Technology
Arts Sciences
TH Köln

Evaluation of 5G Open Radio Access Network Simulation Environments

Md Nur Mohammad

Master Research Project
MSc. Communication Systems and Networks

Supervisor: Prof. Dr. Andreas Grebe

August 9, 2023

Abstract

The main focus of this research project is to deploy the O-RAN simulator and connect with the 5G core network in the future. The O-RAN Alliance software community still needs to develop its complete architecture successfully. According to the latest (G-Release) release, the O-RAN Alliance software community successfully developed the RAN intelligent controller (RIC) components. Now the focus is on assessing the deployment environments and deploying O-RAN RIC components by following the O-RAN architecture in a containerized environment orchestrated by Kubernetes. According to the release, during this project deployed non-real-time and near-real-time RIC, connected both, and analyzed the data through Wireshark. The findings contribute to understanding RAN intelligent components function for future open RAN deployment for 5G networks. All implementation is performed on TH Köln's (DN.LAB) on-premise servers to be available for future research or performance.

Keywords: open RAN, O-RAN software, RAN intelligent controller (RIC), containerized environment, Kubernetes, 5G networks.

Contents

List of Figures

1 Introduction

The rise of 5G networks has become crucial to meeting the growing demand for fast data transmission and improved network capacity. The Open Radio Access Network (O-RAN) architecture offers a solution to enhance the flexibility and interoperability of 5G networks. To ensure the successful deployment and optimization of O-RAN, it is essential to evaluate and test its components and functionalities. Simulating O-RAN in controlled environments provides a cost-effective way to assess its performance.

This research project aims to evaluate simulation environments for deploying O-RAN. By analyzing simulation tools and methodologies, this study aims to identify their components and suitability for simulating O-RAN. Additionally, this research will test the connectivity of components and create a service using the Open Network Automation Platform (ONAP). This platform was introduced by the O-RAN Alliance software community.

O-RAN Alliance has developed O-RAN components and published their releases, including this research paper based on G-Release. G-Release is developed on Docker and Kubernetes, and this research followed Kubernetes implementation.

Apart from that, this Kubernetes implementation needs additional support from Docker and Helm Chart.

Kubernetes: Kubernetes, an open-source container orchestration platform, revolutionizes how applications are deployed, scaled, and managed. Initially created by Google and overseen by the Cloud Native Computing Foundation, Kubernetes provides developers with a powerful and intuitive solution to handle applications across server clusters or virtual machines effectively. By abstracting the complexities of the underlying infrastructure, it offers features such as load balancing, automatic scaling, and storage orchestration. With its widespread adoption, Kubernetes has emerged as the de facto standard for container orchestration, empowering the development of resilient and scalable cloud-native solutions.

Docker for Kubernetes: Docker is a platform that helps automate the deployment of applications using containers. In Kubernetes, Docker is used as the container runtime. It allows applications to be packaged as Docker images, which are portable and self-contained. Kubernetes utilizes Docker to manage and run these containers on a cluster, making it easier to scale and deploy applications efficiently. Together, Docker and Kubernetes provide developers with the ability to build, package, and manage applications in a containerized environment.

Helm Chart: Helm emerges as a convenient package manager designed specifically for Kubernetes, streamlining the process of deploying and managing applications. With Helm, applications can be packaged into Helm Charts, which consist of multiple files that configure and encapsulate Kubernetes resources.

This packaging approach enables effortless installation and consistent management of applications, fostering repeatability and ease of use. By providing a standardized mechanism for defining, sharing, and deploying applications, Helm significantly simplifies the management of complex applications while promoting seamless collaboration among teams.

Chapter 2 will detail information about the O-RAN Alliance and the ecosystem. Apart from that, the O-RAN architecture *[Figure 1]* will be presented, along with complete information on every component of O-RAN.

In Chapter 3, possible solutions with an environment for the deployment of virtual infrastructure able to deploy O-RAN RIC components are discussed—moreover, a detailed description of prerequisite installation steps with necessary commands and example output.

Next, Chapter 4 will describe the steps to connect RIC components and create a service between them. In addition, in the result section, there is a Wireshark capture and analysis of packets.

2 O-RAN Alliance and O-RAN Ecosystem

2.1 O-RAN Alliance Overview

The O-RAN Alliance was established with the primary objective of advancing openness and intelligence within the Radio Access Network (RAN) industry. Since its inception, the alliance has garnered significant momentum, attracting over 200 members, including major mobile operators, network equipment vendors, and system integrators.

A key focus of the O-RAN Alliance is the development of open interface specifications that facilitate multi-vendor interoperability and enable the deployment of virtualized RAN solutions. These open interfaces empower network operators to select and integrate components from different vendors, fostering competition and avoiding vendor lock-in. To ensure compliance with the alliance's specifications and promote interoperability, the organization offers testing and certification services. This ensures that RAN components from various vendors can seamlessly work together, reducing the risk of vendor lock-in and providing network operators with the flexibility to choose the most suitable components for their specific requirements.

In addition, the O-RAN Alliance has created a range of use cases that exemplify the benefits of an open RAN architecture. These use cases span diverse scenarios and exemplify how an open and interoperable RAN can enhance coverage, offer flexible deployment options, and deliver improved performance.

The O-RAN Alliance seeks to create a more open and intelligent RAN by:

1. ***Open RAN architecture:*** The O-RAN Alliance promotes an open RAN architecture that enables network operators to select best-of-breed components from different vendors. This approach allows network operators

to avoid vendor lock-in and promotes competition in the industry.

2. ***O-RAN specifications:*** The Alliance develops and maintains a set of specifications that define the interfaces between different RAN components. These specifications are designed to be open and interoperable, allowing network operators to mix and match components from different vendors.

3. ***O-RAN testing and certification:*** The Alliance provides testing and certification services to ensure that RAN components from different vendors are interoperable and compliant with the Alliance's specifications. This helps reduce vendor lock-in risk and enables network operators to choose the best components for their specific needs.

4. ***O-RAN use cases:*** The Alliance has developed several use cases that demonstrate the benefits of an open RAN architecture. These use cases cover a range of scenarios, from rural coverage to indoor deployments, and showcase the advantages of open and interoperable RAN components.

5. ***O-RAN market momentum:*** The O-RAN Alliance has gained significant momentum recently, with many major mobile network operators and equipment vendors joining the organization. The Alliance's focus on open and interoperable RAN components has struck a chord with many industry stakeholders seeking more choice and flexibility.

Overall, the O-RAN Alliance is crucial in driving innovation and competition in the RAN industry. Its focus on openness and interoperability is helping to create a more diverse and vibrant market. At the same time, its testing and certification services ensure that RAN components from different vendors work together seamlessly.

2.2 Release Specifications

The O-RAN Alliance has released several specifications and technical reports since its inception. These releases are designed to promote interoperability and support the development of open and intelligent RAN solutions.

Here are some of the key releases from the o-ran Alliance:

1. ***O-RAN Architecture:*** In this release, we aim to give you an in-depth understanding of the O-RAN architecture. We outline the key components and interfaces that constitute this system. The architecture is purposefully designed to offer flexibility and scalability, enabling it to adapt to various deployment scenarios. Whether it's a small-scale or large-scale implementation, the O-RAN architecture is built to accommodate diverse needs and ensure seamless scalability.

2. ***O-RAN Use Cases:*** The O-RAN Alliance has developed several use cases demonstrating the benefits of an open and interoperable RAN. These use cases cover a range of scenarios, from small indoor cells to rural deployments, and highlight the flexibility and cost savings that can be achieved with an open RAN architecture.

3. ***O-RAN Radio Intelligent Controller (RIC) Functional Description:*** At the heart of the O-RAN architecture, the O-RAN RIC (Radio Intelligent Controller) is a central intelligence layer. It is vital for managing and orchestrating RAN (Radio access network) resources. This specification offers an extensive overview of the RIC's functionalities and interfaces, providing a comprehensive understanding of its capabilities and how it interacts within the O-RAN ecosystem.
4. ***O-RAN Testing and Integration:*** The O-RAN Alliance provides testing and integration services to ensure that RAN components from different vendors are interoperable and compliant with the Alliance's specifications. This release provides an overview of the testing and certification process and information on the testing tools and environments used.
5. ***O-RAN Software Community:*** The O-RAN Alliance has established a software community to promote the development of open-source software for the RAN industry. The community is focused on developing software that supports the Alliance's specifications and promotes interoperability between different vendors.

These updates demonstrate the O-RAN Alliance's strong commitment to openness and interoperability in the RAN industry. The Alliance provides precise specifications and guidelines to create a more diverse and competitive market. This, in turn, benefits both network operators and end-users by offering more choices and improving performance in the industry.

2.3 O-RAN Software Community

The O-RAN Alliance has started a cooperative project called the O-RAN Software Community to promote the creation of open-source software for the RAN sector. The community is designed to bring together developers, vendors, and network operators to work on software projects that support the O-RAN Alliance's specifications and promote interoperability between different RAN components.

The community's primary focus is on developing software for the O-RAN Intelligent Controller (RIC), which is a key component of the O-RAN architecture. The RIC provides a centralized intelligence layer that can be used to manage and orchestrate RAN resources, and the community is working on developing software that supports the RIC's functionality and interfaces.

In addition to the RIC, the community is also working on other software projects that support the O-RAN Alliance's specifications. These include projects related to network management, performance monitoring, testing, and integration. One of the key benefits of the O-RAN Software Community is that it provides a collaborative environment for developers to work on software projects that are aligned with the O-RAN Alliance's goals. By working together, developers can share knowledge and resources and ensure that their software is compatible with other components of the RAN ecosystem.

Overall, the O-RAN Software Community is a key initiative that is helping to drive innovation and openness in the RAN industry. By promoting the development of open-source software, the community is helping to create a more diverse and competitive market, which ultimately benefits network operators and end-users alike. Developing software that supports the RIC's functionality and interfaces.

2.4 Architectural Release

Accordingly, the O-RAN architecture *[Figure 1]* contains several components that are developed by O-RAN Alliance software community.

There are eight releases with release dates:

Release Number	Release Name	Release date
1	A Release (Amber)	November 2019
2	B Release (Bronze)	Jun 2020
3	C Release (Cherry)	December 2020
4	D Release (Dawn)	July 2021
5	E Release (Emerald)	December 2021
6	F Release	July 2022
7	G Release	December 2022
8	H Release	Upcoming

Table 1: O-RAN Alliance Releases

2.4.1 Release Notes

This research focused on G-Release, which has specific release notes. The release notes are below:

- RAN Intelligent Controller Applications (RICAPP) features.
- Near Real-time RAN Intelligent Controller (RIC) features.
- Non-Real time RAN Intelligent Controller (NONRTRIC) features.
- Operations and Maintenance (OAM) features.
- O-RAN Central Unit (OCU) features.
- O-RAN Distributed Unit High Layers (ODU-HIGH) features.
- Infrastructure (INF) features.
- Integration and Testing (INT) features.
- Service Management and Orchestration (SMO) features.

2.5 Architecture of O-RAN

The Open RAN logical architecture provides versatility, adapting to diverse network needs. With standardized interfaces, RAN components from different vendors can seamlessly communicate and work together. The division into radio and non-radio domains enables effective management of both physical and control aspects of the RAN system. This architecture empowers network operators to have greater flexibility in deploying and optimizing their RAN infrastructure. By promoting interoperability and modularity, the Open RAN architecture drives innovation and competition in the RAN industry.

The architecture *[Figure 1]* is thoughtfully divided into two main domains to facilitate efficient management and control. The radio domain encompasses the base station and associated equipment responsible for wireless signal transmission and reception. The non-radio domain focuses on management and orchestration functions, providing the necessary intelligence to oversee and optimize RAN resources effectively.

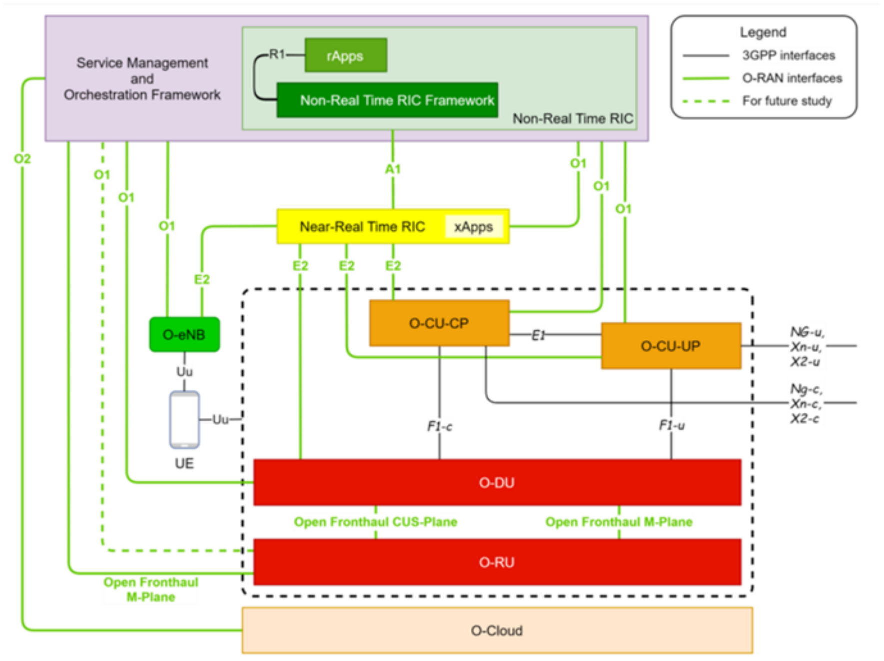


Figure 1: Logical Architecture of Open RAN.
(Source*)

2.5.1 Service Management and Orchestration (SMO)

Service Management and Orchestration (SMO) is a functional area in the O-RAN architecture *[figure 1]* that is responsible for managing and orchestrating services and applications across the RAN network. The SMO layer is designed to work in conjunction with other functional areas, such as the Management and Orchestration (MANO) layer and the Network Slicing Function (NSF) layer, to ensure that services and applications are deployed and managed effectively.

