H2020 5Growth Project
Grant No. 856709

D2.4: Final implementation of 5G End-to-End Service Platform

Abstract

The goal of this deliverable is to document the final release of 5Growth platform software developed within the project to support the innovations implemented in the end-to-end service platform. This document describes the new functionalities introduced in the second release of the 5Growth platform and provides the software documentation of its components. The code provided along with this document implements the innovations described in D2.3, building on top of the initial release delivered in D2.2.

Document properties

Document number	D2.4
Document title	Final implementation of 5G End-to-End Service Platform
Document responsible	Juan Brenes (NXW)
Document editor	Juan Brenes (NXW)
Editorial team	Giada Landi (NXW), Claudio Casetti (POLITO), Carla Fabiana Chiasserini (POLITO), Marco Malinverno (POLITO), Corrado Puligheddu (POLITO), Agustín Caparrós (TELCA), Teresa Giner (TELCA), Manuel A. Jiménez (TELCA), Ramón Pérez (TELCA), Javier Sacido (TELCA), Aitor Zabala (TELCA), Oleksii Kolodiazhnyi (MIRANTIS), Konstantin Tomakh (MIRANTIS), Denys Kucherenko (MIRANTIS), Andres Garcia Saavedra (NEC), Xi Li (NEC), Luca Vettori (CTTC), Ricardo Martínez (CTTC), Engin Zeydan (CTTC), Josep Mangues-Bafalluy (CTTC), Jorge Baranda (CTTC), Andrea Sgambelluri (SSSA),Molka Gharbaoui (SSSA), Lorenzo De Marinis (SSSA), Luca Valcarenghi (SSSA), Fabio Ubaldi (TEI), Andrea Boddi (TEI), Giulio Bottari (TEI), Paola Iovanna (TEI), Kiril Antevski (UC3M), Antonio de la Oliva (UC3M), Ivan Vidal (UC3M), Carlos Guimaraes (UC3M)
Target dissemination level	PU
Status of the document	Final
Version	1.0
Delivery date	May 31, 2021
Actual delivery date	May 31, 2021

Production properties

Reviewers	Carlos Guimarães (UC3M), Oleksii Kolodiazhnyi (MIRANTIS), Andres Garcia Saavedra (NEC), Xi Li (NEC)

Disclaimer

Contents

List of Figures

List of Tables

List of Acronyms

5Gr-RL – 5Growth Resource Layer

5Gr-SO – 5Growth Service Orchestrator

5Gr-VS – 5Growth Vertical Slicer

5GT-MTP – 5G-TRANSFORMER Mobile and computing Transport Platform

5GT-SO – 5G-TRANSFORMER Service Orchestration

5GT-VS – 5G-TRANSFORMER Vertical Slicer

AD – Anomaly Detection

AI – Artificial Intelligence

BSS – Business Support System

CMD – Cyber Mimic Defense

CP – Control Plane

CSMF – Communication Service Management Function

CsA – Class of Service Algorithm)

CSP – Communications Service Provider

CU – Cloud/Central Unit

DB – Database

DHR – Dynamic Heterogeneous Redundancy

DLT – Distributed Leger Technology

DSL – Domain-Specific Language

DSS – Decision Support System

DU – Distributed Unit

E2E – End-to-End

EBI – EastBound Interface

EC – Edge Computing

EE – Execution Entity

eMBB – Enhanced Mobile Broadband

FG – Forwarding Graph

gNB – next Generation Node B

GUI – Graphical User Interface

InA – Infrastructure Abstraction

IPO – Input Process - Output model

KPI – Key Performance Indicator

LC – Lifecycle

LCM – LC Management

LSTM – Long term Short Term Memory

MEC – Multi-access Edge Computing

MF – Management Function

ML – Machine Learning

mMTC – Massive Machine Type Communications

MILP – Mixed Integer Linear Program

MP – Monitoring Platform

MQ – Message Queue

MTD – Moving Target Defense

NBI – Northbound Interface

NFV – Network Function Virtualization

NFVI – NFV Infrastructure

NFV-NS – NFV Network Service

NFVO – NFV Orchestrator

NS – Network Slice

NSD – Network Service Descriptor

NSI – Network Slice Instance

NSMF – Network Slice Management Function

NSO – Network Service Orchestrator

NSSMF – Network Slice Subnet Management Function

NST – Network Slice Template

OSS – Operations Support System

PA – Placement Algorithm

PoP – Point of Presence

QA – Quality Assurance

QoS – Quality of Service

RA – Resource Allocation

RAN – Radio Access Network

RIC – Radio Intelligent Controller

RO – Resource Orchestrator

(R)RU – (Remote) Radio Unit

RT – Real Time

RVM – Remove Virtual Machine

SBI – SouthBound Interface

SDN – Software Defined Networking

SFC – Service Function Chaining

SLA – Service Level Agreement

SLPOC – Single Logical Point of Contact

SO – Service Orchestrator

TSP – 5G-TRANSFORMER Service Provider

TSDB – Time Series Database

UC – Use Case

UE –User Equipment

UP – User Plane

URLLC – Ultra Reliable Low Latency Communications

VA – Vertical Application

VIM – Virtual Infrastructure Manager

VM – Virtual Machine

VNF – Virtual Network Function

VNFD – VNF Descriptor

VNFM – VNF Manager

VoMS – Vertical-oriented Monitoring System

vRAN – Virtual Radio Access Network

VSB – Vertical Service Blueprint

VSD – Vertical Service Descriptor

VSI	Vertical Slice Instance
WBI	Westbound Interface
WAN	Wide Area Network
WIM	WAN Infrastructure Manager
ZDM	Zero-Defect Manufacturing
ZSM	Zero-touch Service Management

Executive Summary and Key Contributions

This deliverable provides the final reference implementation of the 5Growth end-to-end service platform and the release notes of its software components, namely:

- Vertical Slicer
- Service Orchestrator
- Resource Layer
- Vertical-oriented Monitoring System
- AI/ML Platform
- Forecasting Functional Block

In addition, this document details the Continuous Integration/Continuous Development (CI/CD) pipelines established to automate the end-to-end service platform testing and deployment in order to facilitate the development and integration lifecycles. The repositories containing the software code and assets produced during the project are published as open source and reported in this document to support the installation of the platform in third party environments and the re-use of its components in future research activities. Furthermore, the developments described in this document resulted in the open source "Contribution to ONOS Repository" described in D5.4 [36] (see section 4.1.9.1 for further details).

The software described in this document was developed to support the innovations designed in WP2 and provides a baseline implementation for the additional developments focused on pilot-specific extensions, as addressed in WP3 and WP4 activities. The content of this deliverable builds upon the release plan established in D2.2 [8] and describes the software functionalities developed for Release 2 to support the innovations in D2.3 [1]. This Release 2 is the final release of the 5Growth end-to-end in terms of innovations. The feature plan established in D2.2 [8] was updated to focus on the new functionalities required to demonstrate the innovations in the context of the 5Growth pilots. Moreover, this document includes the description of two new components that were added in the 5Growth final release, namely the AIML platform and the Forecasting module, detailing their internal components and their interfaces.

The **key contributions** presented in this document are the final reference implementation of all the modules composing the 5Growth end-to-end service platform. The developed software is available in software repositories on GitHub [2], [10], [12], [13], [14], [15], [16], [17], [18], [20], [22], [23] and [35].

1. 5Growth high-level architecture

The global architecture of 5Growth platform is described in D2.3 [1]. For the sake of completeness, we next introduce its key components, shown in Figure 1. In the remaining of the document, we present the relevant software components made publicly available.

