 O-RAN.WG6.O2-IMS-K8s-R003-v01.00

Technical Report

O-RAN Work Group 6

O2 IMS K8s Technical Report

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 document 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 document for your personal use, or copy the material of this document 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

Foreword 3

Modal verbs terminology 3

1 Scope 4

2 References 4

2.1 Informative references 4

3 Definition of terms, symbols and abbreviations 5

3.1 Terms 5

3.2 Symbols 5

3.3 Abbreviations 5

4 Study of O2-IMS K8s interface 5

4.1 Context 5

4.2 Objectives 5

4.3 Declarative K8s API approach 6

4.3.1 Kubernetes Object Management techniques 6

4.3.2 Declarative object management benefits 6

4.3.3 Industry Adoption 7

4.4 O2-IMS Services Analysis 7

4.4.1 Introduction 7

4.4.2 Inventory Service 7

4.4.3 Provisioning Service 8

4.4.4 Lifecycle Management Service 9

4.4.5 Software Management Services 9

4.4.6 Monitoring Service 9

4.5 Proposal for O2-IMS K8s interface support 10

4.5.1 Introduction 10

4.5.2 URI compatibility 10

4.5.3 Example of K8s API for Cluster Template 11

4.5.4 Proposal 11

4.6 Impact on WG6 Specifications and Use Cases 12

History 12

# Foreword

This Technical Report (TR) has been produced by O-RAN Alliance.

# Modal verbs terminology

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

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

# 1 Scope

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

version xx.yy.zz

where:

xx: the first digit-group is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc. (the initial approved document will have xx=01). Always 2 digits with leading zero if needed.

yy: the second digit-group is incremented when editorial only changes have been incorporated in the document. Always 2 digits with leading zero if needed.

zz: the third digit-group included only in working versions of the document indicating incremental changes during the editing process. External versions never include the third digit-group. Always 2 digits with leading zero if needed.

The present document specifies a study of the applicability of a K8s declarative interface for O-RAN O2 IMS interface.

# 2 References

## 2.1 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, O-RAN 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.

1. Kubernetes® API, online <https://kubernetes.io/docs/reference/kubernetes-api/>
2. Kubernetes® Object Management, online: <https://kubernetes.io/docs/concepts/overview/working-with-objects/object-management/>
3. Kubernetes® CLI book, online: <https://kubectl.docs.kubernetes.io/>
4. The Cloud-Native Telco Market Perception Report, Telecom TV, online: <https://www.telecomtv.com/content/cloud-native-telco-report-sep-23/>
5. Cloud Native Definition, v1.0, online: <https://github.com/cncf/toc/blob/main/DEFINITION.md>
6. GitOps Definition, online: <https://opengitops.dev/>
7. Kubernetes Resource URIs, online, <https://kubernetes.io/docs/reference/using-api/api-concepts/#resource-uris>
8. Kubernetes CRD Validation Rules, online, <https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/#validation-rules>
9. O-RAN-WG6.O2IMS-INTERFACE: “O2ims Interface Specification”
10. Helm, online, <https://helm.sh/>
11. Git, online, <https://git-scm.com/>
12. Kustomize, online, <https://kustomize.io/>

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.

# 3 Definition of terms, symbols and abbreviations

## 3.1 Terms

For the purposes of the present document, the following terms apply:

**GitOps:** Set of practices to manage infrastructure and application configurations using Git as a single source of truth ensuring the operational state of systems is consistent with the intended state.

NOTE: This is defined by the CNCF GitOps Working Group [6]

## 3.2 Symbols

For the purposes of the present document, the following symbols apply:

## 3.3 Abbreviations

For the purposes of the present document, the [following] abbreviations [given in ... and the following] apply:

# 4 Study of O2-IMS K8s interface

## 4.1 Context

The background for this study arises from the increasing adoption of cloud-native network functions and their impact on infrastructure management systems. Cloud-native principles advocate for declarative-based management for network functions, wherein the desired end state is described, rather than specifying the sequence of steps to achieve that state. The shift towards declarative models has not been fully explored for infrastructure management services within the context of the O2 interface.