The Service Management and Orchestration (SMO) layer within the O-RAN architecture plays a crucial role in enabling the non-RT RIC to access specific functionalities related to RAN optimization actions, such as collecting Performance Measurements (PM) through O1 and O2 interfaces. However, SMO also has a much broader mandate, including the orchestration of the Network Functions Virtualization Infrastructure (NFVI) and managing the lifecycle of O-RAN network elements, which can be either Virtual Network Functions (VNFs) hosted in specific locations of the O-Cloud infrastructure or Physical Network Functions (PNFs) exposed by cell sites.

For non-virtualized parts, such as O-RU functionalities that are related to area coverage and must be placed at cell sites, the SMO supports the deployment of physical network elements on dedicated physical resources with management through the O1 interface. However, for virtualized network elements, the SMO has the capability to interact with the O-Cloud to perform network element lifecycle management. For example, it can instantiate the virtualized network element on the target infrastructure through the O2 interface or indicate the selected geo-location for each VNF to be instantiated.

To ensure smooth communication between the deployed network elements, the SMO is also responsible for IP addressing, network reconfiguration, and system updates. To support a range of deployment solutions, the Operation and Maintenance architecture defined by O-RAN describes in detail the requirements necessary for the SMO framework to be provided by third-party Network Management Systems (NMS) or orchestration platforms, such as the Linux Foundation's Open Network Automation Platform (ONAP).

Overall, the Service Management and Orchestration framework is a critical component of the O-RAN architecture, providing a standardized interface and protocols for managing and orchestrating services and applications across the RAN network. With its ability to interact with both virtualized and physical network elements, the SMO enables greater interoperability, flexibility, and efficiency, ultimately benefiting both network operators and end-users.

2.5.2 near-real time RAN Intelligent Controller (near-RT RIC)

In the O-RAN architecture *[figure 1]*, the near-RT RIC, or near-real-time RAN Intelligent Controller, serves as a crucial function for enabling real-time control and optimization of O-RAN resources via fine-grained data collection and actions through the E2 interface.

Figure 2 shows the logical architecture and interfaces of the near-RT RIC. The near-RT RIC is connected to the non-RT RIC through the A1 interface. A1 interface work for policy-based guidance.

Now the E2 is a logical interface that connects the near-RT RIC with an E2 node. The O-CU-CP, O-CU-UP, O-DU, and O-eNB are connected with the near-RT RIC. Only one E2 node will be connected with near-RT RIC, and

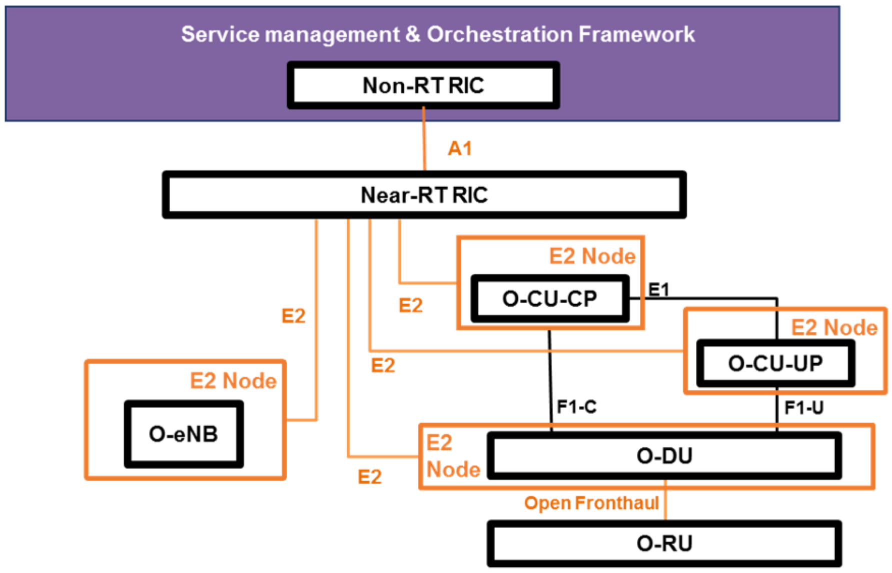


Figure 2: O-RAN architecture overview showing Near-RT RIC interfaces (Source*) Download WG3

multiple E2 nodes will be connected with E2 nodes, i.e., multiple O-CU, O-DU, and O-eNBs. The F1 and E1 are logical 3GPP interfaces.

Additionally, the Near-RT RIC serves as a host for multiple xApps that collect real-time information and provide extra services using the E2 interface. It can receive policies and obtain data enrichment information through the A1 interface. The E2 interface protocols are based on control plane protocols. In the event of an E2 or Near-RT RIC failure, the E2 Node can still offer services, but certain value-added services exclusive to the Near-RT RIC may experience an outage.

- ***Near-RT RIC Requirements:*** The Near-RT RIC architecture is expected to meet the following set of requirements:

  – Each E2 node configured to directly supply RIC services to the Near-RT RIC must be uniquely identified via the dedicated E2 connection that the Near-RT RIC uses.

  – A fact Several E2 nodes that each support a different RAT type may be able to establish E2 connections with near-RT RIC.

  – The Near-RT RIC is in charge of requesting from the E2 Nodes a list of the functions that provide RIC services and their related E2 service models.

  – The Near-RT RIC hosts a set of applications known as xApps. Each xApp can target specific RAN functions within a specific E2 node.

– The Near-RT RIC, like other network elements, should provide an O1 interface to the Service Management Orchestration layer. This interface makes element administration and configuration easier.

– The Near-RT RIC should provide an A1 interface to the Non-RT RIC, allowing for the exchange of policies that can affect the behavior of the Near-RT RIC and its hosted xApps, influencing the behavior of E2 Nodes.

– In the event of an E2 interface or Near-RT RIC failure, the E2 nodes ought to be able to function without the Near-RT RIC.

– The 10 ms to 1 s latency requirements for near-real-time optimization should be met by the Near-RT RIC.

- ***Near-RT RIC functions:*** Following functions are supported by near-RT RIC:

  – **Termination of the A1 interface:** First, terminate the interface from a non-RT RIC. After that, forward the A1 messages.

  – **Termination of the O1 interface:** In order to send management messages to the Near-RT RIC management function, the O1 interface from the Service Management Orchestration layer terminates at the Near-RT RIC.

  – **Termination of E2 interface:** The Near-RT RIC terminates the E2 interface from an E2 Node, directing xApp-related messages to the intended xApp and non xApp-related messages to the E2 Manager.

  – **xApps host:** The Near-RT RIC enables the execution of RRM control functionalities within its domain and enforces them in the E2 Nodes through the E2 interface. It also initiates xApp-related transactions over the E2 interface and handles the corresponding responses received from the E2 interface.

2.5.3 non-real time RAN Intelligent Controller (non-RT-RIC))

The non-RT RIC, or non-real-time RAN Intelligent Controller, is responsible for non-real-time control and optimization of RAN resources, including AI/ML workflows such as model training and updates and policy-based guidance of applications/features in the near-RT RIC. *[Figure 3]* demonstrating the interface and services of non-real time RIC.

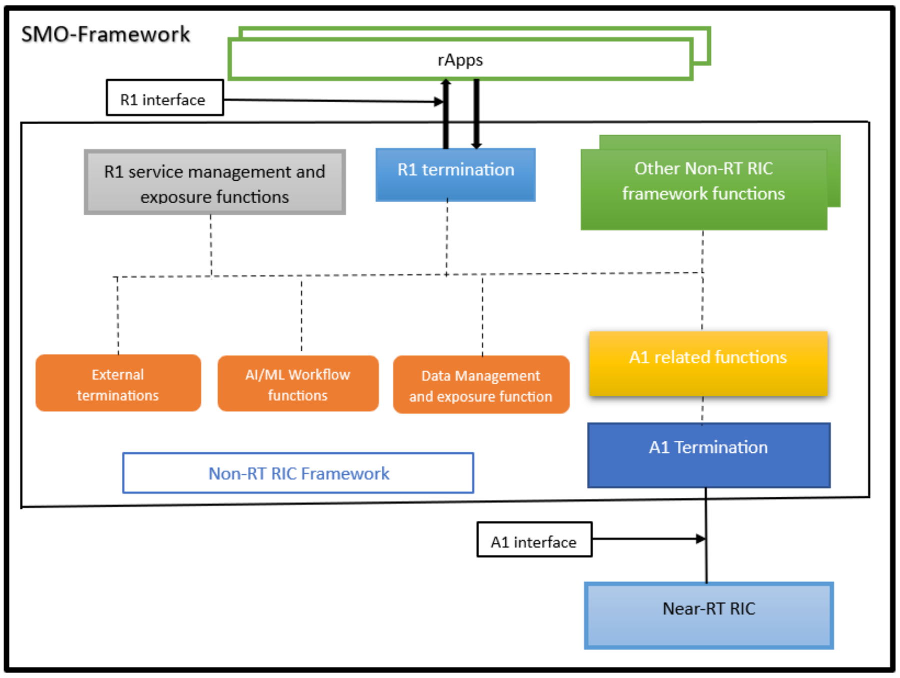


Figure 3: Non-RT RIC reference Architecture
(Source*) Download WG2

R1 Services:

R1 services refer to a collection of services provided by logical functions within the Non-RT RIC framework or SMO (Service Management and Orchestration) framework, as well as by rApps (RIC applications).

These services are designed to enable various functions such as service registration, service discovery, service notification, authorization, authentication, communication support, and potentially bootstrap and heartbeat services.

The R1 services are essential for managing and exposing functionalities within the Non-RT RIC framework, or rApps.

R1 Service Management and Exposure:

R1 service management and exposure functions are responsible for facilitating the usage and access of R1 services.

These functions handle tasks such as registering R1 services, discovering available services, notifying relevant parties about service updates, and providing authorization and authentication mechanisms for secure access to the services.

Additionally, communication support is provided to enable the exchange of messages between the Non-RT RIC framework, rApps, and the R1 services.

R1 Termination and R1 Interface:

The R1 termination represents the endpoint or interface through which the Non-RT RIC framework and rApps communicate with the R1 services.

The R1 interface acts as the means for exchanging messages and accessing the R1 services.

It enables the Non-RT RIC framework and rApps to interact with the R1 services, leveraging the functionalities provided by the R1 service management and exposure functions.

Service management and exposure services:

Service management and exposure services within the Non-RT RIC (Non-Real-Time RAN Intelligent Controller) framework or SMO (Service Management and Orchestration) framework encompass essential functionalities for handling services. These services include registration, discovery, notification, and secure access.

Service Registration: It involves making services available within the framework, or SMO environment. The framework or related apps register the services they offer, making it possible for other entities to find and use them.

Service Discovery: This feature enables entities within the framework or SMO environment to find and identify available services. By querying a registry or directory, entities can locate the specific services they require for their operations or interactions.

Service Notification: When changes or updates occur to registered services, relevant parties are informed through notifications. This ensures that entities stay informed about modifications, such as updates or terminations, to the services they rely on.

Service management and exposure services play a critical role in the Non-RT RIC framework or SMO framework, ensuring efficient operation, coordination, and secure access to services. By providing registration, discovery, notification, authorization, and communication support, these services enable seamless utilization of services within the framework or SMO environment.

A1 policy related services:

A1 policy-related services are services that pertain to the management, enforcement, and administration of policies within a system or framework. These services are responsible for defining, configuring, and enforcing policies to guide the behavior, actions, and access rights of entities within the system. Here are key aspects of A1 policy-related services:

Policy Definition and Configuration: These services allow the definition and configuration of policies that govern various aspects of the system, such as security, access control, resource allocation, or behavior guidelines.

Policies are typically defined using policy languages or rule-based systems, specifying conditions and actions to be enforced.

Policy Evaluation and Enforcement: A1 policy-related services evaluate and enforce policies in real-time or as needed. They assess the current state of the system, including the actions of entities and the context, and compare it against the defined policies.

If a policy violation is detected, appropriate actions are taken to enforce compliance or trigger mitigation procedures.

Policy Monitoring and Reporting:

These services provide monitoring capabilities to track policy compliance and system behavior. They generate reports and alerts regarding policy violations, exceptions, or patterns of non-compliance, enabling system administrators to take appropriate actions.

Policy Adaptation and Learning:

A1 policy-related services may incorporate adaptive or learning capabilities to dynamically adjust policies based on evolving system conditions or changing requirements.

They can analyze system behavior, gather feedback, and modify policies accordingly to optimize system performance or adapt to new circumstances.

Policy Interoperability and Integration:

A1 policy-related services facilitate the integration and interoperability of policies across different components or systems within an ecosystem. They enable policy exchange, translation, or synchronization mechanisms to ensure consistent policy enforcement and coordination across the system.