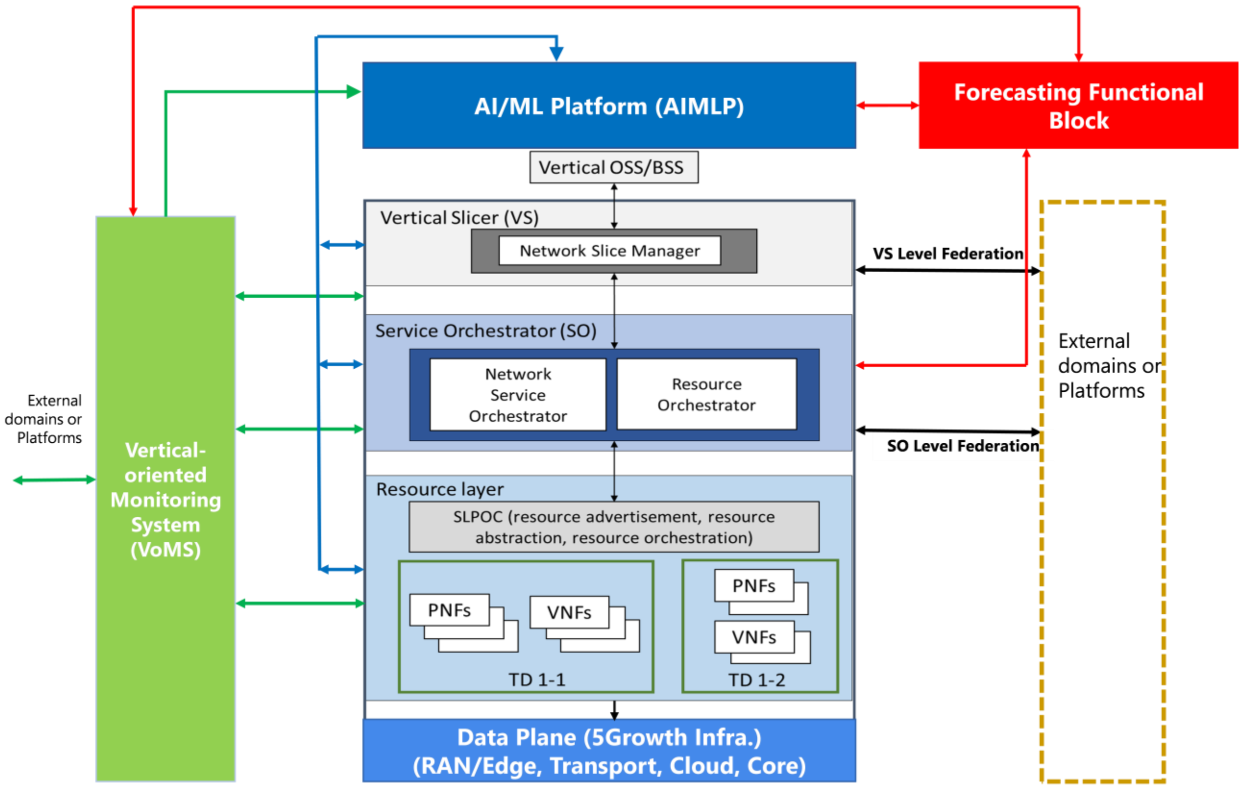

AI/ML Platform (AIMLP)
Forecasting Functional Block
Vertical OSS/BSS
Vertical Slicer (VS)
Network Slice Manager
VS Level Federation
Service Orchestrator (SO)
Network Service Orchestrator
Resource Orchestrator
External domains or Platforms
SO Level Federation
External domains or Platforms
Vertical-oriented Monitoring System (VoMS)
Resource layer
SLPOC (resource advertisement, resource abstraction, resource orchestration)
PNFs
VNFs
TD 1-1
PNFs
VNFs
TD 1-2
Data Plane (5Growth Infra.) (RAN/Edge, Transport, Cloud, Core)

FIGURE 1: 5GROWTH HIGH-LEVEL ARCHITECTURE

- The Vertical Slicer (VS or 5Gr-VS, as defined in the rest of the document to differentiate from the old version of the Vertical Slicer developed in the context of the 5G-TRANSFORMER project) is the front-end point for verticals demanding the provisioning and management of vertical services via a simplified northbound interface with the vertical OSS/BSS. The 5Gr-VS implements the functionalities for the management of vertical services and their mapping to the end-to-end 5G network slices.
- The Service Orchestrator (SO or 5Gr-SO) handles both the network service and resource orchestration capabilities of: (i) enabling end-to-end orchestration of NFV-Network Services (NFV-NS) by mapping them across either a single or multiple administrative domains based on service requirements and availability of the services/resources offered by each of the domains; and (ii) managing their life-cycles (i.e., on-boarding, instantiation, update, scaling, termination, etc.).
- The Resource Layer (RL or 5Gr-RL) is responsible for managing the local infrastructure interacting with the underlying controllers (i.e., VIMs and WIMs). To this end, the 5Gr-RL manages all the compute, storage, and networking (physical and virtual) resources where the elements forming network slices and end-to-end services are accommodated.

- The Vertical-oriented Monitoring System (VoMS or 5Gr-VoMs) constitutes the monitoring platform aiming to monitor heterogeneous set of services and technological domains. To do that, it offers required functionalities such as log aggregation, a scalable data distribution system and dynamic probe reconfiguration.
- The AI/ML Platform (5Gr-AIMLP) is the entity that manages (i.e., deploys, configures and trains) AI models relying on the data gathered from the VoMS. The 5Gr-AIMLP exposes a catalogue of AI models, susceptible to be tuned/chained to come up with more complex models, that are consumed by other components of the 5Growth stack for taking decisions at different layers e.g., about service arbitration or service scaling.
- The Forecasting Functional Block (5Gr-FFB) aims at generating forecast values of parameters monitored by the 5Gr-VoMS. These forecast values can be utilized by decision modules of the platform to take a proactive approach in reacting to changing status. The 5Gr-FFB interacts with the 5Gr-AIMLP (to request trained models to be utilized in forecasting tasks), with the 5Gr-VoMS (to retrieve parameters to be forecast as well as to provide forecast data to be used as input for other decisional elements like the 5Gr-SO), and with other main functional elements of the 5Growth architecture, such as the 5Gr-SO.

2. 5Gr-VS implementation

The high-level software architecture of the 5Gr-VS Release 2 is shown in Figure 2, highlighting in yellow the components, which were introduced or updated to support Release 2 functionalities enabling the innovations described in D2.3 [1]. As represented in the figure, the 5Gr-VS software architecture is composed of two main building blocks: The Vertical Service Management Functionality (VSMF) and the Network Slice Management Functionality (NSMF). On the one hand, the VSMF embraces the modules related to the vertical service design, customization, and lifecycle management. On the other hand, the NSMF contains the catalogue of network slice templates, and the modules handling the lifecycle management of network slices.

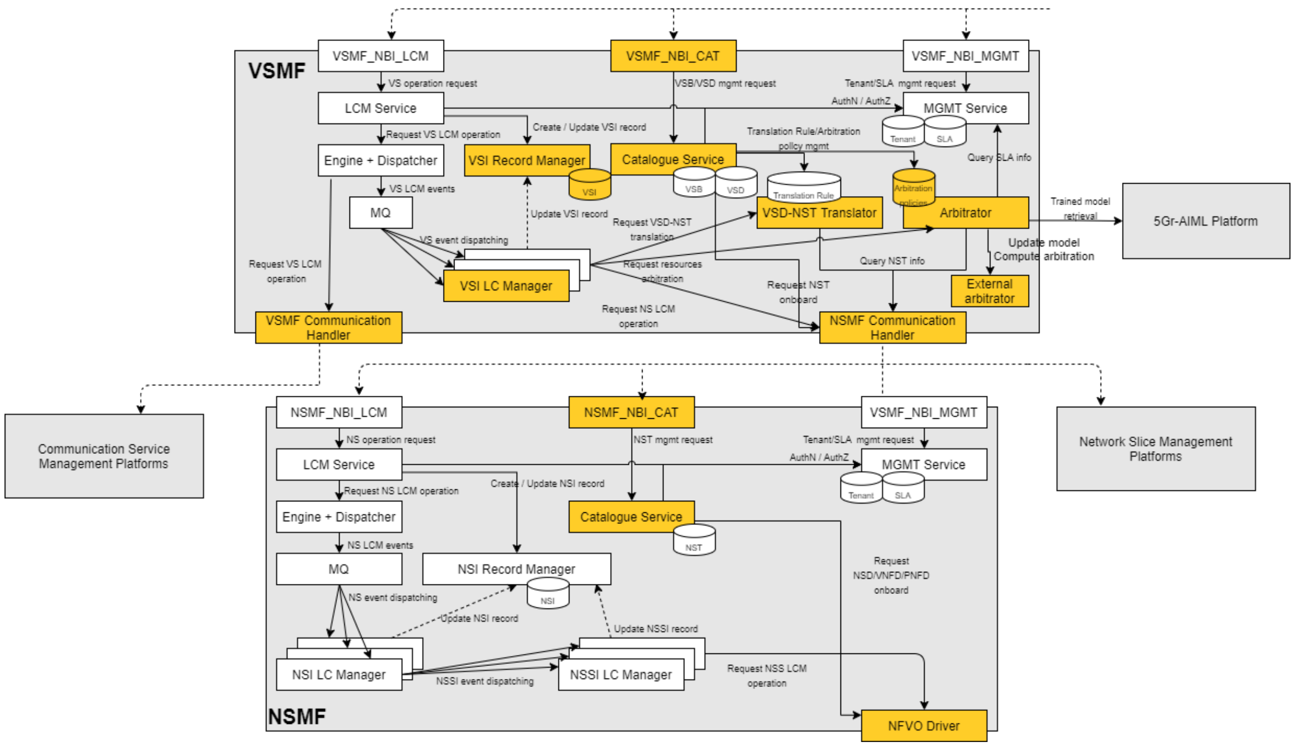


FIGURE 2: 5GR-VS HIGH-LEVEL SOFTWARE ARCHITECTURE

These two main building blocks follow the same design principle, offering different REST-based APIs (similar to a service-oriented architecture) for:

1. Group, tenant, and SLAs management (VSMF_NBI_MGMT);
2. Descriptors, associated policies, and rule catalogue management (VSMF_NBI_CAT and NSMF_NBI_CAT);
3. Lifecycle management (VSMF_NBI_LCM and NSMF_NBI_LCM).

All these REST-based interfaces include authentication and authorization mechanisms for each type of request. All modules are implemented using Java based Apache Maven projects, which are then bundled together as Spring Boot applications, using the Spring framework.

