.

## Referenced K8s resource objects

The Kubernetes APIs and resource objects are fully specified by the open-source Kubernetes project and this document adopts those definitions, APIs, and resource objects [5]. For the purposes of this document, the terms and definitions for Kubernetes API resource objects given in [5] would apply. This section provides details of relevant K8s APIs and resource objects along with recommended use and constraints, if any.

### Kubernetes Native Namespace Scoped Resource Objects

#### Kubernetes native workload resources

##### Pod

A Pod is the smallest and most basic deployable unit of functionality in a K8s cluster. A Pod resource represents a single instance of one or multiple containers that are scheduled and run on a single worker node in a K8s cluster. The containers in a Pod are managed as a single unit and share the Pod’s networking and storage resources.

Table 4.3.1.1.1-1 provides details of the standard K8s Pod resource specification.

**Table 4.3.1.1.1-1: Pod resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVerison | string | Defines the versioned schema of the Pod resource. |
| kind | string | Represents the REST resource for Pod resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the Pod resource [6]. |
| spec | PodSpec | Specifies the desired behavior of the Pod resource. |
| status | PodStatus | Represents the most recent observed status of the Pod resource. |

Table 4.3.1.1.1-2 provides relevant K8s Pod resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a K8s Pod resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.1.1-2 with K8s standard query and body parameters for Pod resource included in the HTTP request [5].

**Table 4.3.1.1.1-2: Pod resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant Pod Resource URIs** | **Supported HTTP Methods** |
| /api/v1/namespaces/{namespace}/pods | * POST: Create a Pod resource in the specified namespace * GET: List or watch Pod resources in the namespace * DELETE: Delete all Pods resources from the namespace |
| /api/v1/namespaces/{namespace}/pods/{name} | * PUT: Replace the Pod resource in the namespace * PATCH: Partial update the Pod resource in the namespace * DELETE: Delete the Pod resource from the namespace * GET: Get status of the Pod resource in the namespace |
| /api/v1/namespaces/{namespace}/pods/{name}/eviction | * POST: Terminate the running Pod instance |

NOTE: A Pod resource should not be created directly via O2dms because a Pod resource created directly is not managed by K8s control plane and it cannot repair or replace itself. A Pod resources should be created through K8s controller resources instead (e.g., Deployment, Job, Statefulset).

##### Deployment

A K8s deployment resource is used for running one or multiple identical stateless Pods in a K8s cluster. A Deployment resource ensures that the desired number of replicas of a Pod are running at any given time by replacing any failed or unresponsive Pod instances. The Deployment resource object contains the Pod resource template which specifies the details of the Pods to be created and instantiated on K8s worker node/s. Deployment resource is generally used for running stateless applications in K8s clusters since it does not provide a unique identity to the Pods it manages.

Table 4.3.1.1.2-1 provides details of the standard K8s Deployment resource specification.

**Table 4.3.1.1.2-1: Deployment resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVerison | string | Defines the versioned schema of the Deployment resource. |
| kind | string | Represents the REST resource for Deployment resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the Deployment resource [6]. |
| spec | DeploymentSpec | Specifies the desired behavior of the Deployment resource. |
| status | DeploymentStatus | Represents the most recent observed status of the Deployment resource. |

Table 4.3.1.1.2-2 provides relevant K8s Deployment resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a K8s Deployment resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.1.2-2 with K8s standard query and body parameters for Deployment resource included in the HTTP request [5].

**Table 4.3.1.1.2-2: Deployment resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant Deployment Resource URIs** | **Supported HTTP Methods** |
| /apis/apps/v1/namespaces/{namespace}/deployments | * POST: Create a Deployment resource in the specified namesapce * GET: List or watch Deployment resources in the namespace * DELETE: Delete all Deployment resources from the namespace |
| /apis/apps/v1/namespaces/{namespace}/deployments/  {name} | * PUT: Replace the Deployment resource in the namespace * PATCH: Partial update the Deployment resource in the namespace * DELETE: Delete the Deployment resource from the namespace * GET: Read the Deployment resource in the namespace |

NOTE: The creation of a Deployment resource in a K8s cluster also results in the creation of the K8s ReplicaSet resource in the same cluster. The ReplicaSet resource ensures that the number of Pods running in the cluster are equal to the number requested in the Deployment resource. A Deployment resource should be used instead of creating the ReplicaSet resource directly. The K8s Deployment resource fully substitutes the ReplicaSet functions and should be used in most cases.