A1 policy-related services play a crucial role in managing and enforcing policies within a system, ensuring compliance, security, and appropriate behavior. They provide the necessary mechanisms to define, evaluate, adapt, and monitor policies, promoting system stability, consistency, and governance.

2.5.4 O-RAN Central Unit (O-CU)

The O-RAN Central Unit, or O-CU, is a logical node that houses the RRC, SDAP, and PDCP protocols. Specifically, the O-CU-CP hosts the RRC and control plane parts of the PDCP protocol, while the O-CU-UP hosts the user plane parts of the PDCP protocol and the SDAP protocol.

2.5.5 O-RAN Distributed Unit (O-DU)

The O-RAN Distributed Unit, or O-DU, is a logical node that houses the RLC, MAC, and High-PHY layers based on a lower layer functional split.

2.5.6 O-RAN Radio Unit (O-RU)

The O-RAN Radio Unit, or O-RU, is a logical node that houses the low-PHY layer and RF processing based on a lower-layer functional split. This is like 3GPP's "TRP" or "RRH," but is more specific in including the Low-PHY layer, such as FFT/IFFT and PRACH extraction.

2.5.7 O1 Interface

O1 is an interface that facilitates communication between management entities in the Service Management and Orchestration Framework and O-RAN managed elements. Its primary purpose is to enable the operation and management of these elements by supporting functions such as FCAPS management, software management, and file management. The O1 interface provides a standardized and secure means of accessing and controlling O-RAN network elements, ensuring efficient and effective management of these critical components.

2.6 RAN Intelligent Controller

RAN Intelligent Controller (RIC) *[Figure 4]* is a logical function that provides near-real-time control and optimization of Radio Access Network (RAN) resources through fine-grained data collection and actions over the E2 interface. RIC architecture is based on the principles of Open RAN and is designed to be flexible, scalable, and vendor neutral.

The RIC comprises two main elements: the near-RT RIC and the non-RT RIC. The near-RT RIC provides near-real-time control and optimization of RAN elements and resources, such as radio resource allocation, traffic steering, and interference management. On the other hand, the non-RT RIC enables non-real-time control and optimization of RAN elements and resources, including AI/ML workflows for model training and updates and policy-based guidance of applications/features in near-RT RIC.

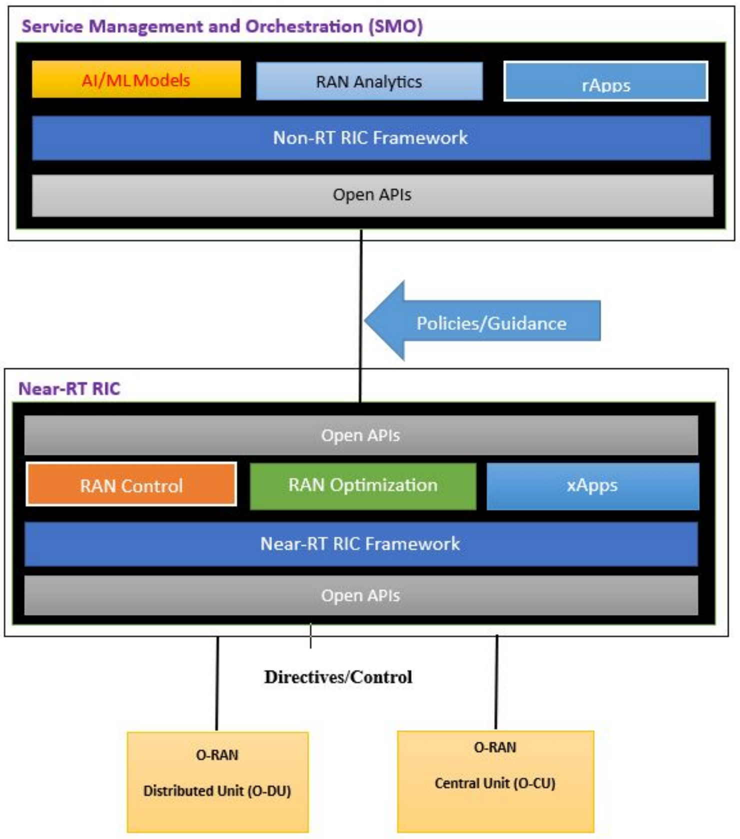


Figure 4: RAN Intelligent Controller Architecture
(Source*) Download WG1

The RIC architecture also includes other key elements such as O-CU, O-CU-CP, O-CU-UP, O-DU, and O-RU. O-CU is the central unit in the RAN that hosts RRC, SDAP, and PDCP protocols. O-CU-CP is the control plane part of the PDCP protocol, while O-CU-UP hosts the user plane part of the PDCP protocol and the SDAP protocol. O-DU hosts RLC, MAC, and high-PHY layers based on a lower-layer functional split, while O-RU hosts low-PHY layers and RF processing based on a lower-layer functional split.

RIC architecture also includes an O1 interface between management entities in the Service Management and Orchestration Framework and O-RAN managed elements. The O1 interface is used for operation and management, through which FCAPS management, software management, file management, and other operations are achieved.

3 RIC Environment Set-up and Deployment

3.1 Infrastructure Preparation and Deployment of near-RT RIC.

Hardware Requirements:

Operating System	Ubuntu 20.04
Core	8 vCPU
Memory	16 GB
Disk	100 GB
GPU	16 GB

Table 2: Hardware Requirements for near-RT RIC

Software Requirements:

- ***Container orchestration:*** Kubernetes- v1.16
- ***Container runtime:*** Docker and docker-compose (latest)
- ***Kubernetes Package manager*** Helm Chart v3.5
- ***Helm Chart Repository:*** ChartMuseum

3.1.1 Prerequisites Installation and deployment of near-RT RIC.

1. Start by preparing a fresh Ubuntu 20.04 environment specifically tailored for near-RT RIC deployment.
2. Install Kubernetes, helm, and the necessary base chart, ensuring they are properly configured for the upcoming Near-RT RIC deployment.
3. Proceed with the installation of near-RT RIC, taking care to follow the recommended guidelines and dependencies.
4. Compile and establish a seamless connection to the O-RAN E2 (e2-node) simulator sourced from the O-RAN SC simulator project.
5. Utilize the dms-cli tool to effortlessly deploy xApps, streamlining the deployment process and enhancing efficiency.
6. Compile, onboard, and install the hw-go xapp derived from the O-RAN SC xApp project, incorporating its functionality into the system with precision.

Step (1): Prepared ubuntu 20.04 from DN.Lab with hardware specifications.

According to the prerequisites, set up the Ubuntu server from DN.LAB (TH Köln). Now we can check the Ubuntu version *[Figure 5]* against the requirements by following the command.

```
    lsb_release -a
```

To ensure that, Ubuntu version is right:

```
    root@srv6:/home/o-ran# lsb_release -a
    No LSB modules are available.
    Distributor ID: Ubuntu
    Description:    Ubuntu 20.04.6 LTS
    Release:        20.04
    Codename:       focal
```

Step (2): Install Kubernetes, helm, and the necessary base chart, ensuring they are properly configured for the upcoming Near-RT RIC deployment.

(Note: Run all command in root user.)

Sample command: *sudo su*

Now clone the git repository that has deployment scripts and additional files (ric-plt/dep) with the following command.

```
    git clone https://gerrit.o-ran-sc.org/r/ric-plt/ric-dep
```

To deploy near-RT RIC we need to install Docker, Kubernetes, Helm, and Kubernetes-CNI.

To install Kubernetes, run the following command in the right directory:

```
    cd ric-dep/bin
    ./install\_k8s\_and\_helm.sh
```

Note: It will take time, and once it's done, check if Kubernetes is installed or not with the following command. It will show whether Kubernetes system pods are running or not.

```
    Kubectl get pods -n kube-system
```

Output:

```
root@srv:/home/o-ran# kubectl get pods -n kube-system
NAME                                 READY   STATUS    RESTARTS  AGE
coredns-5644d7b6d9-58xml             1/1     Running   2         12d
```

```
coredns-5644d7b6d9-d446p           1/1    Running  2          12d
etcd-srv6.5g.dn.th.koeln.de        1/1    Running  2          12d
kube-apiserver-srv6.5g.dn.th...    1/1    Running  2          12d
kube-flannel-ds-rr56g              1/1    Running  2          12d
kube-proxy-dgxvm                   1/1    Running  2          12d
kube-scheduler-srv6.5g.dn.th....   1/1    Running  2          12d
```

This is the output of the Kubectl get pods -n kube-system command. Here, all necessary pods are running, which is mandatory for Kubernetes installation. Now Kubernetes is installed properly.

Note: Make sure curl is installed. If not, then run this command to install curl:

```
    apt install curl
```

To set up the new systemconfig folder, we need to create it and then include the *PROXY variables. Additionally, make sure to add the line ”EnvironmentFile=-/etc/sysconfig/docker” and include the registry-mirrors in the ExecStart.

Include the line ”EnvironmentFile=-/etc/sysconfig/docker” in the ”systemconfig” folder. This line configures the Docker environment.

Add the registry-mirrors entry in the ExecStart section. This enables the system to use mirror servers for faster Docker image downloads.

Ensure that the systemconfig folder contains all the necessary configurations, including the *PROXY variables, the ”EnvironmentFile=-/etc/sysconfig /docker” line, and registry-mirrors.

Run the commands below

```
    mkdir /etc/sysconfig
    vi /etc/sysconfig/docker
    vi /lib/systemd/system/docker.service
    systemctl daemon-reload
    service docker restart
```

Now we will install Chart Museum into Helm and add ric-common templates with the following command.

```
    ./install\_common\_templates\_to\_helm.sh
```

Step (3):Proceed with the installation of near-RT RIC, taking care to follow the recommended guidelines and dependencies.

Now Create a txt file to separate the image pull from the public image registry from the actual installation. make sure to use same versions as in

../RECIPE_EXAMPLE/example_recipe_oran_f_release.yaml

Run this command:

```
vim versions.txt
```

Note: 'vim' is an editor. Make sure vim is installed.

Now create a Docker container text file.From the developer images list (below), copy them and save them as *versions.txt*.

```
versions.txt
nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-a1:3.0.0
nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-appmgr:0.5.7
nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-dbaas:0.6.2
nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-e2mgr:6.0.1
nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-e2:6.0.1
nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-rtmgr:0.9.4
nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-submgr:0.9.5
nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-vespamgr:0.7.5
nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-o1:0.6.1
nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-alarmmanager
:0.5.14
nexus3.o-ran-sc.org:10002/o-ran-sc/it-dep-init:0.0.1
docker.io/prom/prometheus:v2.18.1
docker.io/kong/kubernetes-ingress-controller:0.7.0
docker.io/kong:1.4
docker.io/prom/alertmanager:v0.20.0
```

So, we saved all Docker images in versions.txt file . Now Pull the docker images from nexus3.o-rano-sc.org

Run the command below.

```
for i in `cat versions.txt`; do echo $i; docker pull $i;
done
```

Note: It will take time. We pulled this image before creating Kubernetes pods because Kubernetes will create as following RECIPE_EXAMPLE file. We pulled the actual Docker container file for near-real-time ric deployment.

After the recipes are edited and helm started, the Near Realtime RIC platform is ready to be deployed, but first update the deployment recipe as per instructions in the next section.

Now modify and deploy near-RT RIC. Now we can start the installation of near-RT RIC. Before that, we need the IP address of the particular node, and to get the IP address, run the following command.

```
ip a
```

Edit the recipe files ./RECIPE_EXAMPLE/example_recipe_latest_stable.yaml (which is a softlink that points to the latest release version). "example_recipe_latest_unstable.yaml points to the latest example file that is under current development.

```
    vim ../RECIPE\_EXAMPLE/example\_recipe\_oran\_f\_release.
    yaml
```

Deployment scripts support both helm v2 and v3. The deployment script will determine the helm version installed in cluster during the deployment.

After updating the recipe, we can deploy the RIC with the command below. Note that we generally use the latest recipe marked stable or one from a specific release.

Run the command below:

```
    cd ric-dep/bin
    ./install -f ../RECIPE\_EXAMPLE/example\_recipe\_oran\_f\
    _release.yaml
```

Note: It will take time. After installation succeeds, we can see that near the RT RIC platform, pods are actually running.

Now check the deployment status with the following command.

```
    Kubectl get pods -n ricplt
```

The output of *Kubectl get pods -n ricplt* command is below:

```
root@srv6:/home/o-ran# kubectl get po -n ricplt
NAME                                      READY      STATUS
deployment-ricplt-a1mediator-669cc7       1/1        Running
    4647-t22qp
deployment-ricplt-alarmmanager-577        1/1        Running
    85458dd-p95s4
deployment-ricplt-appmgr-77986c9c         1/1        Running
    bb-l6b21
deployment-ricplt-e2mgr-5dd878f58         1/1        Running
    b-npdff
deployment-ricplt-e2term-alpha-68         1/1        Running
    98f8696d-5smjj
deployment-ricplt-01mediator-5ddd6        1/1        Running
    6b4d6-5xlv9
deployment-ricplt-rtmgr-788975975         1/1        Running
    b-ghlgs
deployment-ricplt-submgr-68fc6564         1/1        Running
    88-6wnd6
deployment-ricplt-vespamgr-84f7d8         1/1        Running
    7dfb-rq7zx
r4-infrastructure-kong-7995f4679b         2/2        Running
    -bs95n
r4-infrastructure-prometheus-aler         2/2        Running
```

```
    tmanager-5798b78f48-gkkcn
r4-infrastructure-prometheus-serv         1/1        Running
    er-c8ddcfdf5-6kd48
statefulset-ricplt-dbaas-server-0         1/1        Running
```

Here, near-real-time RIC pods (ricplt) are running with all of the RIC services installed properly.