When a vertical service lifecycle management request is received at the VSMF_NBI_LCM (i.e., vertical service instantiation, termination, status request), the request is initially validated and passed on to the VSMF LCM Service. In turn, the VSMF LCM Service forwards the request to the appropriate Vertical Service Instance lifecycle manager (VSI LCM), or to the VSMF Communication Handler in case of vertical services crossing multiple domains or targeting external domains. The VSMF Communication Handler implements a driver-based interface to interact with external communication service provider platforms in order to delegate the deployment and management of communication services to other domains. In 5Growth, this functionality is exploited to deploy vertical services in platforms managed by ICT-19 projects, like 5G EVE.

Figure 3 shows the high-level instantiation workflow performed within the VSMF and NSMF functional blocks of the 5Gr-VS. The VSI LCM translates the vertical service request into network slice operations, which are passed on to the NSMF Communication Handler. In the case of a vertical service instantiation request, this translation is performed using the translation and arbitration modules. The translator module computes the network slices and their dimension in terms of resources given the specific vertical service requirements, and the arbitration module determines priority-based actions and possible sharing of network slice instances to fulfil the tenant and the vertical service SLAs. Analog to the VSMF Communication Handler, the NSMF Communication Handler uses a driver-based approach that allows using the NSFM functionality provided by the 5Gr-VS or that of external Network Slice Management platforms, provided by other domains, transparently.

The NSMF processes the network slice lifecycle management operations in a similar manner. The request is received and initially validated by the NSMF_NBI_LCM and passed on to the NSMF LCM Service, which forwards the request to the specific LCM of the network slice instance. There, the network slice instance is translated into the corresponding set of NFV network services (NFV-NSs), which are then requested to the NFVO Driver, in charge of all the interactions with the 5Gr-SO.

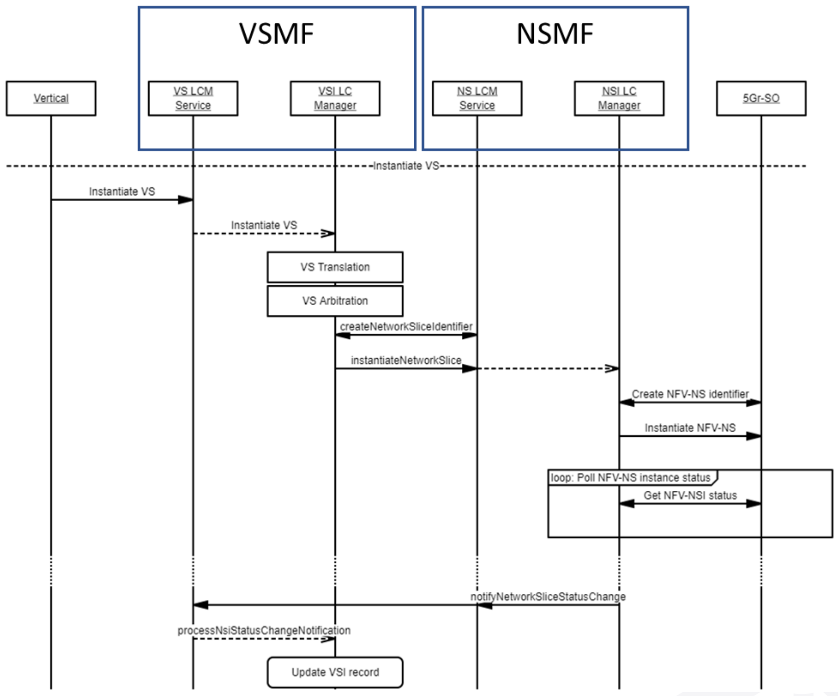


FIGURE 3: 5GR-VS VSMF AND NSMF VS INSTANTIATION WORKFLOW

The 5Gr-VS relies on RabbitMQ as a message bus for internal asynchronous communication and PostgreSQL for the backend database used to store the information available in the records of vertical service and network slice instances. The following list provides a description of the different software components of the 5Gr-VS. These components are Java-based maven projects leveraging the Spring framework. All the components are licensed under the Apache 2.0 license:

- **VSMF Northbound Interface (NBI) Catalogue (VSMF_NBI_CAT) & VSMF Catalogue service:** The VSMF_NBI_CAT interface models the methods to onboard and manage the vertical service blueprints and descriptors and the policies associated with the service lifecycle management. The VSMF Catalogue service provides the implementation of this interface, with the logic for vertical service blueprints and descriptors management. The source code of these modules is available on the *VS_BLUEPRINTS_CATALOGUE* project in the slicer-catalogue repository [14].
- **VSMF NBI Management (VSMF_NBI_MGMT) & Management Service (MGMT_Service):** The VSMF_NBI_MGMT hosts the interfaces for the management of the tenants and their associated SLAs, which are managed through the MGMT_Service. The source code of these modules is available on the *SEBASTIAN_MGMT_CATALOGUE* project at the slicer-identity-mgmt repository [15].
- **VSD-NST Translator service:** The Translator service is responsible for determining the network slices, network slice subnets and their dimensions given the specific customization of the vertical service (established by means of the VSD and the instantiation request). The source code of this module is available on the *TRANSLATOR_SERVICE* project in the slicer-catalogue repository [14].
- **Arbitrator service:** The arbitration software module implements the logic establishing the network slice instances to share when deploying a new vertical service, and to determine possible service priority-based actions (e.g. scaling or termination of existing low-priority services) in order to accommodate resources to an incoming service. The source code of this module is part of the *ARBITRATOR_SERVICE* project in the 5Gr-VS repository [16].
- **VSMF LCM Service:** The VSMF LCM Service is the module responsible for the overall lifecycle management of the vertical service instances, creating and holding the registry of the managers for each specific instance and forwarding the network slice and vertical service status notifications and commands to the corresponding managers. The source code for this module is available in the *VSMF_SERVICE* in the 5Gr-VS repository [16].
- **Vertical Service Instance (VSI) Lifecycle (LC) manager:** The VSI LC manager is the module responsible for processing the lifecycle management actions for one particular VSI, namely instantiate, terminate and scale, consequently managing the lifecycle of the network slice instances associated with the vertical service. The source code for this module is available in the *VSMF_SERVICE* in the 5Gr-VS repository [16].
- **VSI Record Manager:** The VSI Record manager holds the information and status about the different vertical service instances and the associated network slice instances. The source code for this module is part of the *RECORD_SERVICE* project in the 5Gr-VS repository [16].

- **NSMF Northbound Interface Catalogue (NSMF_NBI_CAT) & NSMF Catalogue Service:** These two modules are responsible for the management of network slice templates and their associated descriptors. The source code of these modules is available on the *NS_TEMPLATES_CATALOGUE* in the slicer-catalogue repository [14].
- **Network Slice Instance (NSI) Record Manager:** The NSI Record manager holds the information and status about the different network slice and slice subnet instances and the associated NFV-NS instances. The source code for this module is part of the *RECORD_SERVICE* project in the 5Gr-VS repository [16].
- **NFVO Driver:** The NFVO Driver hosts the implementation of the 5Gr-SO driver, which is the mediator between the 5Gr-SO and the 5Gr-VS for the NFV-NS lifecycle management and the VNFD and NSD onboarding. The source code for this module is part of the *NFVO_CATALOGUE_DRIVERS* project in the 5Gr-VS repository [16].

2.1. Final release features

Table 1 describes the different releases of the 5Gr-VS in terms of functionalities, mapping the functionalities to the innovations described in D2.3 [1].

TABLE 1: 5GR-VS RELEASES

Release 1	Release 2	WP2 Innovations support (See D2.3 [1])
Support of RAN modelling in network slices: • NST catalogue • Update of VSB information model • Update of Translator, VS LCM, and NS LCM • Update of VS-SO interface (client side)	Support of Physical Network Function Descriptor (PNFD) on-boarding: • Updated VSB onboarding GUI wizard and REST interface to allow PNFD onboarding • Updated NST onboarding GUI wizard to enable PNFD onboarding • Enabled PNFD onboarding functionality on 5Gr-SO driver	Support to I1
	Support of AI/ML based arbitration: • Support of external arbitrators for arbitration logic decoupling • 5Gr-VS – 5Gr-AIML Platform interaction for retrieval of trained model • Policy-based arbitration model management • Enabled mechanisms for network slice sharing among vertical services	Support to I4 and I5
	Support of VS subservice decomposition: • VSB/VSD information model and workflow update • Multi-domain vertical service deployment	Support to I6
	Support for Vertical Service translation rule management:	Support for I4 and I5

	• REST-based interface for retrieval, onboarding, and modification of translation rules	
	Support for network slice level multi-domain scenarios: • Updated translator module for network slice level multi-domain scenarios • Added NSMF Communication Handler for driver-based network slice lifecycle management per domain	Support for I6
	New 5Gr-VS deployment: • Containerized deployment of the 5Gr-VS components • New 5Gr-VS GUI	Support to 5Growth CI/CD

The following sections detail the modifications made to the different modules. We refer to D2.3 [1] for further details about the innovations and the overall platform approach.

2.1.1. VSMF northbound interface (NBI) Catalogue (VSMF_NBI_CAT)

This interface was updated in order to support two of the functionalities introduced in Release 2:

- *Support for PNFD onboarding*: a new parameter was added to the VSB onboarding request, allowing the admin user the possibility of providing the PNFDs for the PNFs available on the infrastructure.
- *Policy-based Arbitration Model management*: a new API was added to this module to allow the management (i.e., onboarding, update, and deletion) of the arbitration policies, which are used to determine the trained model offered by the 5Gr-AIMLP to be used by the External Arbitrator.
- Translation rule management: A new REST interface was developed to allow the onboarding, modification, deletion and query of the translation rules between Vertical Services and Network Slices.