##### StatefulSet

A K8s statefulset resource is used for running one or multiple stateful Pods that maintain unique and persistent identities and hostnames during their lifetime. The Pods belonging to a StatefulSet resource are created from the same Pod spec but maintain their individual identities across any rescheduling events caused by Pod failures. In addition, statefulsets provide an ordered and graceful deployment and scaling of Pods in K8s clusters. By default, the Pod deployment for statefulsets is done in sequential order and termination in reverse order but this behavior can be overridden. For state persistency, StatefulSet resources are generally paired with persistent storage volumes and this paired relationship is maintained across Pod rescheduling events.

Table 4.3.1.1.3-1 provides details of the standard K8s StatefulSet resource specification.

**Table 4.3.1.1.3-1: StatefulSet resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVerison | string | Defines the versioned schema of the StatefulSet resource. |
| kind | string | Represents the REST resource for StatefulSet resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the StatefulSet resource [6]. |
| spec | StatefulSetSpec | Specifies the desired identities of the Pods in the StatefulSet resource. |
| status | StatefulSetStatus | Represents the most recent observed status of the Pods in the StatefulSet. |

Table 4.3.1.1.3-2 provides relevant K8s StatefulSet resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a K8s StatefulSet resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.1.3-2 with K8s standard query and body parameters for StatefulSet resource included in the HTTP request [5].

**Table 4.3.1.1.3-2: StatefulSet resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant StatefulSet Resource URIs** | **Supported HTTP Methods** |
| /apis/apps/v1/namespaces/{namespace}/statefulsets | * POST: Create a StatefulSet resource in the specified namespace * GET: List or watch all StatefulSet resources in the namespace * DELETE: Delete all StatefulSet resources from the namespace |
| /apis/apps/v1/namespaces/{namespace}/statefulsets/  {name} | * PUT: Replace the StatefulSet resource in the namespace * PATCH: Partial update the StatefulSet resource in the namespace * DELETE: Delete the StatefulSet resource from the namespace * GET: Read the StatefulSet resource in the namespace |

##### DaemonSet

A K8s DaemonSet resource is used for running a Pod instance on all or a subset of the worker nodes in a K8s cluster. If the cluster scales up or down i.e., worker nodes are added ore removed from the cluster, the daemonset Pods are scaled accordingly to ensure that one Pod per worker node requirement is met.

Table 4.3.1.1.4-1 provides details of the standard K8s DaemonSet resource specification.

**Table 4.3.1.1.4-1: DaemonSet resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVerison | string | Defines the versioned schema of the DaemonSet resource. |
| kind | string | Represents the REST resource for DaemonSet resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the DaemonSet resource [6]. |
| spec | DaemonSetSpec | Specifies the desired behavior of the DaemonSet resource. |
| status | DaemonSetStatus | Represents the most recent observed status of the DaemonSet resource. |

Table 4.3.1.1.4-2 provides relevant K8s DaemonSet resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a K8s DeamonSet resource in a K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.1.4-2 with K8s standard query and body parameters for DeamonSet resource included in the HTTP request [5].

**Table 4.3.1.1.4-2: DaemonSet resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant DaemonSet Resource URIs** | **Supported HTTP Methods** |
| /apis/apps/v1/namespaces/{namespace}/daemonsets | * POST: Create a DaemonSet resource in specified namespace * GET: List or watch all DaemonSet resources in the namespace * DELETE: Delete all DaemonSet resources from the namespace |
| /apis/apps/v1/namespaces/{namespace}/daemonsets/  {name} | * PUT: Replace the DaemonSet resource in the namespace * PATCH: Partial update the DaemonSet resource in the namespace * DELETE: Delete the DaemonSet resource from the namespace * GET: Read the DaemonSet resource in the namespace |

##### Job

A K8s Job resource is used for running workloads in K8s clusters that run to completion and are terminated after completion. A Job resource can create one or more Pods and execute them until a specified number of Jobs are successfully terminated. A Job resource tracks successful completions of Pod executions and upon reaching the desired number of completions, the Job is considered complete.