As of specification v04.00, O2-IMS interfaces follow an imperative model approach for infrastructure management. This approach does not enable direct cloud native integration with Kubernetes which is a prominent container orchestrator widely used in the industry.

The study’s findings are expected to provide insights into the feasibility and benefits of adopting a declarative K8s API approach for the O2-IMS interface. This analysis will assist WG6 in making informed choices regarding the evolution of O2-IMS ecosystem. By understanding the impact and potential modifications, the study aims to accelerate the adoption of O2-IMS and leverage industry best practices.

## 4.2 Objectives

The main objective of the present Technical Report is to conduct a comprehensive analysis of the O2-IMS interface with a focus on exploring the feasibility, drawback and benefits of adopting a declarative K8s API approach.

To achieve this goal, the study is structured around the following specific objects:

* Declarative K8s API approach
* O2-IMS services analysis
* Proposal for O2-IMS K8s interface support
* Impact on WG6 Specifications and Use Cases

## 4.3 Declarative K8s API approach

### 4.3.1 Kubernetes Object Management techniques

As defined in the Kubernetes Object Management [2] documentation, three management techniques exist:

* Imperative commands
  + “A user operates directly on live objects in a cluster. The user provides operations to the kubectl command as arguments or flags.”
  + “This is the recommended way to run a one-off task in a cluster; it operates directly on live objects and provides no history of previous configurations.”
* Imperative object configuration
  + “The kubectl command specifies the operation (create, replace, etc.), optional flags, and at least one file name. The file specified contains a full definition of the object in YAML or JSON format.”
* Declarative object configuration
  + “A user operates on object configuration files stored locally; however, the user does not define the operations to be taken on the files. Create, update, and delete operations are automatically detected per object by kubectl. This enables working on directories, where different operations might be needed for different objects.”

The Kubernetes documentation exemplifies and defines advantage and disadvantage for each of these techniques.

### 4.3.2 Declarative object management benefits

Declarative Kubernetes (K8s) API is one of the fundamental concepts that Kubernetes is built upon. As depicted in the previous section, it refers to the practice of defining the desired state of applications in a declarative manner, rather than imperatively specifying the steps to achieve that state. There are several benefits to using a declarative approach with the Kubernetes API:

* **Desired State Management**: Declarative APIs enable specifying what the system is expected to look like (the desired state) rather than how to get there (the imperative steps). Kubernetes controllers continuously work to reconcile the current state with the desired state, automatically making adjustments as needed. This simplifies operations of keeping resources in sync with the intention and reduces the risk of configuration drift.
* **Resilient**: Declarative object configuration enables Kubernetes to automatically detect and recover from failures. If a pod or other resource goes down or becomes unhealthy, Kubernetes will automatically create a new instance to maintain the desired state.
* **Idempotence**: In a declarative API, applying the same configuration multiple times have the same result as applying it once. This idempotence property reduces the risk of unexpected changes or side effects when managing resources.
* **Scalability**: Objects created declaratively are often easier to scale because multiple instances of the same resource using the same configuration template can be defined. Kubernetes can then create and manage these instances as needed.
* **Version Control and Auditing**: Declarative configurations can be stored in version control systems like Git [11], allowing for versioning, history tracking, and easy collaboration. This helps with auditing and troubleshooting as the evolution of the configuration is visible over time.
* **Predictable Rollouts and Rollbacks**: With declarative APIs, a new desired state can be specified, and Kubernetes will manage the rollout and rollback of resources to achieve that state. This makes it easier to perform updates and handle failures gracefully.
* **Portability**: Declarative configurations are often agnostic to the underlying infrastructure, making it easier to move workloads between different Kubernetes clusters or cloud providers. If the declarative configurations embed some details that are specific to some environments, then the portability becomes limited to these environments.
* **Automation and Tooling**: Declarative APIs are well-suited for automation because configuration is defined as code enabling the use of tools like Helm [10], Kustomize [12], or operators to manage complex applications and resources.
* **Human-Readable**: Declarative configurations are often more human-readable and maintainable compared to imperative scripts or instructions, which can be error-prone and hard to understand.