2.1.2. VSMF Catalogue service

The VSB and VSD information models and the onboarding workflows were modified in order to support the vertical service decomposition into vertical (sub)-services. In particular, the information model of the VSB was modified to allow an “Atomic Component” of the VSB to be of the type “Service”, and to link the associated blueprint and the candidate domain to which this vertical (sub)-service is to be requested. The VSD onboarding workflow was modified to automatically create the vertical (sub)-service descriptors based on the one defined by the end-to-end VSD, as shown in Figure 4.

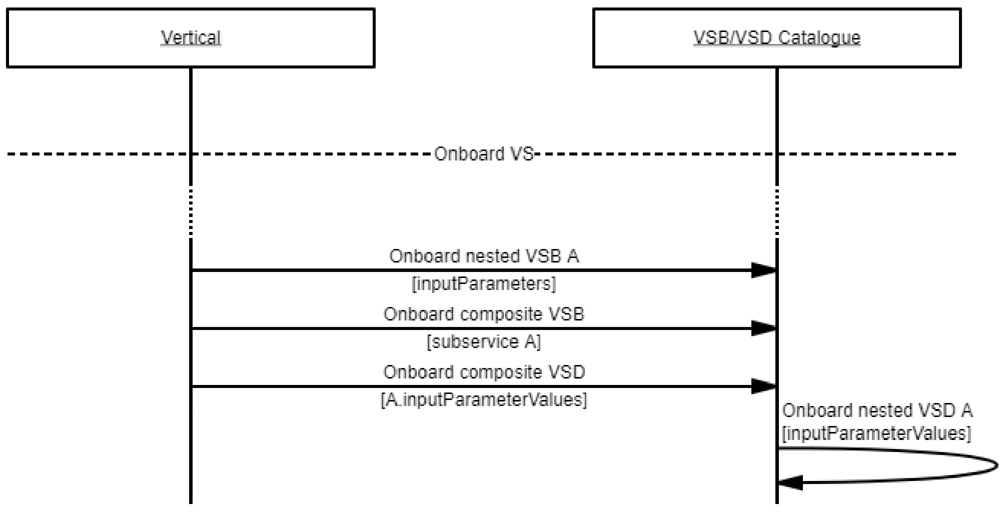


FIGURE 4: 5GR-VS COMPOSITE VSB/VSD ONBOARDING WORKFLOW

The catalogue logic was also enhanced to support the PNFD onboarding: the PNFDs descriptors in the VSB onboarding request are forwarded to the NSMF as part of the NST onboarding requests.

2.1.3. VSD-NST Translator service

This module was updated to support the determination of the domain to which a network slice subnet is to be requested. This outcome of the translation process is used by the manager of the specific vertical service instance to send the request for a network slice subnet to a certain domain through the "NSMF Communication Handler".

2.1.4. NSMF Communication Handler / VSMF Communication Handler

The NSMF Communication Handler and VSMF Communication Handler modules were introduced in this last release to enable driver-based interactions with external NSMF / CSMF platforms in multi-domain scenarios, which enables, e.g., interactions with ICT-17 platforms. In this way, the VSMF LCM service can request the provisioning of network slice subnets and vertical (sub)-services to external domains using a unified interface that hides the domain-specific logic and information models, while delegating to domain-specific drivers the message translation and the final interaction with the external platforms. In particular, the implementation includes a VSMF Communication Handler driver to interact with ICT-17's 5G EVE Portal and an NSMF Communication Handler driver to interact with ICT-17's 5G VINNI network slice management platform, as fully documented in D3.4 [32]. The source code for this module is part of the *VSMF_SERVICE* project in the 5Gr-VS repository [16].

2.1.5. Vertical Service Instance (VSI) Lifecycle (LC) manager

This module was updated to process the enhanced translator response (containing per domain network slice subnets) and to use the VSMF/NSMF Communication Handlers. In particular, this means to include the vertical (sub)-service management as part of the end-to-end service management. Moreover, the lifecycle management workflow was updated to allow vertical services to share network slices and trigger network slice scaling procedures as established by the enhanced arbitration module.

2.1.6. VSI Record Manager

In this case, the module and the catalogue associated were updated in order to add the information about the vertical (sub)service instances and the network slice subnet instances associated with one VSI.

2.1.7. Arbitrator

The following changes were introduced in this module for Release 2:

- *Arbitration Policies:* Added the catalogue of policies associated to the AI/ML model to be used for vertical service arbitration decisions. The source code for this module is available on the *ARBITRATOR_CATALOGUE* in the slicer-catalogue repository [14].
- *External arbitrator support:* A new type of arbitrator was developed to enable the usage of decoupled arbitration logic, by means of REST APIs. This External Arbitrator enabled the usage of trained models to compute arbitration decisions. Trained models are passed on to this External Arbitrator together based on the active policies. The external arbitrator computes a solution using the trained models.
- *5Gr-VS – 5Gr-AIML Platform integration:* The arbitrator module acts as a client of the 5Gr-AIMLP for the retrieval of trained models.
- External Arbitration Algorithm, which makes use of the ML trained model to determine whether already deployed sub-slices can be reused. We refer to D2.3 [1] Sec. 3.2.1.1.4 for further details regarding the sub-slice sharing algorithm implemented within the External Arbitrator.

Figure 5 depicts the workflow needed for the implementation of the sub-slice sharing mechanism, which implies the cooperation between 5Gr-VS and 5Gr-AIMLP (see section 5). We can identify nine main steps. The first three steps are necessary for creating a trained model able to provide the arbitration algorithm with the best latency class configuration according to the current running services.

1. **Dataset creation (Step 1)**: thanks to the interaction with the VSD Catalogue and the 5Gr-VoMS (see Section 7), which provides respectively the current number of service instances and the number of vCPUs used, it is possible to build a training dataset.
2. **ML model fitting (Step 2)**: the dataset obtained at the previous step is then used to fit the ML model.
3. **ML model validation (Step 3)**: by using the preferred method (in our case the *k-fold cross*-validation) the ML model is validated.

At this point, the trained model is available at the 5Gr-AIMLP and, after the proper request, it can be forwarded to the 5Gr-VS Arbitrator (Step 4).

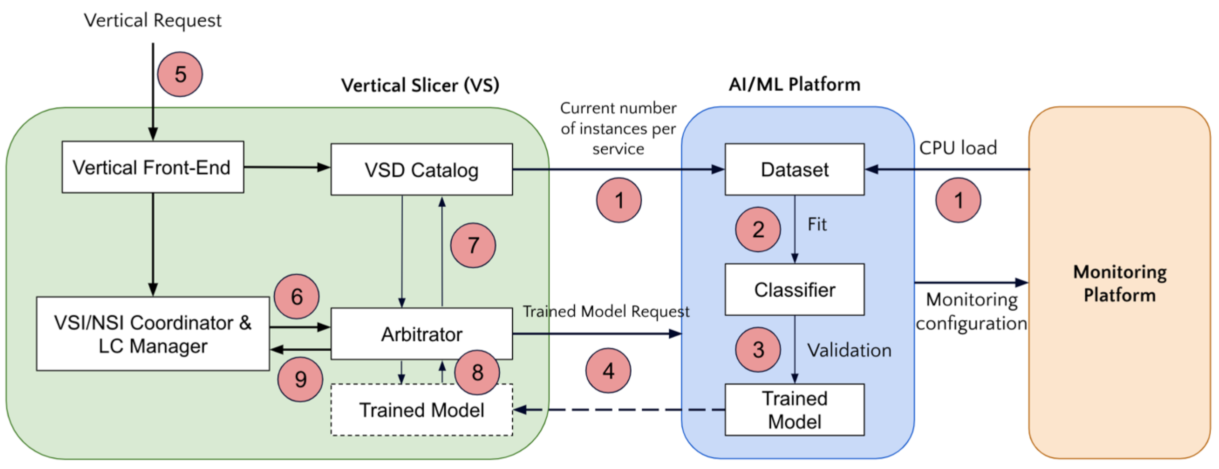


FIGURE 5: NETWORK SUB-SLICE REUSE WORKFLOW

When a vertical wants to deploy a new service, the instantiation request is received by the 5Gr-VS, in particular by the *Vertical Front-End* (Step 5). The request is thus sent to the *VSI/NSI Coordinator & LC Manager*, which processes and forwards it toward the Arbitrator, in charge of managing the service and VNFs instantiation (Step 6). Therefore, the Arbitrator interacts with the VSD Catalogue (Step 7) in order to get the current number of running instances. This value is then sent to the trained model, which uses it as input and provides the best latency class configuration to the Arbitrator (Step 8). Finally, the Arbitrator runs the Algorithm to determine whether it is possible to reuse one or more sub-slices for the new service instantiation and sends back the reply to the VSI/NSI Coordinator (Step 9). Initial results related to the ML-driven slice sharing algorithm can be found in D2.3 [1] Sec. 4.9.1.4.