Table 4.3.1.1.5-1 provides details of the standard K8s Job resource specification.

**Table 4.3.1.1.5-1: Job resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVerison | string | Defines the versioned schema of the Job resource. |
| kind | string | Represents the REST resource for Job resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the Job resource [6]. |
| spec | JobSpec | Specifies the desired behavior of the Job resource. |
| status | JobStatus | Represents the most recent observed status of the Job resource. |

Table 4.3.1.1.5-2 provides relevant K8s Job resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a Job resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.1.5-2 with K8s standard query and body parameters for Job resource included in the HTTP request [5].

**Table 4.3.1.1.5-2: Job resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant Job Resource URIs** | **Supported HTTP Methods** |
| /apis/batch/v1/namespaces/{namespace}/jobs | * POST: Create a Job resource in specified namespace * GET: List or watch all Job resources in the namespace * DELETE: Delete all Job resources from the namespace |
| /apis/batch/v1/namespaces/{namespace}/jobs/  {name} | * PUT: Replace the Job resource in the namespace * PATCH: Partial update the Job resource in the namespace * DELETE: Delete the Job resource from the namespace * GET: Read the Job resource in the namespace |

##### CronJob

A K8s CronJob resource is used for running K8s Jobs on a pre-defined time schedule provided in Linux/Unix Cron format. The time schedule of the CronJob resource follows the timezone set for the K8s native kube-controller-manager function.

Table 4.3.1.1.6-1 provides details of the standard K8s CronJob resource specification.

**Table 4.3.1.1.6-1: CronJob resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVerison | string | Defines the versioned schema of the CronJob resource. |
| kind | string | Represents the REST resource for CronJob resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the CronJob resource [6]. |
| spec | CronJobSpec | Specifies the desired behavior of the CronJob resource. |
| status | CronJobStatus | Represents the most recent observed status of the CronJob resource. |

Table 4.3.1.1.6-2 provides relevant K8s CronJob resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a CronJob resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.1.6-2 with K8s standard query and body parameters for CronJob resource included in the HTTP request [5].

**Table 4.3.1.1.6-2: CronJob resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant CronJob Resource URIs** | **Supported HTTP Methods** |
| /apis/batch/v1/namespaces/{namespace}/cronjobs | * POST: Create a CronJob resource in specified namespace * GET: List or watch all CronJob resources in the namespace * DELETE: Delete all CronJob resources from the namespace |
| /apis/batch/v1/namespaces/{namespace}/cronjobs/  {name} | * PUT: Replace the CronJob resource in the namespace * PATCH: Partial update the CronJob resource in the namespace * DELETE: Delete the CronJob resource from the namespace * GET: Read the CronJob resource in the namespace |

#### Kubernetes native discovery and load balancing resources

##### Service

A K8s Service resource is used for exposing a containerized workload/application running in a K8s cluster to internal and external clients. The Service resource also abstracts the ephemeral nature of Pods and containers by creating a constant point of entry to the group of Pods that provide the same functionality. Each Service gets an IP address and port number that does not change during the lifetime of the Service resource in the K8s cluster. In addition, the Service resource provides load-balancing across all Pod instances that provide the same functionality. Kubernetes supports five types of Service resources identified as ClusterIP, NodePort, LoadBalancer, ExternalName and Headless Service with ClusterIP being the default Service type.

Table 4.3.1.2.1-1 provides details of the standard K8s Service resource specification.

**Table 4.3.1.2.1-1: K8s Service resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVersion | string | Defines the versioned schema of the Service resource. |
| kind | string | Represents the REST resource for Service resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the Service resource [6] |
| spec | ServiceSpec | Specifies the desired behavior and type of the Service resource |
| status | ServiceStatus | Read only. Represents the most recent observed status of the Service resource |

Table 4.3.1.2.1-2 provides relevant K8s Service resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a Service resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.2.1-2 with K8s standard query and body parameters for Service resource included in the HTTP request [5].