In summary, the declarative nature of the Kubernetes API simplifies resource management, enhances automation, increase reliability, and makes it easier to work with complex infrastructure and applications in a dynamic environment like Kubernetes.

NOTE: The present document only assesses Kubernetes technology; it can be possible to realize these benefits with other technologies as well.

### 4.3.3 Industry Adoption

As highlighted in the CNCF Cloud Native Definition v1.0 [5], the adoption of declarative APIs empowers organizations to develop and operate scalable applications in modern and dynamic environments, introducing GitOps as a new approach for managing the operation and the lifecycle of telecom networks. But transitioning to GitOps can be challenging due to potential cultural shifts and skillset gaps.

At its core, GitOps principles encompass the following [6]:

* **Declarative**: A system managed using GitOps expresses its desired state in a declarative manner.
* **Versioned and Immutable**: The desired state of a system is stored in a manner that enforces immutability, versioning, and maintains a complete version history.
* **Automatically Pulled**: Software agents automatically retrieve the desired state declarations from the source.
* **Continuously Reconciled**: Software agents consistently monitor the actual system state and strive to apply the desired state.

The Cloud-Native Telco Market Perception Report [4] stipulates wanting “to know which technologies and companies are perceived by the network operator community to be important to cloud-native telco strategies – so [their report] is all about perception, not which technologies and companies are actually involved with, or linked to, any cloud-native developments at the network operators represented by our survey respondents.”. It reports 89% of respondents have adopted a cloud-native strategy, among which 90% reports Kubernetes as being important to meeting strategic goals (based on 38 responses from 21 companies).

## 4.4 O2-IMS Services Analysis

### 4.4.1 Introduction

As of specification **v04.00**, the O2-IMS interface is composed of services at various stages of standardization. This section analyses each O2 IMS service, along with its applicability to a declarative interface.

### 4.4.2 Inventory Service

This service enables querying the O-Cloud resources and management services.

The inventory service defines objects which are exposed as read-only to the SMO. The only REST resource method supported is “GET”, except for the Subscription object which also supports “POST” and “DELETE”.

As part of the “GET” resource methods, a comprehensive list of query parameters is defined.

It is technically possible to implement an inventory system with the O2 IMS Inventory objects defined as Kubernetes Custom Resource Definitions (CRDs), but there are certain considerations and challenges that can make this approach less desirable:

* **Scalability and Performance**: CRDs can introduce performance overhead, especially when dealing with a large number of objects. Kubernetes itself is primarily designed for managing infrastructure and application workloads, and using it for inventory management might not be as efficient as purpose-built inventory systems.
* **Complexity**: Managing inventory typically involves complex data structures and relationships. CRDs might not provide the necessary data modeling and querying capabilities needed for complex inventory structures, which could lead to a more complex and less efficient inventory management system.
* **Lack of Inventory-Specific Features**: Kubernetes CRDs are not designed specifically for inventory management. They lack features such as built-in reporting, auditing, and advanced querying capabilities that are often essential for inventory systems.
* **Data Consistency**: Ensuring data consistency and integrity in an inventory system is critical. CRDs do not inherently enforce data consistency rules or provide transactional guarantees, which might be necessary for inventory systems with stringent data requirements.
* **Limited Control over Data Storage**: Kubernetes abstracts much of the underlying infrastructure, including data storage. In some cases, it can be needed to have more control over how inventory data is stored, indexed, and retrieved than what CRDs offer.
* **Migration and Portability**: It can be challenging to extract and convert data from Kubernetes CRDs when the inventory system needs to migrate to a different platform or infrastructure.

Kubernetes CRDs provide flexibility and extensibility for managing custom resources within a Kubernetes cluster, they might not be the best choice for implementing a complex inventory management system. It can be more practical to use purpose-built inventory management tools or databases that are designed to handle the complexities and challenges associated with inventory management.