2.1.8. NSMF northbound interface (NBI) Catalogue (NSMF_NBI_CAT) & NSMF Catalogue Service

These two modules are responsible for the management of network slice templates, and their associated descriptors. In Release 2, these modules have been extended to support the onboarding of PNFDs. These latter descriptors are passed on to the NFVO Driver, which triggers the onboarding procedure on the 5Gr-SO. In particular, this feature is needed to fully support the RAN slicing mechanism described in D2.3 [1].

2.1.9. NFVO Driver

This module, as described previously, contains the driver mediating between the 5Gr-VS and the 5Gr-SO. For this Release 2, the driver was extended to support PNFD onboarding as described in section 2.1.8.

3. 5Gr-SO implementation

The 5Gr-Service Orchestrator (5Gr-SO) is responsible for the end-to-end (E2E) orchestration of NFV-Network Services (NFV-NS) and management of their lifecycles across single or multiple administrative domains by interacting with the 5Gr-RL or other peering domains. This module, based on the SO module provided by the 5G-TRANSFORMER project, has been evolved through the different software Releases of 5Growth to include corresponding features to support the work developed in some of the innovations presented in D2.3 [1]. Examples of extensions included in Release 2 of the 5Gr-SO module cover the interaction with the 5Gr-AIMLP and 5Gr-FFB platforms to support AIML-based scaling operations and the inclusion of scaling capabilities for multi-domain scenarios.

The software architecture of 5Gr-SO is depicted in Figure 6.

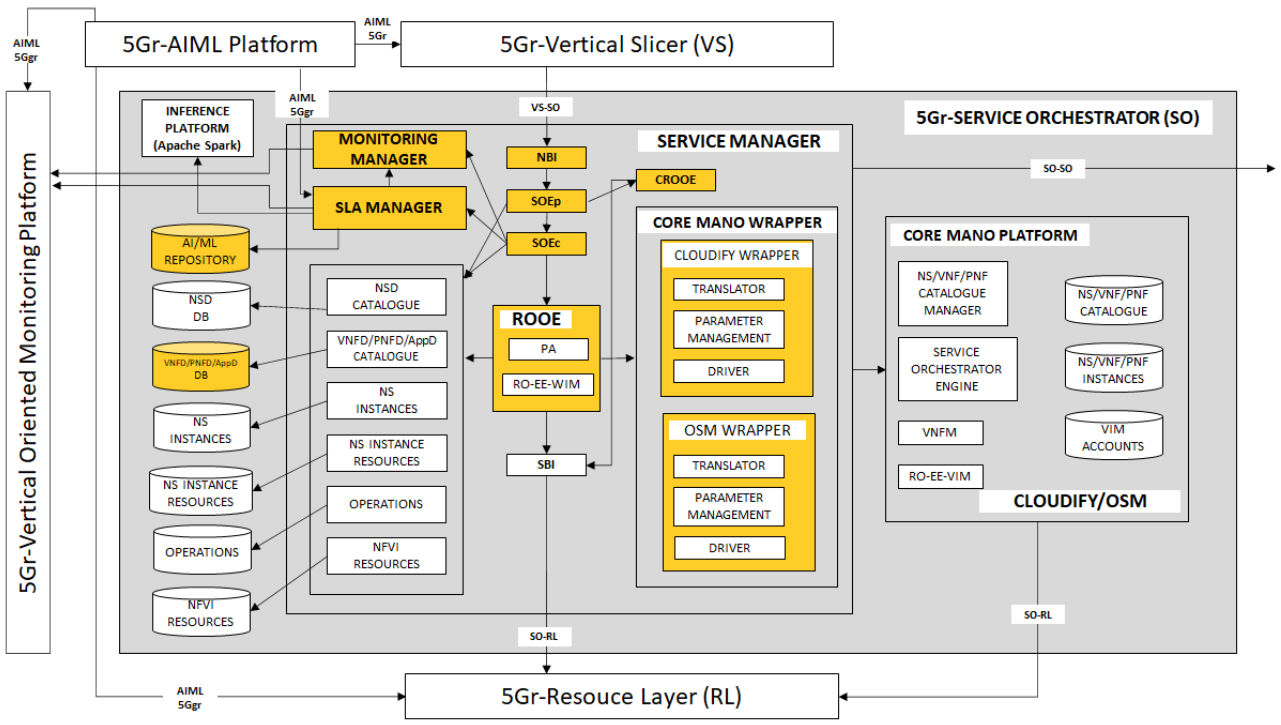


FIGURE 6: 5GR-SO PLATFORM ARCHITECTURE

The software code of the 5Gr-SO Reference Implementation is mostly developed in Python language (greater than version 3.5) and each block is implemented as a Python module. Similar to Release 1, all the software is open source. In particular, the components developed as part of the Service Manager (SM) have been released under Apache 2.0 license and they are available in the project git repository [20] as well as in the public GitHub [2].

Next, a short description of each module will be presented.

- ***Northbound interface (NBI):*** This module represents the interface for the northbound API between the 5Gr-VS and the 5Gr-SO. It is used by the 5Gr-VS to request the onboarding of all the different Descriptors, and to query and send operational commands regarding the lifecycle of the NFV-NS instances.

- ***Service Orchestration Engine (SOE):*** This module is located in the Service Manager of the 5Gr-SO and is responsible for the overall orchestration of the NFV-NS within the 5Gr-SO structure. It is composed of two sub-modules called SOE-parent (SOEp) and SOE-child (SOEc).
    - *SOE-parent (SOEp):* it oversees the orchestration for composite NFV-NSs and in the case of Network Service Federation.
    - *SOE-child (SOEc):* it is in charge of the orchestration of "regular" NFV-NSs, that is the case of a non-composite NFV-NS or a single nested NFV-NS included in the composite NFV-NS.
- ***Composite RO-OE (CROOE):*** This module is in charge of resource orchestration in the case of composite NFV-NSs (either federated or not). In the case of composite NFV-NSs deployed in the local domain, this block interacts with the 5Gr-RL in order to set-up and update the inter-nested connectivity (i.e., connections between VNFs belonging to different nested NFV-NSs).

  In the case of federation, the CROOE also sets up and updates inter-domain inter-nested connectivity. In this case, it interacts with the CROOE in the federated domain to request the necessary resource information to eventually set up the network connections of the inter-domain connectivity (local CROOE), and the network connections in the federated domain of the same connectivity (CROOE in the federated domain).

- ***Resource Orchestration Engine (ROOE):*** This module is in charge of handling the requests related to resource management in coordination with the 5Gr-RL in case of "regular" NFV-NSs during the instantiation, scaling, and termination phases of the NFV-NS. The ROOE is composed of two main sub-modules: the PA and the RO-EE-WIM and during its functional workflows, it also interacts with the Core MANO Platform block to accomplish the request.
    - *Placement Algorithm (PA):* its main function is to determine the placement of the different VNFs and virtual links of the NFV-NS, as well as the selection of the logical links required to interconnect them based on the requirements expressed in the NSD and the resource availability at the NFVI.
    - *Network Resource Orchestration Execution Engine (RO-EE-WIM):* its main function is the handling of the network-related resource allocation operations in case of deployment (for the NFV-NS) in multiple sites.
- ***Core MANO Platform:*** this block is responsible to orchestrate and manage the lifecycle of the NFV-NS. To do that, the 5Gr-SO relies on an external entity, called the core MANO Platform. As mentioned before, currently the 5Gr-SO supports Cloudify [4] and OSM [6] platforms.
- ***Core MANO Wrapper:*** it interconnects the Service Manager (SM) blocks with the Core MANO Platform. Currently, the 5Growth stack supports two Core MANO Platforms: Cloudify and OSM, and dedicated wrappers for these MANOs are contained in this module.
- ***Monitoring Manager:*** this module is responsible for the interaction with the "Config Manager" of the 5Growth Vertical Oriented Monitoring System (5Gr-VoMS) to manage the so-called Monitoring Jobs. Especially, this block configures the collection of the VNF

performance metrics and log files expressed in the NSD and the configuration of visualization dashboards.

- ***SLA Manager:*** this block is in charge of handling SLA compliance for a given NFV-NS. 5Gr-SO can perform SLA compliance operations based on alerts or using AI/ML models.
- ***Inference Platform:*** this module consists of an Apache Spark instance. The SLA Manager interacts with it to create/terminate streaming jobs evaluating the downloaded AI/ML models. The Inference Platform sends the result of the streaming job to the SLA Manager, which upon that, triggers scaling operations on the NFV-NS instance.
- ***5Gr-SO database module:*** this module is directly responsible for the interaction with the databases. It is composed of different sub-blocks corresponding to the different catalogues of the DB. The databases used in the 5Gr-SO are stored as MongoDB collections and the modules that connect with the databases are using the PyMongo Python package, a native Python driver for the MongoDB.
- ***Southbound Interface (SBI):*** this module represents the interface for the Southbound API between the 5Gr-SO and the 5Gr-RL and it is used by the 5Gr-SO to request operational commands to the underlying infrastructure based on the IFA005 and IFA022 specification [7] [21] reports.