**Table 4.3.1.2.1-2: K8s Service resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant Service Resource URIs** | **Supported HTTP Methods** |
| /api/v1/namespaces/{namespace}/services | * POST: Create a Service resource in the specified namespace * GET: List or watch all Service resources in the namespace * DELETE: Delete all Service resources from the namespace |
| /api/v1/namespaces/{namespace}/services/{name} | * PUT: Replace the Service resource in the namespace * PATCH: Partial update the Service resource in the namespace * DELETE: Delete the Service resource from the namespace * GET: Read the Service resource in the namespace |

##### Ingress

A K8s Ingress resource is used for defining HTTP(s) routing rules that control external access to applications running inside the K8s cluster. A K8s Ingress resource can be associated with one or more K8s Service resources (each of which can expose a different set of Pods) and offers load-balancing and SSL termination features. The Ingress resource can assign a specific externally reachable URL to each Service resource associated with it.

Table 4.3.1.2.2-1 provides details of the standard K8s Ingress resource specification.

**Table 4.3.1.2.2-1: K8s Ingress resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVersion | string | Defines the versioned schema of the Ingress resource. |
| kind | string | Represents the REST resource for Ingress resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the Ingress resource [6] |
| spec | IngressSpec | Specifies the desired behavior of the Ingress resource |
| status | IngressStatus | Represents the most recent observed status of the Ingress resource |

Table 4.3.1.2.2-2 provides relevant K8s Ingress resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage an Ingress resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.2.2-2 with K8s standard query and body parameters for Ingress resource included in the HTTP request [5].

**Table 4.3.1.2.2-2: K8s Ingress resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant Ingress Resource URIs** | **Supported HTTP Methods** |
| /apis/networking.k8s.io/v1/namespaces/{namespace}/ingresses | * POST: Create an Ingress resource in the specified namespace * GET: List or watch all Ingress resources in the namespace * DELETE: Delete all Ingress resources from the namespace |
| /apis/networking.k8s.io/v1/namespaces/{namespace}/ingresses/{name} | * PUT: Replace the Ingress resource in the namespace * PATCH: Partial update the Ingress resource in the namespace * DELETE: Delete the Ingress resource from the namespace * GET: Read the specified Ingress resource in the namespace |

NOTE: For an Ingress resource to work, an Ingress controller (i.e., an additional controller to the default K8s controllers) shall be running inside the K8s cluster. K8s does not offer a default ingress controller and leaves the choice of ingress controller to the cluster administrator. In the ORAN context, the K8s cluster may have a default ingress controller installed at the time of cluster creation, or the SMO may decide to extend the K8s cluster with any specific ingress controller after cluster creation using O2dms interface.

##### NetworkPolicy

A K8s NetworkPolicy resource is used to control traffic flow to/from a K8s workloads at the IP address or port level using ingress and egress rules. These rules may apply to other K8s Pods, Namespaces, or generic IP blocks to restrict or allow traffic from these entities. Selectors and IP blocks (CIDR ranges) are used to enforce the NetworkPolicy constraints specified in the NetworkPolicy resource specification.

Table 4.3.1.2.3-1 provides details of the standard K8s NetworkPolicy resource specification.

**Table 4.3.1.2.3-1: NetworkPolicy resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVerison | string | Defines the versioned schema of the NetworkPolicy resource. |
| kind | string | Represents the REST resource for NetworkPolicy. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the NetworkPolicy resource [6] |
| spec | NetworkPolicySpec | Specifies the desired behavior of the NetworkPolicy resource and the ingress and/or egress traffic rules to be enforced in the cluster. |
| status | NetworkPolicyStatus | Describes the current state of the NetworkPolicy |

Table 4.3.1.2.3-2 provides relevant K8s NetworkPolicy resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a K8s NetworkPolicy resource in a K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.2.3-2 with K8s standard query and body parameters for NetworkPolicy resource included in the HTTP request [5].