Once all ricplt pods are running, we need to check whether the helm list was created or not with the following command:

```
    helm list -A
```

Output:

```
root@srv6:/home/o-ran# helm list -A
NAME                 NAMESPACE   REVISION   STATUS    CHART
r4-a1mediator        ricplt      1          deployed a1mediator
r4-a1alarmmanager    ricplt      1          deployed a1alarmm..
r4-appmgr            ricplt      1          deployed appmgr
r4-dbaas             ricplt      1          deployed dbaas
r4-e2mgr             ricplt      1          deployed e2mgr
r4-e2term            ricplt      1          deployed e2term
r4-infrastructure    ricplt      1          deployed infrastr..
r4-01mediator        ricplt      1          deployed 01mediator
r4-rtmgr             ricplt      1          deployed rtmgr
r4-submgr            ricplt      1          deployed submgr
r4-vespamgr          ricplt      1          deployed vespamgr
```

Here, all (r4) of the helm lists are deployed in the same (ricplt) nampespaces, which are for near-RT-RIC.

Step (4): Compile and establish a seamless connection to the O-RAN E2 (e2-node) simulator sourced from the O-RAN SC simulator project.

Now we compile and build the Connection E2 simulator. First, we clone the git repository of source code from O-RAN SC.

Run the command below:

```
    git clone https://gerrit.o-ran-sc.org/r/sim/e2-interface
```

Now we will install the prerequisites to compile and connect the E2 simulator.

Run the command below:

```
    apt-get install cmake g++ libsctp-dev
```

Now we will modify the Dockerfile with this command to change the directory first.

Run the command below:

```
cd e2-interface/e2sim
cd docker/
vi Dockerfile
```

Now go to the last line of the Dockerfile and edit the CMD part and change the IP address, which is the E2 termination of the RIC side.

"*sleep 100000000*"

Let's start the compilation process of the simulator by taking the first step, which involves creating specific Debian packages that will be utilized in the subsequent Docker stage.

Run the command below:

```
mkdir build
cd build
cmake ..   && make package && cmake .. -DDEV_PKG=1 && make
 package
```

Now copy the Debian directory to create a Docker file to connect to the E2 simulator.

Run the command below:

```
cp *.deb ../e2sm_examples/kpm_e2sm/
cd ../e2sm_examples/kpm_e2sm/
```

Now build oran simulator Docker container with following command.

```
docker build -t oransim:0.0.999 .
docker run -d --name oransim -it oransim:0.0.999
```

Now, to run the simulator, we launch the bash terminal within the container with the following command:

```
docker exec -ti oransim /bin/bash
```

Now check if the simulator is running or not with the following command:

```
kpm\_sim "IP" 36422
```

Note: Replace "IP" to run the simulator. For that, run the command below.

```
    kubectl get services -n ricplt
```

Output:

```
root@srv6:/home/o-ran# kubectl get services -n ricplt
NAME                           TYPE       CLUSTER-IP     PORT(S)
aux-entry                      ClusterIP  10.105.132.177 80/TCP
    ,443/TCP
r4-infrastructur-kong-proxy    NodePort   10.98.2.131
    32080:32080/TCP,32443
r4-infrastructure-alertman:    ClusterIP  10.100.7.125   80/TCP
r4-infrastructure-server       ClusterIP  10.105.33.31   80/TCP
service-ricplt-a1media:-http   ClusterIP  10.102.109.89  10000/
    TCP
service-ricplt-a1media:-rmr    ClusterIP  10.96.120.158  4561/
    TCP,4562/TCP
service-ricplt-alarm:-http     ClusterIP  10.105.220.145 8080/
    TCP
service-ricplt-alarm:-rmr      ClusterIP  10.107.6.137   4560/
    TCP,4561/TCP
service-ricplt-appmgr-http     ClusterIP  10.100.8.154   8080/
    TCP
service-ricplt-appmgr-http     ClusterIP  10.102.179.240 4561/
    TCP,4560/TCP
service-ricplt-dbass-tcp       ClusterIP  None           6379/
    TCP
service-ricplt-e2mgr-http      ClusterIP  10.109.47.153  3800/
    TCP
service-ricplt-e2mgr-rmr       ClusterIP  10.108.234.78  4561/
    TCP,3801/TCP
service-ricplt-e2term-pro..    ClusterIP  10.96.118.19   8088/
    TCP
service-ricplt-e2term-rmr..    ClusterIP  10.102.139.254 4561/
    TCP,38000/TCP
service-ricplt-e2term-sctp..   NodePort   10.102.107.2
    36422:32222/SCTP
service-ricplt-o1mediator-htt  ClusterIP  10.111.228.73  9001/
    TCP,8080/TCP,3000/TCP
```

Now copy the IP from output services in ricplt with the 36422 port and use it in the previous step.

```
    Example:kpm_sim 10.102.107.2 36422
```

Once we run that command, we will see the simulator is connected to near-RT RIC.

To show that the e2 simulator is connected to the near RT RIC follow the below steps.

Copy the E2 manager IP and run the curl command with the following command:

```
    curl -X GET http://"IP":3800/v1/nodeb/states 2>/dev/null|
    jq
```

Now run the following command and take the correct IP:

```
    kubectl get services -n ricplt
```

Output:

```
root@srv6:/home/o-ran# kubectl get services -n ricplt
NAME                          TYPE      CLUSTER-IP     PORT(S)
aux-entry                     ClusterIP 10.105.132.177 80/TCP
    ,443/TCP
r4-infrastructur-kong-proxy   NodePort  10.98.2.131
    32080:32080/TCP,32443
r4-infrastructure-alertman:   ClusterIP 10.100.7.125   80/TCP
r4-infrastructure-server      ClusterIP 10.105.33.31   80/TCP
service-ricplt-a1media:-http  ClusterIP 10.102.109.89  10000/
    TCP
service-ricplt-a1media:-rmr   ClusterIP 10.96.120.158  4561/
    TCP,4562/TCP
service-ricplt-alarm:-http    ClusterIP 10.105.220.145 8080/
    TCP
service-ricplt-alarm:-rmr     ClusterIP 10.107.6.137   4560/
    TCP,4561/TCP
service-ricplt-appmgr-http    ClusterIP 10.100.8.154   8080/
    TCP
service-ricplt-appmgr-http    ClusterIP 10.102.179.240 4561/
    TCP,4560/TCP
service-ricplt-dbass-tcp      ClusterIP None           6379/
    TCP
service-ricplt-e2mgr-http     ClusterIP 10.109.47.153  3800/
    TCP
service-ricplt-e2mgr-rmr      ClusterIP 10.108.234.78  4561/
    TCP,3801/TCP
service-ricplt-e2term-pro..   ClusterIP 10.96.118.19   8088/
    TCP
service-ricplt-e2term-rmr..   ClusterIP 10.102.139.254 4561/
    TCP,38000/TCP
service-ricplt-e2term-sctp..  NodePort  10.102.107.2
    36422:32222/SCTP
service-ricplt-o1mediator-htt ClusterIP 10.111.228.73  9001/
    TCP,8080/TCP,3000/TCP
service-ricplt-o1mediator-tcp NodePort  10.108.87.21
    830:30830/TCP
service-ricplt-rtmgr-http     ClusterIP 10.109.105.95  3800/
    TCP
service-ricplt-rtmgr-rmr      ClusterIP 10.105.222.135 4561/
    TCP,4560/TCP
service-ricplt-submgr-http    ClusterIP None           3800/
    TCP
service-ricplt-submgr-rmr     ClusterIP None           4560/
    TCP,4561/TCP
service-ricplt-vespamgr-http  ClusterIP 10.100.198.129 8080/
    TCP,9095/TCP
```

Now, copy the E2 manager IP (service - ricplt - e2mgr - http) and Replace ”IP” with the e2 manager IP. Run this command in the ric-dep/bin directory. And we will see the E2 simulator connected to RT RIC.

```
    Example: curl -X GET http:// 10.109.47.153:3800/v1/nodeb/
    states 2>/dev/null|jq
```

Step (5): Utilize the dms-cli tool to effortlessly deploy xApps, streamlining the deployment process and enhancing efficiency.

The DMS component plays a vital role in the RIC platform as it manages the near-real-time (NRT) data and metadata utilized by the platform, including policy and configuration data. To interact with the DMS and perform tasks like querying, updating, and deleting data and metadata, the "dms_cli" tool is commonly employed through command-line instructions.

In the context of xApp onboarding services for operators, the xApp onboarder offers a convenient cli tool named "dms_cli." This tool takes in the xApp descriptor and optionally an additional schema file and generates xApp helm charts.

To ensure the successful deployment of any xApp, it is essential to load its corresponding Helm chart into a designated private Helm repository. This repository serves as a prerequisite for xApp deployment.

Run the command below.

```
docker run --rm -u 0 -it -d -p 8090:8080 -e DEBUG=1 -e STORAGE
=local -e STORAGE_LOCAL_ROOTDIR=/charts -v $(pwd)/charts:/
    charts chartmuseum/chartmuseum:latest
```

Note: Run it in the home directory.

Set up the environment variables for a CLI connection using the same port as used above. Now export the chart repository url with the following command:

```
    export CHART\_REPO\_URL=http://0.0.0.0:8090
```

Step(6): Utilize the dms-cli tool to effortlessly deploy xApps, streamlining the deployment process and enhancing efficiency.

To deploy xapp we need to clone the app-manager repository with the following command:

```
    git clone https://gerrit.o-ran-sc.org/r/ric-plt/appmgr -b
    f-release
```

Now change the xapp_onboarder directory to install Python for onboarding xapp with the following command:

```
cd appmgr/xapp\_orchestrater/dev/xapp\_onboarder
```

Note: If pip3 is not installed, install pip packages for python install with the following command:

```
apt-get install python3-pip
```

Note: If the dms_cli binary is already installed, then uninstall it first´using following command.

```
pip3 uninstall xapp\_onboarder
```

Now it's time to install xapp_onboarder. For that first install Python dependencies with the following command:

```
pip3 install ./
```

Now modify the permissions. The instructions are for Python 3.6, and here we will install Python 3.8 with the following commands:

```
ls -la /usr/local/lib/python3.8
chmod -R 755 /usr/local/lib/python3.8
```

Step(7): Compile, onboard, and install the hw-go xapp derived from the O-RAN SC xApp project, incorporating its functionality into the system with precision.

To compile and deploy hw-go xapp, we need source code, and for that, run clone the ric-app repository by using the following command:

```
git clone https://gerrit.o-ran-sc.org/r/ric-app/hw-go
```

Build Docker templates and examples. con:80 is a tag. It could be any url. It will run in the local registry. This will be used later. Follow the below commands:

```
cd hw-go
docker build -t example.com:80/hw-go:1.2 .
```

Now modify *config-file.json* to build a connection with xapp onboard for deployment. We are changing the registry, name, and tag in the configuration. For this, run the command below:

```
vim config/config-file.json
```

Note: Edit the config file in the following steps:

```
registry: nexus3.o-ran-sc.org:10004 to example.com:80.
name: hw-go.
tag: 1.2
```

Now make sure the xapp descriptor config file and the schema file are on my local file system. We can do it by following the command:

```
dms_cli onboard ./config/config-file.json ./config/schema.
json
```

To Install hw-go use the following command:

```
dms_cli install hw-go 1.0.0 ricxapp
```

When hw-go is installed, it will make e2 subscriptions to all e2 nodes that it finds connected. Now we can see ric-xapp is running and connected with the E2 simulator.

Run this command to check if ricxapp pods are running or not.

```
kubectl get pods -n ricxapp
```

Summary: Now the near-RT RIC is successfully deployed with the specifications of the O-RAN software community.

3.2 Infrastructure Preparation and Deployment of non-RT RIC.

Hardware Requirements:

Operating System	Ubuntu 20.04
Core	8 vCPU
Memory	32 GB
Disk	150 GB
GPU	16 GB

Table 3: Hardware Requirements for non-RT RIC

Software Requirements:

- ***Container orchestration:*** Kubernetes- v1.19+

- ***Container runtime:*** Docker and docker-compose (latest)
- ***Kubernetes Package manager*** Helm Chart v3.5
- ***Helm Chart Repository:*** ChartMuseum

3.2.1 Prerequisites Installation of non-RT RIC.

To deploy non-RT RIC, it requires Kubernetes, Kubernetes CNI, Helm Chart, Chartmeusume, and Docker with non-RT RIC dependencies. The following steps will show the installation process for all of them.