3.1. Final release features

The implementation of the 5Gr-SO has been accomplished in two different phases that spawned two software releases. The main features of Release 1 of the 5Gr-SO (released at M12) have been described in D2.2 (Section 2) [8]. For what concerns Release 2 (released at M24), the 5Gr-SO implements additional features, especially in support of some of the Innovations presented in D2.3 [1]. Table 2 shows the list of features comparing both releases. Next, an explanation of the enhancements included in the different modules to support the features of Release is included.

TABLE 2: 5GR-SO RELEASES

Release 1	**Release 2**	**WP2 Innovations support** (See D2.3 [1])
Support of enablers for RAN configuration procedures: • Update of VS-SO interface (server side)	Support of enablers for RAN configuration procedures: • Extended VS-SO interface to support PNFD management • New algorithms for RAN-aware resource placement • support PNF management ○ extended logic for PNFD management • Update of Graphical User Interface (GUI) to show all the information about the on-boarded PNFD	Support to I1

Support to the Vertical-oriented monitoring system: • Remove Virtual Machine (RVM) agent installation • Remote command execution through RVM agent • Prometheus collectors creation/deletion	Support to the Vertical-oriented monitoring system in additional components: • Support of native and 3rd-party client-side monitoring probes (e.g., latency probe) • Prometheus exporter for collecting customer metrics (PECCM) • Enhanced Core MANO wrapper implementation (Cloudify, OSM) to support the management of RVM agents and Day-1 scripts • Support for VNF log monitoring probes	Support to I2 and I3.
Implement a closed-loop for AI/ML based scaling of NFV-NS: • Update NSD with new IE "aimlRules" • Extend SLA Manager for an AI/ML driven orchestration of the scaling process • Extend Monitoring Manager at 5Gr-SO for assisting the SLA manager in the configuration of monitoring actions' related elements • Extend SOEc to interact with the extended SLA Manager • Additional element "AI/ML repository" Initial AIML model using Spark MLlib as a closed-loop enabler • Exploit Apache Spark for a continuous NFV-NS performance inspection and trigger the corrective actions	Support on enhanced closed-loop operations: • Define and implement interaction with 5Gr-AIML Platform interfaces to model retrieval and metrics collection for AIML-based scaling operations. In collaboration with I5, in charge of designing the 5Gr-AIML Platform. • Design additional AI/ML-driven closed-loop operations (in collaboration with I8) for more advanced AI/ML-based scaling at the 5Gr-SO. • Update of Graphical User Interface (GUI) to connect with the added notification system reporting lifecycle management operations (instantiation, scaling, termination) • Improved logging tracing feeding data-engineering pipelines for automatic orchestration operation processing	Support to I4 and I5.

	Support to Federation and Inter-domain: • Enhancement of service and resource orchestration engines to support scaling operations in composite NS deployments, including also network service federation scenarios • NBI extension supporting the interaction with the 5G-EVE platform for multi-domain scenarios	Support to I6.
	Support to Advanced algorithms and Smart Orchestration: • Non-Real Time Orchestration of next generation RAN resources • Update of ROE module to comply with different transport network abstractions (e.g., CsA, InA)	Support to I7 and I8.
	• Processing of new NSD IE expressing the need for forecasted metrics (X) • Inclusion of interactions with forecasting functional block to launch forecasting jobs during instantiation and to configure AIML-based scaling operations[1] (Y)	Support to I10.

3.1.1. NBI

In Release 2, the NBI has been extended to support Innovation I1 (described in D2.3 [1]) by including the possibility of on-boarding PNF Descriptors. Additionally, the NBI has been extended for Innovation I6 (described in D2.3 [1]) to support the interaction with 5G-EVE platform for multi-domain scenarios.

[1] As May, 31st X, and Y are available in the code released in [20]. Integration and validation are ongoing and possible enhancements will be included in the upcoming versions to be released (as they become ready) in [20]

3.1.2. SOE

In Release 2, the SOEp has been enhanced to handle the scaling of composite NFV-NS deployments under different composite NFV/NS deployment scenarios (e.g., sharing and/or federation) and covering scaling actions triggered at different levels (composite or nested level). This work has been done in the context of Innovation 6 (described in D2.3 [1]). The SOEc has been enhanced to interact with the SLA Manager to orchestrate the AI/ML-based scale operations, within the scope of Innovation 4 and Innovation 5.

3.1.3. CROOE

In Release 2, the CROOE has been evolved to handle the scaling of composite NFV-NS. It compares the deployment status before the scaling operation and after the scaling operation to determine the interconnections that need to be updated (either created/removed). This decision is communicated to the peering CROOE with a new defined message flowing through the SO-SO interface.

3.1.4. Database Module

As already said, in relation to Innovation 1 the PNF Descriptors Catalogue (and DB) has been added to this module. Further extensions in the information elements of different DBs (NS Instance NS Instance Resources) have been done to support I6 (scaling of composite NFV-NS), I2 (register RVM information), I4-I5 (AIML and data-engineering pipeline configuration), I10 (Forecasting and Inference) (described in D2.3 [1]).

The database has been extended with the addition of the AIML repository for Innovation 4 and 5, as well. It contains the AI/ML models and the inference files downloaded from the 5Gr-AIML platform to be used by the Inference platform.

3.1.5. ROOE

Within the scope of Innovation 8 (described in D2.3 [1]), the ROE module has been updated in Release 2 to comply with different transport network abstractions (e.g., CsA, InA). Related to Innovation 4, the ROOE has been extended to verify that there are enough computing resources at the underlying NFVI-PoPs before handling a scaling operation for "regular NFV-NSs".

3.1.6. Core MANO Wrappers

The implementation of the specific Core MANO Wrappers (for both Cloudify and OSM) has been updated (Cloudify) and extended (OSM) to support the management of RVM agents and Day-1 scripts within the scope of Innovation 2 (described in D2.3 [1]).

3.1.7. Monitoring Manager

In Release 2 of the 5Gr-SO, the functionality of this module has been extended within the context of Innovation 2 to consider the configuration of the log monitoring pipeline in charge of collecting log files from the different VNF instances. Additional logic has been included in the interaction (i.e., creation of exporters) with the 5Gr-VoMS and in the interaction with the Forecasting Functional Block (i.e., creation of forecasting processes) to handle workflows including forecasted metrics.

3.1.8. SLA Manager

With Release 2 of the 5Gr-SO, the functionality of this module has been extended in the framework of I4 and I5 to perform the corresponding interactions with the 5Gr-AIMLP, as further detailed in Sec. 3.1.4 in D2.3 [1]. Additional logic has been included in the interaction with the 5Gr-VoMS (i.e., creation of scrapers) to handle workflows including the Forecasting Functional Block.

4. 5Gr-RL implementation

The 5Gr-RL is responsible for the orchestration of resources and the instantiation of PNFs and VNFs over the infrastructure under its control, as well as for managing the underlying physical radio, transport, computing, and storage infrastructure.

The software architecture of 5Gr-RL is reported in Figure 7.

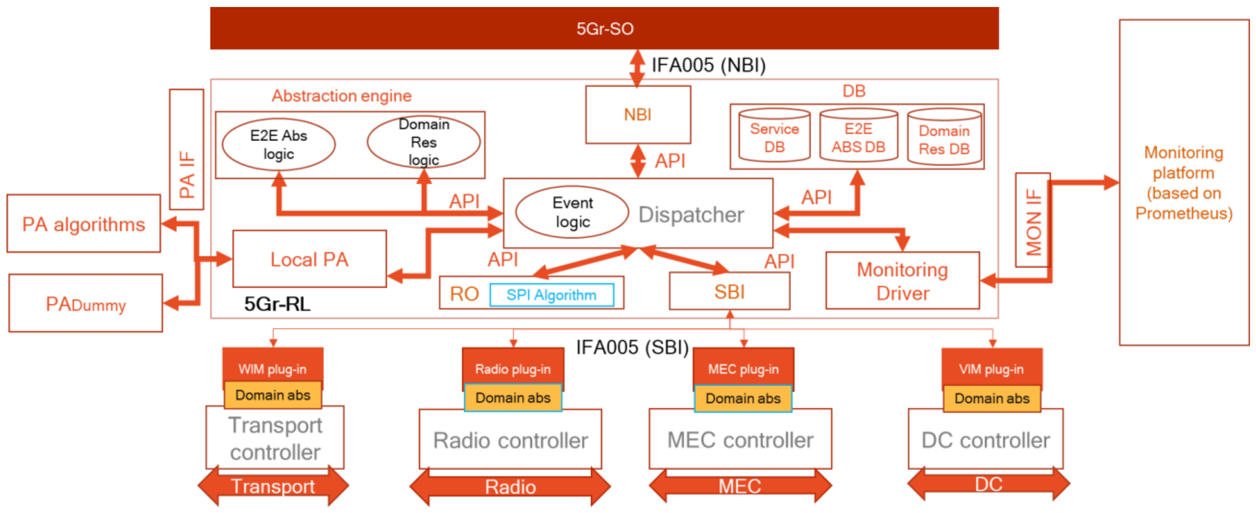


FIGURE 7: 5GR-RL ARCHITECTURE

In Release 2, several modules are extended and a new module is added (shown in blue in Figure 7) that is the Slice Performance Isolation algorithm.