**Table 4.3.1.2.3-2: NetworkPolicy resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant NetworkPolicy Resource URIs** | **Supported HTTP Methods** |
| /apis/networking.k8s.io/v1/namespaces/{namespace}/ networkpolicies | * POST: Create a new NetworkPolicy resource in the specified namespace * GET: List or watch all NetworkPolicy resources in the namespace * DELETE: Delete all NetworkPolicy resources from the namespace |
| /apis/networking.k8s.io/v1/namespaces/{namespace}/ networkpolicies/{name} | * PUT: Replace the NetworkPolicy resource in the namespace * PATCH: Partial update the NetworkPolicy resource in the namespace * DELETE: Delete the NetworkPolicy resource from the namespace * GET: Read the NetworkPolicy resource in the namespace |

NOTE: The functionality of the NetworkPolicy resource depends on the presence of a compatible K8s network plugin that supports K8s network policies. Without a supporting network plugin, the NetworkPolicy resource creation will have no impact on the ingress and egress traffic to/from a K8s workload.

#### Kubernetes native storage and workload configuration resources

* + - * 1. ConfigMap

A K8s ConfigMap resource is used for storing non-confidential data in the form of key-value pairs that can be read as runtime configurations by Pods in a K8s cluster. ConfigMap resource can be used to store configuration files, environment variables or command line arguments for workloads to use at runtime. A ConfigMap resource decouples the workload containers from environment specific configuration data thereby making the K8s Pods easily portable across the worker nodes in a K8s cluster. The data stored in a ConfigMap cannot exceed 1 MiB.

Table 4.3.1.3.1-1 provides details of the standard K8s ConfigMap resource specification.

**Table 4.3.1.3.1-1: K8s ConfigMap resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVersion | string | Defines the versioned schema of the ConfigMap resource. |
| kind | string | Represents the REST resource for ConfigMap resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the ConfigMap resource [6] |
| data | object | Contains the configuration data in key-value pairs. Keys shall be alphanumeric characters and ‘- ‘, ‘\_’ or ‘.’ Values shall be UTF-8 byte sequence. Keys in the data field shall not overlap with keys in binaryData field to pass validation checks at ConfigMap creation time. |
| binaryData | object | Contains binary data in key-value pairs. Keys shall be alphanumeric characters and ‘-‘, ‘\_’, or ‘.’ Values can contain byte sequences that are not in UTF-8 range. Keys in the binaryData field shall not overlap with keys in the data field to pass validation checks at ConfigMap creation time. |
| immutable | boolean | When true, the data stored in the ConfigMap cannot be updated. When false, the data can be modified at any time. |

Table 4.3.1.3.1-2 provides relevant K8s ConfigMap resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage an ConfigMap resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.3.1-2 with K8s standard query and body parameters for ConfigMap resource included in the HTTP request [5].

**Table 4.3.1.3.1-2: K8s ConfigMap resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant ConfigMap Resource URIs** | **Supported HTTP Methods** |
| /api/v1/namespaces/{namespace}/configmaps | * POST: Create a ConfigMap resource in the specified namespace * GET: List or watch all ConfigMap resources in the namespace * DELETE: Delete all ConfigMap resources from the namespace |
| /api/v1/namespaces/{namespace}/configmaps/{name} | * PUT: Replace the ConfigMap resource in the namespace * PATCH: Partial update the ConfigMap resource in the namespace * DELETE: Delete the ConfigMap resource from the namespace * GET: Read the specified ConfigMap resource in the namespace |

##### Secret

A K8s Secret resource is used for storing sensitive or confidential data in key value pairs format. Secrets are like ConfigMaps but specifically intended to be used for storing confidential data such as passwords, OAuth tokens, SSH keys etc. By default, data stored in Secrets is not encrypted in a K8s cluster.

Table 4.3.1.3.2-1 provides details of the standard K8s Secret resource specification.

**Table 4.3.1.3.2-1: K8s Secret resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVersion | string | Defines the versioned schema of the Secret resource. |
| kind | string | Represents the REST resource for Secret resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the Secret resource [6] |
| data | object | Contains the secret data in key-value pairs. Keys shall be alphanumeric characters and ‘- ‘, ‘\_’ or ‘.’ Values are serialized form of the secret data specified as based64 encoded string. |
| stringData | object | Contains non-binary secret data in string form. |
| immutable | boolean | When true, the data stored in the ConfigMap cannot be updated. When false, the data can be modified at any time. |
| type | String | A non-constrained string value. Used to facilitate programmatic handling of secret data. Defaults to Opaque (arbitrary user-defined data) |

Table 4.3.1.3.2-2 provides relevant K8s Secret resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a Secret resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.3.2-2 with K8s standard query and body parameters for Secret resource included in the HTTP request [5].