### 4.4.3 Provisioning Service

This service support procedures for the automation of O-Cloud cluster lifecycle events, such as O-Cloud cluster creation, deletion or scaling. It is currently undefined, but actively discussed.

Provisioning clusters declaratively through CRDs can offer several advantages, especially in scenarios where a high degree of automation, customization, and control over cluster deployments is expected. Some of the advantages are:

* **Infrastructure as Code (IaC):** Declarative provisioning with CRDs enables defining the cluster lifecycle management operations as code. This means the cluster configuration can be versioned, reviewed, and managed using familiar development practices, making it easier to track changes and collaborate with others.
* **Automation**: CRDs enable automation of cluster provisioning. By defining the cluster's desired state in CRDs, tools like Kubernetes Operators or custom controllers can be used to automatically reconcile the actual state with the desired state, reducing manual intervention and potential human error.
* **Customization**: Kubernetes CRDs enable customization of the cluster's configuration to the specific needs. Custom resource types to represent different types of clusters (e.g., development, production, testing) and configure them with distinct settings, such as node sizes, networking options, and addons, can be defined.
* **Reusability**: CRD definitions can be reused and shared across different infrastructures or environments. This can lead to consistent cluster configurations and simplify the management of multiple clusters with similar requirements.
* **Versioning and Rollbacks**: CRD-based configurations are versioned, making it straightforward to roll back to previous cluster configurations if issues arise during an update. This provides a safety net for changes to the cluster infrastructure.
* **Ecosystem Integration**: Using CRDs aligns the cluster provisioning process with Kubernetes-native concepts, making it easier to integrate with other Kubernetes components and tools in the ecosystem. This includes monitoring, logging, and scaling solutions designed to work seamlessly with Kubernetes.
* **Scalability**: As the needs change, scaling clusters up or down is achieved by updating the CRD instance configurations. This flexibility ensures the clusters can adapt to evolving workloads and resource requirements.
* **Continuous Integration/Continuous Deployment (CI/CD)**: Declarative cluster provisioning fits seamlessly into CI/CD pipelines by automating cluster creation as part of the deployment process, ensuring clusters are always up to date with the latest configurations.

Declarative provisioning of clusters through CRDs offers a powerful way to automate, customize, and manage clusters efficiently. It aligns well with Kubernetes principles and best practices, making it a compelling choice for organizations seeking greater control and automation over their cluster infrastructure.

The concept of Cluster Template aims at capturing O-Cloud Node Cluster configurations for an O-Cloud. It is inline with the goal of the present document of adopting a declarative API approach for the provisioning service. And this is inline with the goal of having a declarative way to manage lifecycle events of O-Cloud Node Clusters.

NOTE: As of issuing the present document version, the cluster template concept is being still developed. Depending on the standardization of cluster templates (such as models, associated interfaces, etc.), the descriptions above can be revisited.

### 4.4.4 Lifecycle Management Service

This service supports procedures for the automation of O-Clouds lifecycle events, such as O-Cloud Genesis and O-Cloud Scaling. It is currently not defined.

As it stands, no analysis can be made for the first version of the present Technical Report.

### 4.4.5 Software Management Services

This service supports procedures that enable SMO/FOCOM to initiate the software update process for the O-Cloud Infrastructure Management software, the Deployment Management software, Server OS Software, updates, and patches, and firmware updates for accelerators. It is currently not defined.

As it stands, no analysis can be made for the first version of the present Technical Report.

### 4.4.6 Monitoring Service

This service supports procedures that enable a consumer i.e., SMO, to monitor the reachability of the IMS, as well as subscribe to alarms, faults and performance metrics that can be generated by Infrastructure Inventory objects. The information model and some service operations are defined.