5Gr-RL software is written in Java (Java version 1.8 is used) and each module is implemented as a Java class package. The following modules form the 5Gr-RL software:

- ***Abstraction Engine***: this java package implements all the algorithms and procedures related to the abstraction. The package consists of two java classes.
  - *E2E Abstraction Logic:* it retrieves from the Database the resource information of the underlying infrastructure and computes the abstract view. Such abstract view is stored in the Database and provided to 5Gr-SO when requested via northbound interface (NBI)
  - *Domain Resource Logic:* it retrieves from the abstract view the physical resources. Specifically, such a class is invoked when a resource allocation/termination request is received by 5Gr-SO via NBI and it is responsible to select the underlying resource in order to provide the required SLA in terms of bandwidth and latency.
- ***Database (DB):*** this package is a single Java class and it is a front-end connecting to an external SQL server that stores all information and details regarding the domain resources, the abstraction view and the allocated services. As MySQL [31] is used as an open-source SQL server implementation, the package uses the open-source library to handle MySQL connections.
- ***Dispatcher***: this package manages the inter-process communication between all the 5Gr-RL modules using a publish-subscribe pattern. Specifically, the dispatcher collects the list

of events that each module publishes. When a module needs to interact with another module to execute a certain functionality it posts the specific event. The post method is captured by the dispatcher and delivered to the modules that handle such events. The publish-subscribe pattern is based on the Event Bus method implemented by the open-source GUAVA library [30].

- ***Resource Orchestrator (RO)***: this package orchestrates the resource allocation/termination procedures between the Compute, Radio, Transport, and MEC domains. The orchestration implements the workflow defined in ETSI IFA022 [7].
- ***Local PA:*** this package is a front-end to contact a dedicated and specialized local Placement Algorithm (PA) that provides a local optimization of the physical resources. This package is optional and activated via a configuration file. When enabled, the Abstraction Engine uses the local PA as enforcement to select the physical resource from the abstraction view.
- ***Monitoring Driver:*** this package is a front-end to handle the communication with the monitoring platform (based on Prometheus). It uses the monitoring interface exposed by the Monitoring Platform to register itself to the platform, create performance monitoring jobs and get alert notifications. Moreover, it handles notifications about the status of infrastructure resources (i.e., compute, storage, networking, radio, and MEC resources) by posting specific events to the dispatcher.
- ***Northbound Interface (NBI)***: this package acts like a web server and implements the REST API IFA005 based communication towards 5Gr-SO. Such package is generated using Swagger Code Generator [28] .
- ***Southbound Interface (SBI)***: this package handles the IFA005 [5] communication with the plugins that are based on REST API. In a similar way to NBI, the package acts as a REST client and the code is generated using Swagger Code Generator [28].
- ***WIM/VIM/Radio/MEC plugins***: these are standalone web servers that translate SBI REST API commands into commands specific to the domain controller. Table 3 reports the list of available plugins.

TABLE 3: 5GR-RL PLUGINS

Plugin list	Notes
Radio Dummy Plugin	used for internal test
Ericsson Radio Plugin	used in COMAU pilot
VIM Dummy Plugin	used for internal test
Kubernetes VIM plugin	tested with Kubernetes version 1.19
Openstack VIM plugin	tested with Openstack Queen release
Xen VIM Plugin	tested with Xen version 6.x
Dummy MEC plugin	used for internal test
ETSI MEC plugin	MEC plugin compliant to ETSI MEC 10-2 [29]
WIM Dummy plugin	used for internal test
Ericsson WIM plugin	WIM Transport domain for COMAU pilot

ONOS WIM plugin	WIM plugin interacting with ONOS controller
Optical WIM plugin	configure optical WDM node

The 5Gr-RL code is released open-source under GPL license and available in [17].

4.1. Final release features

Table 4 depicts the functionality developed for the 5Gr-RL in the two releases, linked to the innovations described in [1].

TABLE 4: 5GR-RL RELEASES

Release 1	**Release 2**	**WP2 Innovations support** (See D2.3 [1])
RAN slice configuration: • Initial PNF support • Preliminary enforcement of RAN configuration • Dummy radio plugin for integration test	RAN slice configuration: • Extended VS-RL interface (server side) • Evolution of RAN abstraction o Added PNF capability on Radio Capable NFVI POP • Extended RL interfaces o Added PNF management API in 5Gr-RL NBI o Added PNF management API in 5Gr-RL SBI o Added MF management API in 5Gr-RL NBI • RAN configuration enforcement o MF support compliant to IFA 024 • Evolution of radio plugin Ericsson radio plugin (based on the hardware used in COMAU premises)	Support to innovation I1 – RAN segment in network slice
Network Slice and Smart orchestration: • Support requesting expansion/abstraction computations via the new REST API with the	Network Slice and Smart orchestration: • Support for retrieving the model from the AIML platform to support	Support to innovation I8 – Smart orchestration and resource control algorithms

Resource Allocation (RA) server • Support of LLs bound to different Class of Services (e.g., Gold, Silver, Bronze) used for the CSA algorithm • For the InA algorithm, creation of the request among all the NfviPop pairs and deriving the LLs consumed by the 5Gr-SO • Support deployment of baseline QoS policy (performance isolation).	anomaly detection (e.g., degraded WAN links) • Resolving affected WAN connections due to an anomaly detection decision • Support for deploying monitoring jobs of the underlying WAN infrastructure • Support of triggering asynchronous re-optimization mechanisms of the existing WAN connections • Interaction with AI/ML platform • Enable alarms on QoS policy from monitoring platform. • Support more complex QoS policies (e.g., data plane arbitration via programmable scheduling)	
PA algorithm named Resource Allocation (RA) Server: • Support processing and serving abstraction computations through new REST API with the core 5Gr-RL • CSA algorithm for computing (expanding) feasible WAN paths bound to a specific LL CoS • InA algorithm for computing WAN paths among pairs of NfviPops used to derive the abstraction of LLs towards the 5Gr-SO • Support of the new designed 5Gr-RL and RA server REST API enabling bulk of	PA algorithm named RA Server: • Support of an algorithm targeting global concurrent optimization of multiple WAN connections impacted by a detected network anomaly • Support of the 5Gr-RL and RA server REST API to cover restoration and re-optimization algorithms at the RA server: Extending the REST API (at both client and server sides) carrying information about WAN paths to be restored / re-optimized	Support to innovation I8 – Smart orchestration and resource control algorithms It is released in a separate branch (named "cttc-rl")

expansion/abstraction computations		

5Gr-RL has implemented for release 1 and 2, some features that are related to innovation I1 and I8 described in D2.3 [1].

Concerning Innovation I1, 5Gr-RL implements some additional features to perform the RAN slice configuration. With respect to the plan proposed in D2.2 [8], there are some minor deviations related to PNF implementation that requires some tuning and refinement with respect to the feature released in version 1 of the software. Specifically, the abstraction was modified in order to report PNF capability in Radio Abstract NFVI POP. Moreover, we adopt the implementation option to pass the RAN slice parameter from 5Gr-VS to 5Gr-RL via 5Gr-SO extending the related interface and avoid implementing a direct interface between 5Gr-VS and 5Gr-RL.

Concerning Innovation I8, 5Gr-RL implements the support for slice performance isolation that introduces a new workflow. Such workflow impacts all internal modules and insert a new submodule that is a new algorithm for resource allocation taking into account the slice isolation policy. Moreover, The RA server has been improved to handle the re-computation/restoration of a received set of existing WAN connections affected by a detected network anomaly (e.g., link failure, degraded performance of a node/link, node failure). To this end, the framework of the RA server is improved in two aspects:

i. the 5Gr-RL – RA server API to support new request/response messages for the automatic re-computation of a bulk of disrupted WAN connections along with identifying the network [27].
ii. a novel WAN path computation algorithm referred to as Global Concurrent Optimization (GCO) mainly devised to handle the on-line restoration of a disrupted set of WAN connections.

The planned mechanism reported in D2.2 [8] about targeting the re-optimization of WAN connections has not been explicitly addressed in Release 2. It is expected that this mechanism could be eventually supported adding new capabilities in the GCO algorithm as well as enhancing the 5Gr-RL- RA server API.

4.1.1. Abstraction Engine

The abstraction engine is extended to support Innovation I1 by including the PNF capability information within the Abstract node representing the Radio Domain (Radio Capable NFVI POP). Detailed information on the PNF information model can be found in D2.3 [1].

4.1.2. Database

The database was extended to support Innovation I1 by adding new tables to record information about:

- PNF capability information in the abstract view.

- MF association with VNF/PNF according to IFA024 specification (RAN enforcement configuration feature).

The database was extended to support Innovation I8 by adding new tables to record information about:

- QoS Policy that is used as input by the SPI algorithm.
- QoS Service Policy that records the services using the slice performance isolation and which policy is adopted.

Refer to D2.3 [1] for more details on MF, PNF, QoS Policy information model.