1. Install Packages: To install the package, we need ca-certificates, and we will install it with the command below:

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-
certificates curl
```

2. Install Containerd: We have a few container runtimes available. Before we install a container, we need to create its configuration file. We will do it with the following commands:

```
curl -fsSLo containerd-config.toml\https://gist.
githubusercontent.com/oradwell/31
ef858de3ca43addef68ff971f459c2/raw/5099
df007eb717a11825c3890a0517892fa12dbf/containerd-config.
toml
sudo mkdir /etc/containerd
sudo mv containerd-config.toml /etc/containerd/config.toml
```

Now, without any convenience, we can install container ed from the official GitHub repo with the following commands:

```
curl -fsSLo containerd-1.6.14-linux-amd64.tar.gz\https://
github.com/containerd/containerd/releases/download/v1
.6.14/containerd-1.6.14-linux-amd64.tar.gz
```

Note: It will pull compressed binaries, so we need to extract that, which will happen with the below command:

```
sudo tar Cxzvf /usr/local containerd-1.6.14-linux-amd64.
tar.gz
```

Now it's time to install containered as a service. Follow the command below:

```
    sudo curl -fsSLo /etc/systemd/system/containerd.service \
    https://raw.githubusercontent.com/containerd/containerd/
    main/containerd.service
    sudo systemctl daemon-reload
    sudo systemctl enable --now containerd
```

3. Install runc: To create and run containers we need to install a native feature called runc which is a low level of container runtime. Following command will install it:

```
curl -fsSLo runc.amd64 \
  https://github.com/opencontainers/runc/releases/download/v1
    .1.3/runc.amd64
sudo install -m 755 runc.amd64 /usr/local/sbin/runc
```

4. Install network plugins CNI: We install network interface plugins from their official git repository. Following commands will install CNI:

```
curl -fsSLo cni-plugins-linux-amd64-v1.1.1.tgz \
  https://github.com/containernetworking/plugins/releases/
    download/v1.1.1/cni-plugins-linux-amd64-v1.1.1.tgz
sudo mkdir -p /opt/cni/bin
sudo tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.1.1.tgz
```

5.Enable IPv4 forwarding and configure iptables for bridged network traffic.

To enable proper functionality, ensure the overlay and *br_netfilter* kernel modules are enabled, and grant iptables the capability to inspect bridged network traffic.

Run the following command:

```
    cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
    overlay
    br_netfilter
    EOF

    sudo modprobe -a overlay br_netfilter
```

The required sysctl parameters must be configured for the setup, ensuring their persistence across reboots. Run the following commands:

```
    cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
    net.bridge.bridge-nf-call-iptables  = 1
    net.bridge.bridge-nf-call-ip6tables = 1
    net.ipv4.ip_forward                 = 1
    EOF

# Apply sysctl params without reboot
    sudo sysctl[U+FFFD]system
```

6. Install Kubectl, Kubelet,Kubeadm: Now we are ready to install kuberenetes. We need to make sure that their versions are compatible with the following commands:

```
# Add Kubernetes GPG key
    sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-
    keyring.gpg \
    https://packages.cloud.google.com/apt/doc/apt-key.gpg

# Add Kubernetes apt repository
    echo "deb [signed-by=/usr/share/keyrings/kubernetes-
    archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes
    -xenial main" \
     | sudo tee /etc/apt/sources.list.d/kubernetes.list

# Fetch package list
    sudo apt-get update
    sudo apt-get install -y kubelet kubeadm kubectl

# Prevent them from being updated automatically
    sudo apt-mark hold kubelet kubeadm kubectl
```

7. Ensure swap is disabled: In order to comply with Kubernetes' lack of support for the swap feature, it is necessary to disable it. Run the following commands:

```
    swapon --show
    sudo swapoff -a
```

8. Create the cluster using kubeadm: Our non-RT RIC will work on single node clusters or multi node clusters. This research project is based on a single-node cluster.

With just a single command, the cluster can be initialized. However, in single-node environments, it may not offer full functionality until certain modifications are made. It is important to note that we are providing the "–pod-network-cidr" parameter, as mandated by our CNI plugin (Flannel). Run the following commands:

```
    sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

9. Configure kubectl: Configuring Kubectl is important to get access to the cluster. Run the following commands:

```
    mkdir -p $HOME/.kube
    sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
    sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

10. Untaint node: To ensure pods can be deployed to our single-node cluster without any issues, it is necessary to untaint the node. Failure to do so may result in pods being stuck in a pending state. Run the commands below to avoid this issue:

```
kubectl taint nodes --all node-role.kubernetes.io/master-
kubectl taint nodes --all node-role.kubernetes.io/control-
plane-
```

Note: Sometimes it will not work. At that time, run the following command:

```
kubectl get nodes
kubectl taint nodes "IP" node-role.kubernetes.io/control-
plane=:NoSchedule
```

11. Install a CNI pligin: To enable networking functionality, the installation of a Container Network Interface (CNI) plugin is required. In our case, we are installing flannel as the chosen plugin with the following command:

```
kubectl apply -f https://raw.githubusercontent.com/coreos/
flannel/master/Documentation/kube-flannel.ym
```

12. Install Helm: For non-RT RIC, we need helm v3.5. We install our package by following the command:

```
curl https://raw.githubusercontent.com/helm/helm/master/
scripts/get-helm-3 | bash
```

13. Install ChartMuseum: To run the chartmuseum we insrtalled packages with the following command:

```
curl https://raw.githubusercontent.com/helm/chartmuseum/
main/scripts/get-chartmuseum | bash
```

14. Install CSI driver: For the storage to work, need to install the Container Storage Interface driver. Here used CSI is OpenEBS with the command below:

```
helm repo add openebs https://openebs.github.io/charts
kubectl create namespace openebs
helm --namespace=openebs install openebs openebs/openebs
```

15. Run Kubectl proxy: To avoid the localhost error for a single node cluster, open a new terminal and run the following command:

```
kubectl proxy --port=8080
```

Summary: To sum up, our Kubernetes is now installed perfectly. To see if the Kubernetes pods are running or not, run the following command:

```
kubectl get po -n kube-system
```

Output:

```
root@srv:/home/o-ran# kubectl get pods -n kube-system
NAME                              READY   STATUS    RESTARTS  AGE
coredns-5644d7b6d9-58xml          1/1     Running   2         12d
coredns-5644d7b6d9-d446p          1/1     Running   2         12d
etcd-srv6.5g.dn.th.koeln.de       1/1     Running   2         12d
kube-apiserver-srv6.5g.dn.th...   1/1     Running   2         12d
kube-flannel-ds-rr56g             1/1     Running   2         12d
kube-proxy-dgxvm                  1/1     Running   2         12d
kube-scheduler-srv6.5g.dn.th....  1/1     Running   2         12d
```

The Kubernetes pods are running, and Kubernetes is installed. Now Kubernetes is ready to install non-RT RIC.

3.2.2 Deployment of non-RT RIC.

After the successful installation of prerequisites, it is now time to deploy non-RT RIC. As we are following G-Release, the next steps will go through o-ran software community guidelines.

1. Preparation: Downloaded the specific repository (it/dep) based on G-release. Make sure the branch exists before cloning from the master branch. For that run the following command:

```
git clone "https://gerrit.o-ran-sc.org/r/it/dep" -b g-
release
or
git clone "https://gerrit.o-ran-sc.org/r/it/dep"
```

2. Installation Component Configuration:

It is simple to configure the installation of nonrtric components, such as the controller and A1 simulators. All you need to do is make adjustments to a specific file known as the Helm package override file to customize the installation according to your preferences. Run the following command and use any editor to edit the example_recipe.yaml file:

```
\"editor" dep/RECIPE\_EXAMPLE/NONRTRIC/example\_recipe.
yaml
```

The file shown below is part of the override example_recipe.yaml.

To enable installation, set any parameters starting with 'install' to 'true', and to disable installation, set them to 'false'.

You can only enable either the install Non-rt- ric gateway or the install Kong parameters at the same time.

The file also contains other parameters that might need to be adjusted for a specific environment, such as the hostname, namespace, and port of the message router. However, this guide does not provide instructions for configuring these integration details.

Editor override file:

```
nonrtric:
installPms: true
installA1controller: true
installA1simulator: true
installControlpanel: true
installInformationservice: true
installRappcatalogueservice: true
installRappcatalogueEnhancedservice: true
installNonrtricgateway: true
installKong: false
installDmaapadapterservice: true
installDmaapmediatorservice: true
installHelmmanager: true
installOruclosedlooprecovery: true
installOdusliceassurance: true
installCapifcore: true

volume1:
# Set the size to 0 if you do not need the volume (if you are
    using Dynamic Volume Provisioning)
size: 2Gi
storageClassName: pms-storage

volume2:
# Set the size to 0 if you do not need the volume (if you are
    using Dynamic Volume Provisioning)
size: 2Gi
storageClassName: ics-storage

volume3:
size: 1Gi
storageClassName: helmmanager-storage
...
...
...
```

3. Installation: There is a script that uses the helm command to pack and install the components. The installation requires a values override file, like the one mentioned before. To run this example, follow these steps:

```
    sudo dep/bin/deploy-nonrtric -f
    dep/nonrtric/RECIPE\_EXAMPLE/example\_recipe.yaml
```

4. Result of the Installation:

During the installation process, a single Helm release will be created, and all associated Kubernetes objects will be placed within a designated name space. The predefined name space for these objects is 'nonrtric' and cannot be modified.

After the installation is complete, we can verify that by following output of the Kubernetes objects that have been created by using the 'kubectl' command. For instance, if all components are enabled, you can check the deployed pods using the following example command:

```
    kubectl get po -n nonrtric
```

Output:

```
root@srv5;/home/o-ran# kubectl get po -n nonrtric
NAME                         READY  STATUS   RESTARTS     AGE
a1-sim-std-0-7d7d6d5b69      1/1    Running  3(98m ago)   3h 55m
-gxx2d
topology-6c5cd99d6d-q4p8r    1/1    Running  3(98m ago)   3h 55m
a1-sim-osc-1-5bb7478885-     1/1    Running  3(98m ago)   3h 55m
7ssh8
a1-sim-std2-0-64cc667968-    1/1    Running  3(98m ago)   3h 55m
cnzs7r
a1-sim-std-1-6d7b644cbb-g    1/1    Running  3(98m ago)   3h 55m
749h
nonrtricgateway-689d9cf595   1/1    Running  6(96m ago)   3h 55m
-mw4vg
dmaapadapterservice-0        1/1    Running  3(98m ago)   3h 55m
a1-sim-osc-0-547cc8fc84-     1/1    Running  3(98m ago)   3h 55m
86vv4
informationservice-0         1/1    Running  3(98m ago)   3h 55m
rappcatalogueservice-8844f   1/1    Running  3(98m ago)   3h 55m
9469-ppwt7
oran-nonrtric-odu-app-86c5   1/1    Running  3(98m ago)   3h 55m
d494fb-pg2mr
oran-nonrtric-kong-594db9    2/2    Running  11(96m ago) 3h 55m
cb8b-h4m4j
oran-nonrtric-odu-app-ics-   1/1    Running  3(98m ago)   3h 55m
version....
a1-sim-std2-1b668b97df-98    1/1    Running  3(98m ago)   3h 55m
cmd
helmanager-0                 1/1    Running  5(98m ago)   3h 55m
dmaapmediatorservice-0       1/1    Running  4(95m ago)   3h 55m
controlpanel-6fb4f88778-     1/1    Running  15(95m ago) 3h 55m
rhmph
oru-app-8db46d4cf-jd66b      1/1    Running  1(105m ago) 3h 55m
```

Summary: Now we can see that all pods are in the running stage. Where the A1 simulator, controller, helmanager and other necessary Kubernetes pods are active. It means the non-RT RIC platform is ready to use.

3.3 Deployment Status Testing.

To check the deployment status of both RAN intelligent controllers, follow the following steps:

3.3.1 near-RT RIC Status.

Kubernetes Pods status: To ensure that near-RT RIC is deployed or not, we need to check pod status. We can see that all pods are running. So, near-RT RIC has been deployed successfully. Run the following command.

```
    kubectl get po -n ricplt
```

Output:

```
root@srv6:/home/o-ran# kubectl get po -n ricplt
NAME                                  READY      STATUS   AGE
deployment-ricplt-a1mediator-669cc7   1/1        Running  18d
4647-t22qp
deployment-ricplt-alarmmanager-577    1/1        Running  18d
85458dd-p95s4
deployment-ricplt-appmgr-77986c9c     1/1        Running  18d
bb-l6b21
deployment-ricplt-e2mgr-5dd878f58     1/1        Running  18d
 b-npdff
deployment-ricplt-e2term-alpha-68     1/1        Running  18d
 98f8696d-5smjj
deployment-ricplt-01mediator-5ddd6    1/1        Running  18d
6b4d6-5xlv9
deployment-ricplt-rtmgr-788975975     1/1        Running  18d
 b-ghlgs
deployment-ricplt-submgr-68fc6564     1/1        Running  18d
88-6wnd6
deployment-ricplt-vespamgr-84f7d8     1/1        Running  18d
7dfb-rq7zx
r4-infrastructure-kong-7995f4679b     2/2        Running  18d
-bs95n
r4-infrastructure-prometheus-aler     2/2        Running  18d
tmanager-5798b78f48-gkkcn
r4-infrastructure-prometheus-serv     1/1        Running  18d
er-c8ddcfdf5-6kd48
statefulset-ricplt-dbaas-server-0     1/1        Running  18d
```

Here,all necessary service pods are running. Importantly, a1 mediator, armanager, appmgr and 01 mediator are running. Which are responsible for building connections with non-RT RIC.

3.3.2 non-RT RIC Status.

For non-RT RIC, we have seen that all pods are running in previous steps. According to o-ran allaince instruction after the deployment of non- RT RIC part, we will browse it and have a control panel *[Figure 5]*.

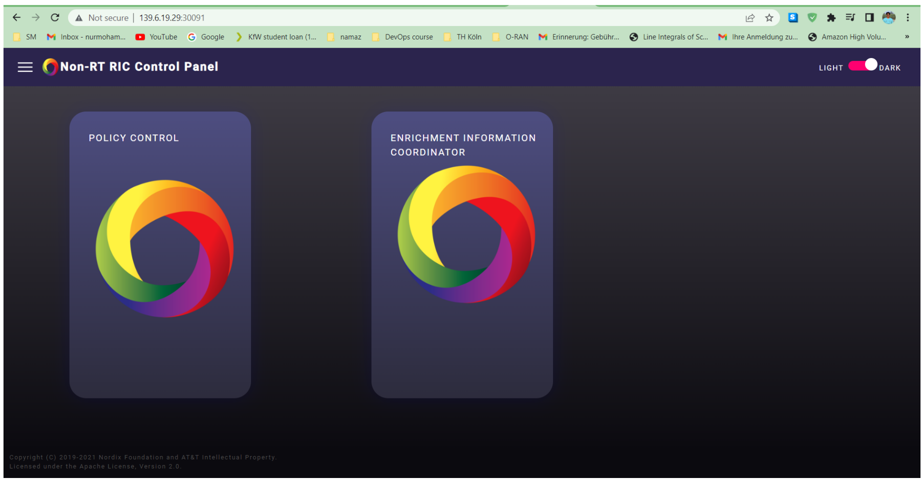

Not secure | 139.6.19.29:30091
Non-RT RIC Control Panel
LIGHT
DARK
POLICY CONTROL
ENRICHMENT INFORMATION COORDINATOR
Copyright (C) 2019-2021 Nordix Foundation and AT&T Intellectual Property.
Licensed under the Apache License, Version 2.0.

Figure 5: Non-RT RIC Control Panel

This non-RT RIC control panel *[Figure 5]* will represent the policy control and enrichment information coordinator.

Policy Control:

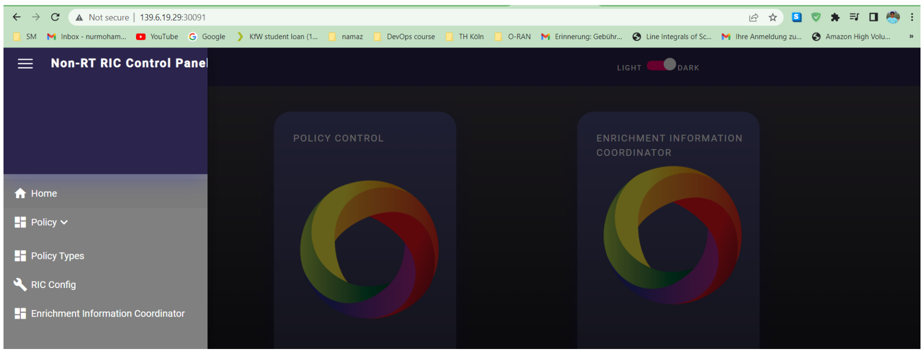
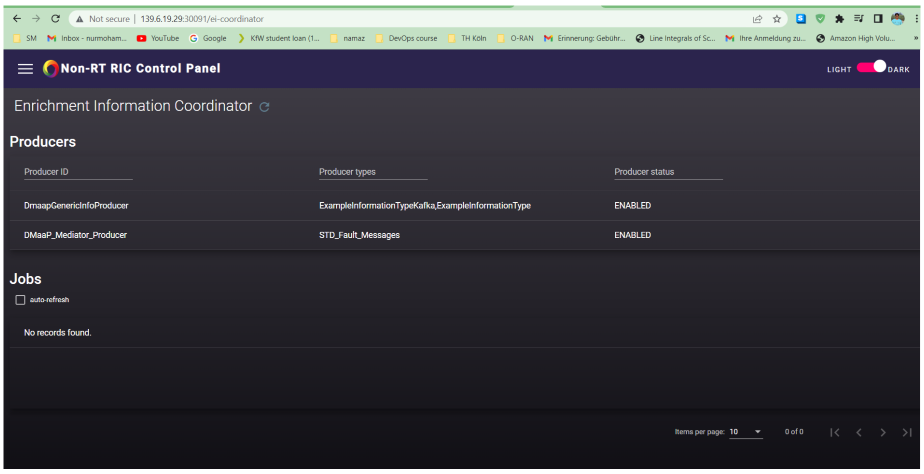

Not secure | 139.6.19.29:30091
Non-RT RIC Control Panel
LIGHT
DARK
Home
Policy
Policy Types
RIC Config
Enrichment Information Coordinator
POLICY CONTROL
ENRICHMENT INFORMATION COORDINATOR

Figure 6: Policy Control functions.

In policy control, policy, policy type, and RIC configuration *[Figure 6]* will appear, and the coordinator will highlight them with ID, type, and status.

Enrichment Information Coordinator:


Not secure | 139.6.19.29:30091/ei-coordinator

Non-RT RIC Control Panel

Enrichment Information Coordinator

Producers

Producer ID	Producer types	Producer status
DmaapGenericInfoProducer	ExampleInformationTypeKafka,ExampleInformationType	ENABLED
DMaaP_Mediator_Producer	STD_Fault_Messages	ENABLED

Jobs

auto-refresh

No records found.

Items per page: 10 0 of 0

Figure 7: Non-RT RIC Enrichment information Coordinator

Now, on the coordinator page *[Figure 7]*, there is also an option to create jobs for further work.

4 Testing and Analysis

4.1 Connection set-up.

To build the connection between near-RT RIC and non-RT RIC we need to follow a few steps. Which is guided by the software community of o-ran Alliance.

Assign external IP to near and non RT RIC services: To connect a near-RT RIC with a non-RT RIC platform, it is mandatory to assign an external IP address to all components of both RIC.

For near-RT RIC The following command will set an external IP:

```
kubectl patch svc "service-ricplt-a1mediator-http" -n "
ricplt" -p '{"spec":{"externalIPs":["139.6.19.30"]}}'
```

After running the command, we can see that the external IP assigned or not with the following command:

```
kubectl get services -n ricplt
```

Output:

```
root@srv6:/home/o-ran# kubectl get services -n ricplt

NAME               TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
aux-entry          ClusterIP  10.105.132.177  139.6.19.30   80/TCP
    ,443/TCP
r4-infrastr        NodePort   10.98.2.131     139.6.19.30
    32080:32080/TCP,32443
uctur-kong-proxy
r4-infrastructu    ClusterIP  10.100.7.125    139.6.19.30   80/TCP
re-prometheus-alertmanager
r4-infrastruct     ClusterIP  10.105.33.31    139.6.19.30   80/TCP
ure-prometheus-server
service-ricplt     ClusterIP  10.102.109.89   139.6.19.30   10000/
    TCP
-a1mediator-http
service-ricplt     ClusterIP  10.96.120.158   139.6.19.30   4561/
    TCP,4562/TCP
-a1mediator-rmr
service-ricplt     ClusterIP  10.105.220.145  139.6.19.30   8080/
    TCP
-alarmanager-http
service-ricplt     ClusterIP  10.107.6.137    139.6.19.30   4560/
    TCP,4561/TCP
-alarmanager-rmr
service-ricplt     ClusterIP  10.100.8.154    139.6.19.30   8080/
    TCP
-appmgr-http
service-ricplt     ClusterIP  10.102.179.240  139.6.19.30   4561/
    TCP,4560/TCP
-appmgr-rmr
service-ricplt     ClusterIP  None            139.6.19.30   6379/
    TCP
-dbass-tcp
service-ricplt     ClusterIP  10.109.47.153   139.6.19.30   3800/
    TCP
-e2mgr-http
service-ricplt     ClusterIP  10.108.234.78   139.6.19.30   4561/
    TCP,3801/TCP
-e2mgr-rmr
service-ricplt     ClusterIP  10.96.118.19    139.6.19.30   8088/
    TCP
-e2term-prometheus-alpha
service-ricplt     ClusterIP  10.102.139.254  139.6.19.30   4561/
    TCP,38000/TCP
-e2term-rmr-alpha
service-ricplt     NodePort   10.102.107.2    139.6.19.30
    36422:32222/SCTP
-e2term-sctp-alpha
service-ricplt     ClusterIP  10.111.228.73   139.6.19.30   9001/
    TCP,8080/TCP,3000/TCP
-o1mediator-http
service-ricplt     NodePort   10.108.87.21    139.6.19.30
    830:30830/TCP
-o1mediator-tcp-netconf
service-ricplt     ClusterIP  10.109.105.95   139.6.19.30   3800/
    TCP
-rtmgr-http
service-ricplt     ClusterIP  10.105.222.135  139.6.19.30   4561/
    TCP,4560/TCP
-rtmgr-rmr
service-ricplt     ClusterIP  None            139.6.19.30   3800/
    TCP
```

```
-submgr-http
service-ricplt   ClusterIP None           139.6.19.30  4560/
    TCP,4561/TCP
-submgr-rmr
service-ricplt   ClusterIP 10.100.198.129 139.6.19.30  8080/
    TCP,9095/TCP
-vespamgr-http
```

In the output of the kubectl services, it shows all pods are assigned an external IP of ”139.6.19.30”.

For non-RT RIC Now we will asigned external IP for non-RT RIC and in this case IP is ”139.6.19.26”. The following command will set an external IP:

```
    kubectl patch svc "service-ricplt-a1mediator-http" -n "
    ricplt" -p '{"spec":{"externalIPs":["139.6.19.29"]}}'
```

To check if it works or not, run the following command:

```
    kubectl ger services -n nonrtric
```

Output:

```
root@srv5;/home/o-ran# kubectl get po -n nonrtric
NAME           TYPE      CLUSTER-IP     EXTERNAL-IP  PORT(S)
a1-sim-std2-0  ClusterIP 10.152.183.167 139.6.19.29  8085/TCP
    ,8185/TCP
helmmanager    ClusterIP 10.152.183.29  139.6.19.29  8112/TCP
topology       NodePort  10.152.183.220 139.6.19.29
    3001:32001/TCP
a1-sim-osc-1   ClusterIP 10.152.183.127 139.6.19.29  8085/TCP
    ,8185/TCP
a1-sim-osc-0   ClusterIP 10.152.183.224 139.6.19.29  8085/TCP
    ,8185/TCP
a1-sim-std-0   ClusterIP 10.152.183.44  139.6.19.29  8085/TCP
    ,8185/TCP
information    ClusterIP 10.152.183.51  139.6.19.29  9082/TCP
    ,9083/TCP
service
a1-sim-std-1   ClusterIP 10.152.183.178 139.6.19.29  8085/TCP
    ,8185/TCP
oran-nonrtric  NodePort  10.152.183.205 139.6.19.29
    8444:30634/TCP
-kong-admin
dmaapadapter   ClusterIP 10.152.183.136 139.6.19.29  9087/TCP
    ,9088/TCP
service
controlpanel   NodePort  10.152.183.174 139.6.19.29
    8182:30091/TCP,8082:30092/TCP
a1-sim-std-1   ClusterIP 10.152.183.71  139.6.19.29  8085/TCP
    ,8185/TCP
oran-nonrtric  LoadBal.. 10.152.183.232 139.6.19.29  80:30742/
    TCP,443:32516/TCP
-kong-proxy
oru-app        NodePort  10.152.183.207 139.6.19.29
    830:30835/TCP
```

```
oran-nonrtric   ClusterIP  10.152.183.149  139.6.19.29   80/TCP
-odu-app
nonrtricgatew   NodePort   10.152.183.13   139.6.19.29
    9090:30093/TCP
dmaapmediatore  ClusterIP  10.152.183.60   139.6.19.29   8085/TCP
    ,8185/TCP
service
oran-nonrtric-  ClusterIP  10.152.183.14   139.6.19.29   8095/TCP
odu-app-ics
-version
rappcatalogue   ClusterIP  10.152.183.251  139.6.19.29  9085/TCP
    ,9086/TCP
service
```

Now it shows that external IPs are assigned, and now both RIC are able to communicate with both of them.

4.2 Create Policy.

The following steps will show the process of creating a policy.

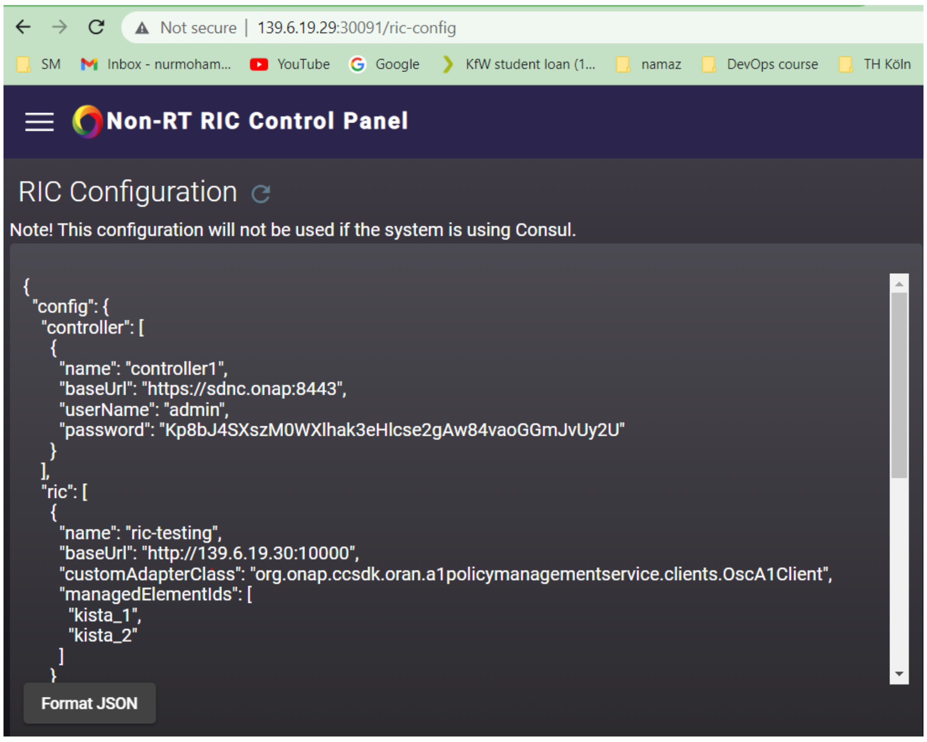
Edit application_configuration.json file (non-rt ric): Run the following to edit this json file:

```
    sudo vim /var/nonrtric/pms-storage/application\
    _configuration.json
```

Edit this json file by using the following codes:

```
{
    "config": {
      "controller": [
        {
          "name": "controller1",
          "baseUrl": "https://sdnc.onap:8443",
          "userName": "admin",
          "password": "
    Kp8bJ4SXszM0WXlhak3eHlcse2gAw84vaoGGmJvUy2U"
        }
      ],
      "ric": [
        {
          "name": "ric-testing",
          "baseUrl": "http://$NONRTRIC_IP:10000",
          "customAdapterClass": "org.onap.ccsdk.oran.
    a1policymanagementservice.clients.OscA1Client",
          "managedElementIds": [
            "kista_1",
            "kista_2"
          ]
        }
      ],
      "streams_publishes": {
        "dmaap_publisher": {
          "type": "message_router",
          "dmaap_info": {
            "topic_url": "http://message-router:3904/events/A1
    -POLICY-AGENT-WRITE"
          }
        }
      },
      "streams_subscribes": {
        "dmaap_subscriber": {
          "type": "message_router",
          "dmaap_info": {
            "topic_url": "http://message-router:3904/events/A1
    -POLICY-AGENT-READ/users/policy-agent?timeout=15000&limit
    =100"
          }
        }
      }
    }
  }
```

Now, if we go to the non-RT RIC control panel, we will see that the configuration is updated perfectly.



Note! This configuration will not be used if the system is using Consul.

```
{
 "config": {
  "controller": [
   {
    "name": "controller1",
    "baseUrl": "https://sdnc.onap:8443",
    "userName": "admin",
    "password": "Kp8bJ4SXszM0WXlhak3eHlcse2gAw84vaoGGmJvUy2U"
   }
  ],
  "ric": [
   {
    "name": "ric-testing",
    "baseUrl": "http://139.6.19.30:10000",
    "customAdapterClass": "org.onap.ccsdk.oran.a1policymanagementservice.clients.OscA1Client",
    "managedElementIds": [
     "kista_1",
     "kista_2"
    ]
   }
```

Figure 8: RIC configuration updates.

[Figure 8] is the non-RT RIC control panel. Now reload the page, and we will see that the RIC configuration is updated with the application file.

Create a policy in non-RT RIC and ONAP components installation:

To create a policy, we need to update the policy schema file. This file is provided by default. Apart from that, ONAP has developed a policy-creating platform. So, before we create policies, we need to set up a few dependencies, which are provided by ONAP. Run the following command to deploy Helm for ONAP services.

```
helm deploy \
   --debug onap local/onap \
   --namespace onap \
   -f /root/dep/smo-install/helm-override/default/onap-
   override.yaml \
   --set global.persistence.mountPath="/dockerdata-nfs/
   deployment-$timestamp" \
   --set dmaap.message-router.message-router-zookeeper.
   persistence.mountPath="/dockerdata-nfs/deployment-
   $timestamp" \
   --set dmaap.message-router.message-router-kafka.
   persistence.mountPath="/dockerdata-nfs/deployment-
   $timestamp"
```

Now non-rt ric is ready to create policy. First we need to edit *policy_schema_ratecontrol.json* file by replacing following script.

```
{
"name": "Policy for Rate Control",
  "policy_type_id":21003,
  "description":"This policy is associated with rate control.
    Entities which support this policy type must accept the
   following policy inputs (see the payload for more specifics
    ) : class, which represents the class of traffic for which
     the policy is being enforced",

  "create_schema":{
      "$schema":"http://json-schema.org/draft-07/schema#",
      "type":"object",
      "additionalProperties":false,
      "required":["class"],
      "properties":{
          "class":{
              "type":"integer",
              "minimum":1,
              "maximum":256,
              "description":"integer id representing class to
    which we are applying policy"
          },
          "enforce":{
              "type":"boolean",
              "description": "Whether to enable or disable
    enforcement of policy on this class"
          },
          "window_length":{
              "type":"integer",
              "minimum":15,
              "maximum":300,
              "description":"Sliding window length in seconds"
          },
          "trigger_threshold":{
              "type":"integer",
              "minimum":1
          },
          "blocking_rate":{
              "type":"number",
              "minimum":0,
              "maximum":100
          }

      }
  },

  "downstream_schema":{
      "type":"object",
      "additionalProperties":false,
      "required":["policy_type_id", "policy_instance_id", "
    operation"],
      "properties":{
          "policy_type_id":{
              "type":"integer",
              "enum":[21000]
          },
          "policy_instance_id":{
              "type":"string"
          },
          "operation":{
              "type":"string",
              "enum":["CREATE", "UPDATE", "DELETE"]
          },
          "payload":{
              "$schema":"http://json-schema.org/draft-07/
    schema#",
              "type":"object",
              "additionalProperties":false,
              "required":["class"],
              "properties":{
                  "class":{
                      "type":"integer",
                      "minimum":1,
                      "maximum":256,
                      "description":"integer id representing
    class to which we are applying policy"
                  },
                  "enforce":{
                      "type":"boolean",
                      "description": "Whether to enable or
    disable enforcement of policy on this class"
                  },
                  "window_length":{
                      "type":"integer",
                      "minimum":15,
                      "maximum":300,
                      "description":"Sliding window length in
    seconds"
                  },
                  "trigger_threshold":{
                      "type":"integer",
                      "minimum":1
                  },
                  "blocking_rate":{
                      "type":"number",
                      "minimum":0,
                      "maximum":100
                  }


              }
          }
      }
  },
  "notify_schema":{
      "type":"object",
      "additionalProperties":false,
      "required":["policy_type_id", "policy_instance_id", "
    handler_id", "status"],
      "properties":{
          "policy_type_id":{
              "type":"integer",
              "enum":[21000]
          },
          "policy_instance_id":{
              "type":"string"
```

```
                },
                "handler_id":{
                    "type":"string"
                },
                "status":{
                    "type":"string",
                    "enum":["OK", "ERROR", "DELETED"]
                }
            }
        }
}
```

Now curl this file, and once it's done, a policy will be created, and to see the policy, we need to go to policy control on the RIC platform: For that, first run the following command to create a policy:

```
    curl -X PUT "http://localhost/a1-p/policytypes/21003/" -H
    "Content-Type: application/json" -d @policy\_schema\
    _ratecontrol.json
```

Now go to the non-RT RIC control panel and enter the option policy type bar. Refresh the page, and we will see that the policy has been created. In my case, the policy name is 21003 *[Figure 9]*

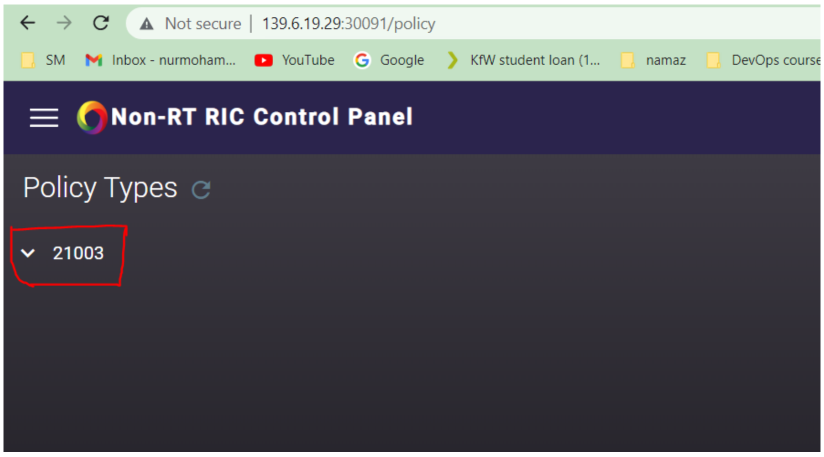


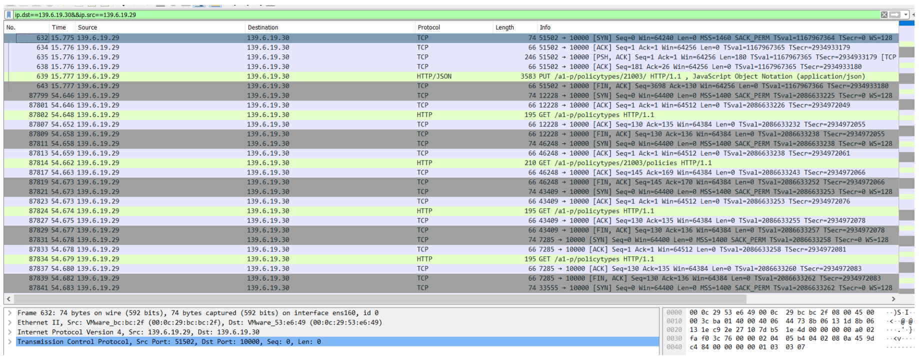
Figure 9: Created Policy 21003

4.3 Result.

In the next step, we need to acknowledge the significance of the policy created earlier. This policy represents the connection between the entities involved. With the policy in place, data packets will be transferred between them. To understand this data transfer better, we will use Wireshark. Wireshark captures and analyzes the packets exchanged during communication. It helps us examine the contents, protocols, and interactions within the captured packets. By analyzing the packet captures, we can gather information such as source and destination addresses, timing, headers, and payload contents. We can also identify any issues or errors that may occur during the data exchange. Using Wireshark, we can validate the expected behavior, troubleshoot problems, and ensure the smooth and secure transmission of data packets between the components. Wireshark provides valuable insights into the network traffic, helping us understand the communication process between the entities involved.

4.3.1 Wireshark Capture and Analysis.

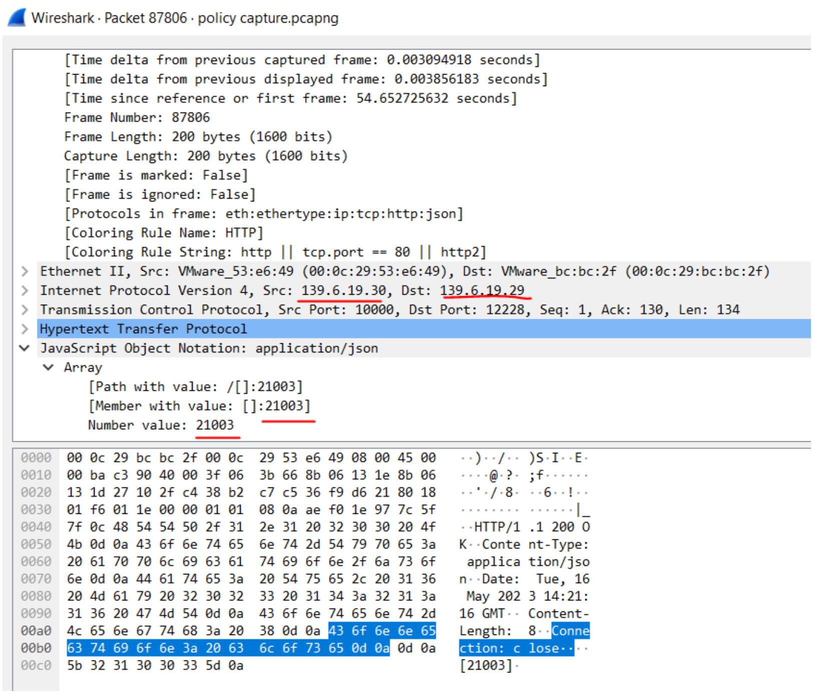
In the following Wireshark *[Figure 10]* capture, we will see that near-RT RIC and non-RT RIC transferring packets. Here, 139.6.19.30 is for near-RT RIC and 139.6.19.29 is for non-RT RIC platforms.