**Table 4.3.1.3.2-2: K8s Secret resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant Secret Resource URIs** | **Supported HTTP Methods** |
| /api/v1/namespaces/{namespace}/secrets | * POST: Create a Secret resource in the specified namespace * GET: List or watch all Secret resources in the namespace * DELETE: Delete all Secret resources from the namespace |
| /api/v1/namespace/{namespace/secrets/{name} | * PUT: Replace the Secret resource in the namespace * PATCH: Partial update the Secret resource in the namespace * DELETE: Delete the Secret resource from the namespace * GET: Read the specified Secret resource in the namespace |

* + - * 1. PersistentVolumeClaim

The K8s PersistentVolumeClaim resource is used for requesting and claiming a K8s PersistentVolume resource. The PersistentVolume may pre-exist in the K8s cluster or created dynamically in response to the creation of PersistentVolumeClaim resource. A PersistentVolumeClaim provides a way to request a persistent storage for Pods without knowing the underlying storage technology. Pods can use PersistentVolumeClaims as volumes and mount them as part of the Pod specification.

Table 4.3.1.3.3-1 provides details of the standard K8s PersistentVolumeClaim resource specification.

**Table 4.3.1.3.3-1: K8s PersistentVolumeClaim resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVersion | string | Defines the versioned schema of the PersistentVolumeClaim resource. |
| kind | string | Represents the REST resource for PersistentVolumeClaim resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the PersistentVolumeClaim resource [6] |
| spec | PersistentVolumeClaimSpec | Specifies the desired characteristics of the persistent volume requested for the Pod. |
| status | PersistentVolumeClaimStatus | Read only. Represents the most recent observed status of the PersistentVolumeClaim resource. |

Table 4.3.1.3.3-2 provides relevant K8s PersistentVolumeClaim resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a PersistentVolumeClaim resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.3.3-2 with K8s standard query and body parameters for PersistentVolumeClaim resource included in the HTTP request [5].

**Table 4.3.1.3.3-2: K8s PersistentVolumeClaim resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant PersistentVolumeClaim Resource URIs** | **Supported HTTP Methods** |
| /api/v1/namespaces/{namespace}/ persistentvolumeclaims | * POST: Create a PersistentVolumeClaim resource in the specified namespace * GET: List or watch all PersistentVolumeClaim resources in the namespace * DELETE: Delete all PersistentVolumeClaim resources from the namespace |
| /api/v1/namespaces/{namespace}/ persistentvolumeclaims/{name} | * PUT: Replace the PersistentVolumeClaim resource in the namespace * PATCH: Partial update the PersistentVolumeClaim resource in the namespace * DELETE: Delete the PersistentVolumeClaim resource from the namespace * GET: Read the PersistentVolumeClaim resource in the namespace |

#### Kubernetes native metadata and cluster configuration resources

##### LimitRange

A K8s LimitRange resource is used to set constraints on the resource utilization of Pods and containers in a namespace. By default, a container can use as much compute resources as it needs in the namespace and that can lead to resource contention problems among Pods and containers that share the namespace and may belong to different priority classes. A LimitRange resource an enforce defaults or min-max values for different resources in the namespace (e.g., CPU, memory, storage) at Pod or container level.

Table 4.3.1.4.1-1 provides details of the standard K8s LimitRange resource specification.

**Table 4.3.1.4.1-1: K8s LimitRange resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVersion | string | Defines the versioned schema of the LimitRange resource. |
| kind | string | Represents the REST resource for LimitRange resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the LimitRange resource [6] |
| spec | LimitRangeSpec | Specifies the limits that will be enforced on resources in a namespace. |

Table 4.3.1.4.1-2 provides relevant K8s LimitRange resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a LimitRange resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.4.1-2 with K8s standard query and body parameters for LimitRange resource included in the HTTP request [5].