Kubernetes CRDs offer a flexible way to extend the Kubernetes API and define custom resources, but they might not be the most suitable choice for implementing a monitoring service for metrics, faults, and alarms for the following reasons:

* **Performance Overhead**: Monitoring systems typically deal with high volumes of data and require low-latency access to metrics and alarms. Storing this data as CRDs within Kubernetes can introduce performance overhead due to the way Kubernetes handles resource storage and retrieval.
* **Data Volume and Retention**: Monitoring systems often generate and store a large amount of time-series data. Storing this data as CRDs could lead to scalability issues and challenges related to data retention and purging, which are better addressed by specialized data stores designed for time-series data.
* **Data Querying and Aggregation**: Monitoring systems require powerful querying and aggregation capabilities to extract meaningful insights from metrics and alarms. CRDs cannot provide the necessary querying features, and querying CRDs directly could be inefficient and cumbersome.
* **Data Transformation and Enrichment**: Monitoring often involves data transformation and enrichment processes to make sense of raw metrics and create meaningful dashboards and reports. Implementing these processes within CRDs can be complex and might not be the most efficient approach.
* **Scalability and High Availability**: Monitoring services scale horizontally to handle increasing data volumes and ensure high availability. Kubernetes CRDs might not offer the same level of scalability and high availability features as dedicated monitoring solutions.
* **Ecosystem Compatibility**: The broader monitoring ecosystem, including popular tools like Prometheus, Grafana, and Elasticsearch, is not natively designed to work with Kubernetes CRDs. Using CRDs needs of significant customization and integration work.
* **Operational Overhead**: Maintaining a monitoring system implemented with CRDs might introduce operational complexity, due to the management and maintenance both the monitoring system and the Kubernetes cluster.

As described, CRDs might not be the best choice for implementing a comprehensive monitoring service. Specialized monitoring solutions are better equipped to handle the complexities of monitoring, alerting, data storage, querying, and integration with external systems and data sources, while also providing the necessary performance and scalability required for monitoring large-scale and critical applications.

## 4.5 Proposal for O2-IMS K8s interface support

### 4.5.1 Introduction

Even though the benefits of a declarative interface are tangible, considering the analysis done in the previous section, they cannot be realized for the Monitoring and Inventory O2 IMS services: the O2 IMS Provisioning service is a good fit for having a declarative K8s interface. Regarding Software and Lifecycle Management, they will have to be re-assessed once fully defined or their specification progresses.

The recommendation is to have an O2 IMS Provisioning data model in the format of a CRD for the following use-cases defined in the O-RAN-WG6.ORCH-USE-CASES-R003-v08.00 specification:

* 3.11.2 Create Kubernetes (K8s) Cluster Use Case
* 3.11.3 Delete Kubernetes (K8s) Cluster Use Case
* 3.11.4 Update O-Cloud Node Cluster Use Case

To avoid having two different interfaces for O2 IMS, the same information model could be used to define the RESTful API(s) and the CRD(s).

NOTE: This recommendation introduces a fragmentation of the O2 IMS Interface for the O2 IMS Provisioning service.

### 4.5.2 URI compatibility

In Kubernetes, CRDs are accessed and managed through the Kubernetes API server, which listens on the following URI endpoint: **/apis/{group}/{version}/{resource}.** More details on Kubernetes Resource URIs can be found in its documentation [7].

An URI endpoint is comprised of the following parts:

* **{group}:** The API group to which the CRD belongs. Custom resources are often grouped under a specific API group to avoid naming conflicts with built-in resources. For example, the group of a CRD for a custom application can be myapp.example.com.
* **{version}:** The version of the CRD. As the CRD evolves, there can create multiple versions to handle changes in the resource definition. For example, version can be v1alpha1, v1beta1, or v1.
* **{resource}:** The name of the custom resource. This is typically pluralized. For example, if the custom resource is called MyApp, the resource name might be myapps.