```
ip.dst==139.6.19.30&&ip.src==139.6.19.29

No.    Time   Source       Destination  Protocol  Length Info
  632 15.775 139.6.19.29  139.6.19.30  TCP         74 51502 → 10000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1167967364 TSecr=0 WS=128
  634 15.776 139.6.19.29  139.6.19.30  TCP         66 51502 → 10000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1167967365 TSecr=2934933179
  635 15.776 139.6.19.29  139.6.19.30  TCP        246 51502 → 10000 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=180 TSval=1167967365 TSecr=2934933179 [TCP
  638 15.776 139.6.19.29  139.6.19.30  TCP         66 51502 → 10000 [ACK] Seq=181 Ack=26 Win=64256 Len=0 TSval=1167967365 TSecr=2934933180
  639 15.777 139.6.19.29  139.6.19.30  HTTP/JSON 3583 PUT /a1-p/policytypes/21003/ HTTP/1.1 , JavaScript Object Notation (application/json)
  643 15.777 139.6.19.29  139.6.19.30  TCP         66 51502 → 10000 [FIN, ACK] Seq=3698 Ack=130 Win=64256 Len=0 TSval=1167967366 TSecr=2934933180
87799 54.646 139.6.19.29  139.6.19.30  TCP         74 12228 → 10000 [SYN] Seq=0 Win=64400 Len=0 MSS=1400 SACK_PERM TSval=2086633225 TSecr=0 WS=128
87801 54.646 139.6.19.29  139.6.19.30  TCP         66 12228 → 10000 [ACK] Seq=1 Ack=1 Win=64512 Len=0 TSval=2086633226 TSecr=2934972049
87802 54.648 139.6.19.29  139.6.19.30  HTTP       195 GET /a1-p/policytypes HTTP/1.1
87807 54.652 139.6.19.29  139.6.19.30  TCP         66 12228 → 10000 [ACK] Seq=130 Ack=135 Win=64384 Len=0 TSval=2086633232 TSecr=2934972055
87809 54.658 139.6.19.29  139.6.19.30  TCP         66 12228 → 10000 [FIN, ACK] Seq=130 Ack=136 Win=64384 Len=0 TSval=2086633238 TSecr=2934972055
87811 54.658 139.6.19.29  139.6.19.30  TCP         74 46248 → 10000 [SYN] Seq=0 Win=64400 Len=0 MSS=1400 SACK_PERM TSval=2086633238 TSecr=0 WS=128
87813 54.659 139.6.19.29  139.6.19.30  TCP         66 46248 → 10000 [ACK] Seq=1 Ack=1 Win=64512 Len=0 TSval=2086633238 TSecr=2934972061
87814 54.662 139.6.19.29  139.6.19.30  HTTP       210 GET /a1-p/policytypes/21003/policies HTTP/1.1
87817 54.663 139.6.19.29  139.6.19.30  TCP         66 46248 → 10000 [ACK] Seq=145 Ack=169 Win=64384 Len=0 TSval=2086633243 TSecr=2934972066
87819 54.673 139.6.19.29  139.6.19.30  TCP         66 46248 → 10000 [FIN, ACK] Seq=145 Ack=170 Win=64384 Len=0 TSval=2086633252 TSecr=2934972066
87821 54.673 139.6.19.29  139.6.19.30  TCP         74 43409 → 10000 [SYN] Seq=0 Win=64400 Len=0 MSS=1400 SACK_PERM TSval=2086633253 TSecr=0 WS=128
87823 54.673 139.6.19.29  139.6.19.30  TCP         66 43409 → 10000 [ACK] Seq=1 Ack=1 Win=64512 Len=0 TSval=2086633253 TSecr=2934972076
87824 54.674 139.6.19.29  139.6.19.30  HTTP       195 GET /a1-p/policytypes HTTP/1.1
87827 54.675 139.6.19.29  139.6.19.30  TCP         66 43409 → 10000 [ACK] Seq=130 Ack=135 Win=64384 Len=0 TSval=2086633255 TSecr=2934972078
87829 54.677 139.6.19.29  139.6.19.30  TCP         66 43409 → 10000 [FIN, ACK] Seq=130 Ack=136 Win=64384 Len=0 TSval=2086633257 TSecr=2934972078
87831 54.678 139.6.19.29  139.6.19.30  TCP         74 7285 → 10000 [SYN] Seq=0 Win=64400 Len=0 MSS=1400 SACK_PERM TSval=2086633258 TSecr=0 WS=128
87833 54.678 139.6.19.29  139.6.19.30  TCP         66 7285 → 10000 [ACK] Seq=1 Ack=1 Win=64512 Len=0 TSval=2086633258 TSecr=2934972081
87834 54.679 139.6.19.29  139.6.19.30  HTTP       195 GET /a1-p/policytypes HTTP/1.1
87837 54.680 139.6.19.29  139.6.19.30  TCP         66 7285 → 10000 [ACK] Seq=130 Ack=135 Win=64384 Len=0 TSval=2086633260 TSecr=2934972083
87839 54.682 139.6.19.29  139.6.19.30  TCP         66 7285 → 10000 [FIN, ACK] Seq=130 Ack=136 Win=64384 Len=0 TSval=2086633262 TSecr=2934972083
87841 54.683 139.6.19.29  139.6.19.30  TCP         74 33555 → 10000 [SYN] Seq=0 Win=64400 Len=0 MSS=1400 SACK_PERM TSval=2086633262 TSecr=0 WS=128

> Frame 632: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface ens160, id 0
> Ethernet II, Src: VMware_bc:bc:2f (00:0c:29:bc:bc:2f), Dst: VMware_53:e6:49 (00:0c:29:53:e6:49)
> Internet Protocol Version 4, Src: 139.6.19.29, Dst: 139.6.19.30
> Transmission Control Protocol, Src Port: 51502, Dst Port: 10000, Seq: 0, Len: 0

0000  00 0c 29 53 e6 49 00 0c  29 bc bc 2f 08 00 45 00   ··)S·I··
0010  00 3c ba 01 40 00 40 06  44 73 8b 06 13 1d 8b 06   ·<··@·@·
0020  13 1e c9 2e 27 10 7d b5  1e 4d 00 00 00 00 a0 02   ····.'·}·
0030  fa f0 3c 76 00 00 02 04  05 b4 04 02 08 0a 45 9d   ··<v····
0040  c4 84 00 00 00 00 01 03  03 07                     ········
```

Figure 10: Packets after creating Policy 21003

Now the *[Figure 11]* is about the transferred packet. Here, packet number 87806 is a transfer from non-RT RIC to near-RT RIC, which is a policy packet.


Wireshark · Packet 87806 · policy capture.pcapng

```
    [Time delta from previous captured frame: 0.003094918 seconds]
    [Time delta from previous displayed frame: 0.003856183 seconds]
    [Time since reference or first frame: 54.652725632 seconds]
    Frame Number: 87806
    Frame Length: 200 bytes (1600 bits)
    Capture Length: 200 bytes (1600 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp:http:json]
    [Coloring Rule Name: HTTP]
    [Coloring Rule String: http || tcp.port == 80 || http2]
> Ethernet II, Src: VMware_53:e6:49 (00:0c:29:53:e6:49), Dst: VMware_bc:bc:2f (00:0c:29:bc:bc:2f)
> Internet Protocol Version 4, Src: 139.6.19.30, Dst: 139.6.19.29
> Transmission Control Protocol, Src Port: 10000, Dst Port: 12228, Seq: 1, Ack: 130, Len: 134
> Hypertext Transfer Protocol
v JavaScript Object Notation: application/json
  v Array
      [Path with value: /[]:21003]
      [Member with value: []:21003]
      Number value: 21003

0000  00 0c 29 bc bc 2f 00 0c  29 53 e6 49 08 00 45 00   ··)··/·· )S·I··E·
0010  00 ba c3 90 40 00 3f 06  3b 66 8b 06 13 1e 8b 06   ····@·?· ;f······
0020  13 1d 27 10 2f c4 38 b2  c7 c5 36 f9 d6 21 80 18   ··'·/·8· ··6··!··
0030  01 f6 01 1e 00 00 01 01  08 0a ae f0 1e 97 7c 5f   ········ ······|_
0040  7f 0c 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   ··HTTP/1 .1 200 O
0050  4b 0d 0a 43 6f 6e 74 65  6e 74 2d 54 79 70 65 3a   K··Conte nt-Type:
0060  20 61 70 70 6c 69 63 61  74 69 6f 6e 2f 6a 73 6f    applica tion/jso
0070  6e 0d 0a 44 61 74 65 3a  20 54 75 65 2c 20 31 36   n··Date:  Tue, 16
0080  20 4d 61 79 20 32 30 32  33 20 31 34 3a 32 31 3a    May 202 3 14:21:
0090  31 36 20 47 4d 54 0d 0a  43 6f 6e 74 65 6e 74 2d   16 GMT·· Content-
00a0  4c 65 6e 67 74 68 3a 20  38 0d 0a 43 6f 6e 6e 65   Length:  8··Conne
00b0  63 74 69 6f 6e 3a 20 63  6c 6f 73 65 0d 0a 0d 0a   ction: c lose····
00c0  5b 32 31 30 30 33 5d 0a                            [21003]·
```

Figure 11: Policy Packet (HTTP/JSON Protocol)

This packet was transferred through the HTTP/JSON protocol. Application/json file presenting the data, and the policy name is 21003, which we have seen before in the RIC control panel. Here in the JavaScript Object Notation section, there is an option with the name "array. This array contains a path and a member with the value 21003.

4.4 Follow-up opportunities

Currently, the system is not prepared to connect with the 5G core network as it solely encompasses RIC components. However, the development of other components is underway, and once completed, the system will be ready for integration with the 5G core network. The O-RAN Alliance is actively involved in the development of these additional components, providing open-source implementation resources through their published releases. As part of future work, the focus will be on deploying these forthcoming components and establishing the necessary connections between them and the 5G core network. This ongoing effort will pave the way for a comprehensive and fully functional system that leverages the capabilities of the 5G core network.

4.5 Challenges

Infrastructure Requirements: Setting up Non-RT RIC and Near-RT RIC involves configuring complex infrastructure. This includes deploying the necessary hardware, networking components, and virtualization platforms and hypervisors. Ensuring that the infrastructure meets the specific requirements of these components can be a time-consuming and intricate process.

Software Dependencies: Non-RT RIC and Near-RT RIC rely on various software dependencies, such as operating systems, container runtimes, and orchestration frameworks. Ensuring compatibility between these dependencies and the target deployment environment can be challenging, requiring careful selection and configuration of software versions.

Overall,installing Non-RT RIC and Near-RT RIC can be challenging because it involves dealing with technical complexities, infrastructure considerations, software dependencies, and optimization requirements. It requires expertise and careful attention to detail to ensure a successful and functional deployment.

5 Conclusion

This research project explores the initiatives of the O-RAN Alliance group, which focuses on software-based solutions for open and programmable radio access networks. The O-RAN Alliance group aims to establish a reference design that offers operators and service providers a unified approach to deploying disaggregated and software-defined 5G RANs. Additionally, the collaboration between O-RAN and the Linux Foundation has resulted in the development of initial software releases and documentation, serving as a prototype platform for further RAN software advancements.

The objective of this project was to deploy developed components of O-RAN in a specific radio access network intelligent controller part over a virtualization infrastructure to investigate the capabilities of those components as well as test their connectivity. Additionally, the study aimed to evaluate the crucial role of Kubernetes in orchestrating containerized applications. To achieve these objectives, the approach taken in this research project primarily revolved around practical aspects, demonstrating the deployment of specific O-RAN software components in a customized virtual environment.

Despite the initial challenges, the project managed to deploy non-RT RIC, near-RT RIC, and xApp components successfully. Basic performance evaluations were conducted to assess their performance. The connectivity between the two RICs was established, allowing effective communication. Furthermore, the creation of policies was accomplished without any issues, enhancing the overall functionality of the deployed components. These achievements demonstrate significant progress in implementing the desired functionalities and capabilities of the system. The positive outcomes from the performance evaluations provide a solid foundation for further optimization and fine-tuning of the components, ensuring optimal performance in real-world scenarios.

In summary, the procedural steps taken in this research project can be seen as a starting point for further exploration into the feasibility of deploying O-RAN software. These steps provide a basic framework for assessing the deployment process, which can be adapted and expanded upon as the software components continue to evolve.

6 Abbreviations

RAN = Radio Access Network.

O-RAN = Open Radio Access Network.

RIC = RAN Intelligent Controller.

SMO = Service Management and Orchestration.

NSF = Network Slicing Function.

MANO= Management and Orchestration.

NFVI = Network Functions Virtualization Infrastructure.

VNFs = Virtual Network Functions.

PNFs = Physical Network Functions.

NMS = Network Management Systems.

O-CU = O-RAN Central Unit.

O-CU-CP = O-RAN Central Unit in control plane.

O-CU-UP = O-RAN Central Unit in user plane.

O-DU = O-RAN Distributed Unit.

O-RU = O-RAN Radio Unit.

RRC = Radio Resource Control.

SDAP = Service Data Adaptation Protocol.

PDCP = Packet Data Convergence Protocol.

ONAP = Open Network Automation Platform.

CNI = Container Network Interface

RF = Radio Frequency.

MAC = Media Access Control Address.

RLC = Radio link control.

FCAPS = fault, configuration, accounting, performance, and security.

Bibliography

1 O-RAN Alliance. "O-RAN Alliance". available online: `https://docs.o-ran-sc.org/en/latest/architecture/architecture.html`.

2 O-RAN Alliance. online: `https://orandownloadsweb.azurewebsites.net/specifications`.

3 O-RAN Software Community. available online: `https://wiki.o-ran-sc.org/display/RICNR/Release+G+-+Run+in+Kubernetes`.

4 O-RAN Alliance. available online: `https://www.o-ran.org/specifications`.

5 o-ran-sc.org. available online: `https://docs.o-ran-sc.org/projects/o-ran-sc-it-dep/en/latest/`.

6 O-RAN Software Community. available online: `https://wiki.o-ran-sc.org/display/REL/F+Release`.

7 rimedolabs.com. available online: `https://rimedolabs.com/blog/o-ran-near-real-time-ric/`.

8 Oliver Radwell available online: `https://blog.radwell.codes/2022/07/`.

9 linuxconfig.org. available online: `https://linuxconfig.org/`.

10 github.com. available online: `https://github.com/o-ran-sc/ric-plt-a1/blob/master/docs/user-guide-api.rst`.