**Table 4.3.1.4.1-2: K8s LimitRange resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant LimitRange Resource URIs** | **Supported HTTP Methods** |
| /api/v1/namespaces/{namespace}/limitranges | * POST: Create a LimitRange resource in the specified namespace * GET: List or watch all LimitRange resources in the namespace * DELETE: Delete all LimitRange resources from the namespace |
| /api/v1/namespaces/{namespace}/limitranges/  {name} | * PUT: Replace the LimitRange resource in the namespace * PATCH: Partial update the LimitRange resource in the namespace * DELETE: Delete the LimitRange resource from the namespace * GET: Read the LimitRange resource in the namespace |

##### ResourceQuota

A K8s ResourceQuota resource is used to set limits on the total/aggregate resource consumption in a namespace. A ResourceQuota object can specify limits for other K8s resource objects (by resource type) and compute resources (e.g., memory, CPU, storage). A single K8s namespace can have multiple ResourceQuota resources with unique names for limiting different types of namespace scoped resources. The ResourceQuota specified limits are enforced at K8s resource creation time and have no impact on exsting Pods and resources in the namespace.

Table 4.3.1.4.2-1 provides details of the standard K8s ResourceQuota resource specification.

**Table 4.3.1.4.2-1: ResourceQuota resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVersion | string | Defines the versioned schema of the ResourceQuota resource. |
| kind | string | Represents the REST resource for ResourceQuota. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the ResourceQuota resource [6] |
| spec | ResourceQuotaSpec | Specifies the desired quota of K8s resource objects and/or compute resources imposed on the namespace |
| status | ResourceQuotaStatus | Defines the actual enforced quota and its current usage |

Table 4.3.1.4.2-2 provides relevant K8s ResourceQuota resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a K8s ResourceQuota resource in a K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.4.2-2 with K8s standard query and body parameters for ResourceQuota resource included in the HTTP request [5].

**Table 4.3.1.4.2-2: ResourceQuota resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant ResourceQuota Resource URIs** | **Supported HTTP Methods** |
| /api/v1/namespaces/{namespace}/resourcequotas | * POST: Create a new ResourceQuota resource in the specified namespace * GET: List or watch all ResourceQuota resources in the namespace * DELETE: Delete all ResourceQuota resources from the namespace |
| /api/v1/namespaces/{namespace}/resourcequotas/  {name} | * PUT: Replace the ResourceQuota resource in the namespace * PATCH: Partial update the ResourceQuota resource in the namespace * DELETE: Delete the ResourceQuota resource from the namespace * GET: Read the ResourceQuota resource in the namespace |

##### HorizontalPodAutoscaler

A K8s HorizontalPodAutoscaler resource is used to automatically scale up or down, the number of Pod instances belonging to a workload resource that supports horizontal scaling (e.g., Deployments and StatefulSets). The scale-up and scale-down levels are controlled by defined minimum and maximum values in the HorizontalPodAutoscaler resource. The HorizontalPodAutoscaler resource works by changing the “scale” property of a workload resource e.g., Deployment, based on observed metrics such as CPU and Memory utilization.

NOTE: For the K8s cluster to horizontally scale a workload based on the metrics specified in the K8s HorizontalPodAutoscaler resource, it should have a metrics collection service running in the cluster.

Table 4.3.1.4.3-1 provides details of the standard K8s HorizontalPodAutoscaler resource specification.

**Table 4.3.1.4.3-1: HorizontalPodAutoscaler resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVersion | string | Defines the versioned schema of the HorizontalPodAutoscaler resource. |
| kind | string | Represents the REST resource for HorizontalPodAutoscaler resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the HorizontalPodAutoscaler resource [6]. |
| spec | HorizontalPodAutoscalerSpec | Specifies the min-max replica count and behavior of the HorizontalPodAutoscaler resource. |
| status | HorizontalPodAutoscalerStatus | Represents the most recent observed status of the HorizontalPodAutoscaler resource. |

Table 4.3.1.4.3-2 provides relevant K8s HorizontalPodAutoscaler resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a K8s HorizontalPodAutoscaler resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.4.3-2 with K8s standard query and body parameters for HorizontalPodAutoscaler resource included in the HTTP request [5].