The K8s URI endpoint structure is inline with the currently defined O2 IMS URI structure, defined in section 3.1.2 in v04.00: {apiRoot}/<apiName>/<apiMajorVersion>/<apiSpecificResourceUriPart>

### 4.5.3 Example of K8s API for Cluster Template

The following is an example CRD for the Cluster Template. It is very simple and only consists of two parameters: name and description.

|  |
| --- |
| apiVersion: apiextensions.k8s.io/v1  kind: CustomResourceDefinition  metadata:  # name match the spec fields below, and be in the form: <plural>.<group>  name: clustertemplates.O2ims-infrastructureprovisioning.oran.alliance.org  spec:  # group name to use for REST API: /apis/<group>/<version>  group: O2ims-infrastructureprovisioning.oran.alliance.org  # list of versions supported by this CustomResourceDefinition  versions:  - name: v1  # Each version can be enabled/disabled by Served flag.  served: true  storage: true  schema:  openAPIV3Schema:  description: 'ClusterTemplate is the schema for O2 IMS Provisioning'  type: object  properties:  spec:  description: 'Information representing an Cluster Template  type: object  properties:  name:  description: Human readable name of the Cluster Template  type: string  description:  description: Human readable description of the Cluster Template  type: string  # either Namespaced or Cluster  scope: Cluster  names:  # plural name to be used in the URL: /apis/<group>/<version>/<plural>  plural: clustertemplates  # singular name to be used as an alias on the CLI and for display  singular: clustertemplate  # kind is normally the CamelCased singular type. Your resource manifests use this.  kind: ClusterTemplate  # shortNames allow shorter string to match your resource on the CLI  shortNames:  - ct |

In order to use this ClusterTemplate CRD declaratively, one can use the following Custom Resource

|  |
| --- |
| apiVersion: O2ims-infrastructureprovisioning.oran.alliance.org/v1  kind: ClusterTemplate  metadata:  name: example  spec:  name: "O-Cloud Node Cluster Template"  description: "Cluster Template for XYZ" |

In order to use this ClusterTemplate CRD through the Kubernetes API Server, one can use the following RESTful URI

|  |
| --- |
| /apis/O2ims-infrastructureprovisioning.oran.alliance.org/v1/clustertemplates |

Finally, Kubernetes enable the extraction of OpenAPIv3 specification for the CRD using the following RESTful URI

|  |
| --- |
| /openapi/v3/apis/O2ims-infrastructureprovisioning.oran.alliance.org/v1 |

### 4.5.4 Proposal

In order to support both a RESTful interface and Kubernetes API for the O2 IMS Provisioning service, the proposal is:

* Ensure the {apiName} comply to a lowercase RFC 1123 subdomain, section 2.1 "Host Names and Numbers" which limits host names to letters-digits-hyphen.

Example: O2ims-infrastructureprovisioning.oran-alliance.org

* Have one common information model.
* Have two data models; one being a JSON document, and one being in the format of CRD(s).

By adopting this proposal, the O-Cloud O2 IMS interface implementation can support both the RESTful API and the Kubernetes CRD(s) as it enables the utilization of the CRD(s) through the RESTful interface created by Kubernetes, as illustrated in section 2.5.1.

Communication between FOCOM and IMS via the O2 Interface, whether employing a RESTful API like the one specified for the O2 IMS Inventory Service or using CRD(s) as proposed in this section for the O2 IMS Provisioning Service, necessitates a single set of credentials for the consumer, i.e., FOCOM. This ensures a consistent authentication process for executing requests.

As part of the data modeling in the format of CRD(s), it is expected that validation rules are also specified to ensure no unwanted functionality gets into the O-Cloud. Kubernetes has a well-define mechanism to perform this validation [8].

NOTE: Depending on the level of completeness of the validation rules of CRD(s) within the 02 IMS Provisioning data model, the O-Cloud vendor could add additional validation rules for enhanced security.

## 4.6 Impact on WG6 Specifications and Use Cases

Further specification of the O2 IMS interface [9] might leverage the recommendations described in the present document.

# History

|  |  |  |
| --- | --- | --- |
| Date | Revision | Description |
| 2023.11.16 | 00.00.03 | Published as Final version 01.00 |