**Table 4.3.1.4.3-2: HorizontalPodAutoscaler resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant HorizontalPodAutoscaler Resource URIs** | **Supported HTTP Methods** |
| /apis/autoscaling/v2/namespaces/{namespace}/  horizontalpodautoscalers | * POST: Create a HorizontalPodAutoscaler resource in the specified namespace * GET: List or watch HorizontalPodAutoscaler resources in the namespace * DELETE: Delete all HorizontalPodAutoscaler resources from the namespace |
| /apis/autoscaling/v2/namespaces/{namespace}/  horizontalpodautoscalers/{name} | * PUT: Replace the HorizontalPodAutoscaler resource in the namespace * PATCH: Partial update the HorizontalPodAutoscaler resource in the namespace * DELETE: Delete the HorizontalPodAutoscaler resource from the namespace * GET: Read the HorizontalPodAutoscaler resource in the namespace |

##### PodDisruptionBudget

A K8s PodDisruptionBudget resource is used to ensure that a specific number of Pod instances belonging to a workload resource (e.g., Deployments, StatefulSets) are available in the cluster in a healthy running state in the event of planned or unplanned Pod disruptions. A K8s PodDisruptionBudget resource ensures that the application or service offered by the Pods remains available in the event of concurrent disruptions due to Pod termination events. The number of healthy Pod instances required during disruption events can be specified as a minimum-available or maximum-unavialable number.

Table 4.3.1.4.4-1 provides details of the standard K8s PodDisruptionBudget resource specification.

**Table 4.3.1.4.4-1: PodDisruptionBudget resource specification [5]**

|  |  |  |
| --- | --- | --- |
| **Resource Field** | **Resource Field Type** | **Resource Field Description** |
| apiVersion | string | Defines the versioned schema of the PodDisruptionBudget resource. |
| kind | string | Represents the REST resource for PodDisruptionBudget resource. The K8s API Server may infer this from the endpoint the client submits the request to. The value cannot be updated. |
| metadata | ObjectMeta | Standard metadata of the PodDisruptionBudget resource [6]. |
| spec | PodDisruptionBudgetSpec | Specifies the desired behavior of PodDisruptionBudge resource. |
| status | PodDisruptionBudgetStatus | Represents the most recent observed status of the PodDisruptionBudget resource. |

Table 4.3.1.4.4-2 provides relevant K8s PodDisruptionBudget resource URIs and supported HTTP methods using O2dms interface. For the SMO to lifecycle manage a K8s PodDisruptionBudget resource in a chosen K8s cluster using O2dms K8s profile, it shall use the URIs, and HTTP methods listed in Table 4.3.1.4.4-2 with K8s standard query and body parameters for PodDisruptionBudget resource included in the HTTP request [5].

**Table 4.3.1.4.4-2: PodDisruptionBudget resource URIs and HTTP methods supported using O2dms K8s profile**

|  |  |
| --- | --- |
| **Relevant PodDisruptionBudget Resource URIs** | **Supported HTTP Methods** |
| /apis/policy/v1/namespaces/{namespace}/  poddisruptionbudgets | * POST: Create a PodDisruptionBudget resource in the specified namespace * GET: List or watch PodDisruptionBudget resources in the namespace * DELETE: Delete all PodDisruptionBudget resources from the namespace |
| /apis/policy/v1/namespaces/{namespace}/  poddisruptionbudgets/{name} | * PUT: Replace the PodDisruptionBudget resource in the namespace * PATCH: Partial update the PodDisruptionBudget resource in the namespace * DELETE: Delete the PodDisruptionBudget resource from the namespace * GET: Read the PodDisruptionBudget resource in the namespace |

### Kubernetes Native Cluster Scoped Resource Objects

The Kubernetes native cluster scoped resource objects are not included in the present version of the document.

# Annex (informative): Change History

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision** | **Description** |
| 2022.03.28 | 01.00 | Final version 01.00 |
| 2022.07.26 | 02.00 | Final version 02.00 |
| 2023.03.24 | 03.00 | Final version 03.00 |
| 2023.07.31 | 04.00 | Final version 04.00 |
| 2024.03.19 | 04.00.01 | Implemented CRs ERI-0026, ERI-0028 |
| 2024.04.01 | 05.00 | Final version 05.00 |