4.1.3. Dispatcher

New events are added to handle MFs to support Innovation I1. Specifically, to implement the MF workflow described in D2.3 [1]. The new events are:

- *E2EPhysicalAllocateRequest*. This event is handled by the Abstraction Engine module to receive the PNF instantiation request
- *E2EPhysicalAllocateReply*. This event is handled by the NBI module to receive the response information about PNF instantiation request
- *PhysicalAllocateDBQuery*. This event is handled by the Database module to receive the request of domain information to allocate the PNF instance
- *PhysicalAllocateDBQueryReply*. This event is handled by the Abstraction Engine module to receive the response of domain information where to allocate the PNF instance. It also includes the MF to be applied
- *E2EPhysicalAllocateInstance*. This event is handled by the Resource Orchestrator module to receive the request to allocate the PNF instance on a specific domain. It also includes the MF to be applied
- *PhysicalAllocateReq*. This event is handled by the SBI module to receive the request to allocate the PNF instance on a specific domain. It also includes the MF to be applied
- *PhysicalAllocateRadioReply*. This event is handled by the Resource Orchestrator module to receive the response from radio domain about the outcome of the PNF instance allocation
- *PhysicalAllocateVIMReply*. This event is handled by the Resource Orchestrator module to receive the response from VIM domain about the outcome of the PNF instance allocation
- *E2EPhysicalAllocateInstanceOutcome*. This event is handled by the Abstraction Engine to receive the outcome of the PNF instance allocation
- *PhysicalAllocateDBQueryOutcome*. This event is handled by the Database module to receive the outcome of the PNF instance allocation
- *E2EPhysicalTerminateRequest*. This event is handled by the Abstraction Engine to receive the PNF termination request
- *E2EPhysicalTerminateReply*. This event is handled by the NBI module to receive the response information about PNF termination request

- *PhysicalTerminateDBQuery*. This event is handled by the Database module to receive the request of domain information to terminate the PNF instance
- *PhysicalTerminateDBQueryReply*. This event is handled by the Abstraction Engine module to receive the response of domain information where to terminate the PNF instance. It also includes the MF to be deleted
- *E2EPhysicalTerminateInstance*. This event is handled by the Resource Orchestrator module to receive the request to terminate the PNF instance on a specific domain. It also includes the MF to be deleted
- *PhysicalTerminateReq*. This event is handled by the SBI module to receive the request to terminate the PNF instance on a specific domain. It also includes the MF to be deleted
- *PhysicalTerminateRadioReply*. This event is handled by the Resource Orchestrator module to receive the response from radio domain about the outcome of the PNF instance termination
- *PhysicalTerminateVIMReply*. This event is handled by the Resource Orchestrator module to receive the response from VIM domain about the outcome of the PNF instance termination
- *E2EPhysicalAllocateInstanceOutcome*. This event is handled by the Abstraction Engine to receive the outcome of the PNF instance termination
- *PhysicalAllocateDBQueryOutcome*. This event is handled by the Database module to receive the outcome of the PNF instance termination

4.1.4. Resource Orchestrator

The Resource Orchestrator (RO) was extended to support innovation I8 by adding a new submodule that implements the slice performance algorithm (SPI algorithm block).

The SPI algorithm is invoked by the RO when slice performance isolation is requested for connectivity. This algorithm retrieves the list of QoS policies from the Database, and applies the one that is suitable to the SLA parameters (i.e., bandwidth, latency). Details on QoS policy selection can be found in D2.3 [1]. The policy is then passed on as an argument to the WIM plugin via the new SBI API.

4.1.5. Local PA algorithms

The pool of PA algorithms supported in the RA server has been extended in Release 2 to support novel functionalities on the automatic WAN computation mechanisms such as re-computation/restoration. That is, besides the Release 1 PA algorithms devised for WAN Logical Link (LL) abstraction and WAN expansion resource computation (i.e., InA and CSA) reported in D2.2 [8], a novel GCO algorithm [27] is added in the RA server for Release 2. To this end, and as aforementioned, this is enabled through pertinently extending the 5Gr-RL – RA server API. The API improvement embraces the integration of new request/response messages for demanding the re-computation of a specific set of WAN connections. Such a re-computation functionality is useful to automatically handle (e.g., restoring) network anomalies (e.g., link failure) disrupting/degrading the required performance of a set of existing WAN connections. In a nutshell, upon receiving a set of WAN connections to be re-computed, the GCO algorithm seeks an effective solution and global solution.

In other words, the algorithm aims at exploring a broad space of candidate solutions and chooses the one leading to attain: i) the highest re-computation success; ii) the most efficient use of the WAN resources.

4.1.6. NBI

New APIs are added to support innovation I1 to handle PNF allocation/termination requests. The new API are:

- *HTTP POST "physical_resources"* to request the PNF instatiation
- *HTTP DELETE "physical_resources"* to delete an existing PNF instance

See D2.3 [1] for more details.

4.1.7. SBI

New APIs are added to support innovation I1 to handle PNF allocation and termination requests, and MF allocation and termination requests. The new API are:

- *HTTP POST "pnfs"* to request the PNF instantiation
- *HTTP DELETE "pnfs"* to delete an existing PNF instance

New API is added to support innovation I8 to request the application of QoS to WIM plugin. The new API are:

- *HTTP POST "/onos/v1/dpe/slice"* to request the instance of a slice performance isolation (SPI)
- *HTTP DELETE "/onos/v1/dpe/slice"* to delete an existing SPI instance

See D2.3 [1] for more details.

4.1.8. Ericsson Radio Plugin

A new Radio plugin is implemented to translate SBI commands into Ericsson's radio controller, which is responsible to configure the Ericsson equipment used in the COMAU pilot. As the plugin uses commercial software and hardware of the vendor, this plugin is not released as open source.

4.1.9. ONOS WIM plugin and ONOS applications

The 5Gr-RL is connected to the WAN Infrastructure Manager (WIM) plugin, which exposes the transport domain resources (SDN-based) view via an IFA005 interface. The plugin allows the 5Gr-RL to query the actual status or the network resources, allocate part of them, or release some of the resources already allocated.

The existing ONOS WIM plugin was extended to support innovation I8 to apply the Slice performance isolation rules selected by the SPI algorithm using the QoS policy information model (see D2.3 [1] for details). The rules are applied both in OpenFlow switches and P4 switches. As each type of switch has its specificity, a separate subsection describes the implementation of ONOS WIM plugin for each type of switch.

4.1.9.1. ONOS WIM plugin for Openflow switch

From the implementation perspective, the WIM plugin implements the integration between the SBI of the 5Gr-RL and the ONOS SDN controller. In particular, upon request, the plugin can provide an abstracted view to the 5Gr-RL that hides the specific characteristics of the underlying infrastructure by representing the transport network as a set of logical links that interconnect NFVI-PoPs gateways. The abstracted view is composed of these logical links, characterized by performance parameters (e.g., latency, available bandwidth, total bandwidth). The logical link abstraction is technology-agnostic (e.g., independent from the fact that the logical link is an optical link or an MPLS link) and does not provide any detail regarding the WAN domain (e.g., switches characteristics, switches inter-connectivity, traffic steering features, etc.). Furthermore, it also allows for requesting the allocation of virtualized network resources over one or more virtual links, used for deploying a specific network service, by enforcing the necessary parameters to create isolated slices.

With respect to Release 1, the WIM plugin has been extended to support the new features introduced by the performance isolation application enhanced with the QoS guarantees (i.e., per-slice traffic, bandwidth and delay isolation) in an SDN-based network. More specifically, in Release 2, the WIM plugin is able to handle requests that aim at setting up network slices with specific QoS characteristics and enforcing the bandwidth isolation feature. To do so, new parameters characterizing the slices have been considered for the allocation operation, such as the maximum and minimum bandwidth, the maximum burst size and the Active Queue Management (AQM) parameters for delay guarantees. Once those parameters are extracted from the request, the plugin interacts with the ONOS SDN controller to first set up the slice and then enforce the QoS guarantees. More specifically, this operation allows to dynamically enforcing the forwarding rules on the data plane devices (i.e., OpenFlow and P4 switches) according to the required QoS parameters.

The source code of the WIM extension is available as open-source, under the Apache v2.0 license, on GitHub [10]. For enabling the communication, a REST interface has been implemented for the ONOS VPLS application that has been merged in the ONOS master distribution [37].

4.1.9.2. ONOS applications for Slice management and P4-based switches support

Currently, ONOS only provides traffic isolation across slices without bandwidth or delay guarantees. Therefore, *QoSlicing* is introduced as an application to manage both meters and virtual queues across slices and expose a unified interface towards the upper layers (i.e., the ONOS WIM plugin, described in Section 4.1.9.1), which ultimately provides bandwidth isolation seamlessly under a mix of P4 and OpenFlow switches.

In this context, *P4-slicing-pipeconf* is also introduced as an ONOS application that integrates VPLS (Virtual Private LAN Service) native ONOS applications with P4 protocol. VPLS allows the creation of network slices using the VLAN protocol, which forwards packets using VLAN tags. VLAN support was added to the P4 implemented pipeline used in the application.

Figure 8 shows the slice creation workflow, showing the interaction between the ONOS WIM plugin and ONOS applications (VPLS, QoSlicing, and P4-slicing-pipeconf). In particular, after receiving the configuration from the SBI of the 5Gr-RL (message 1), the ONOS WIM plugin sends two different requests:

- the creation of a slice (messages 2, 3 and 4), sent to the VPLS application;
- the provision of QoS parameters (messages 5, 6' and 7), sent to the QoSlicing application. As a result of this interaction, the QoSlicing application sets the AQM parameters in the P4-slicing-pipeconf (messages 8 and 9').

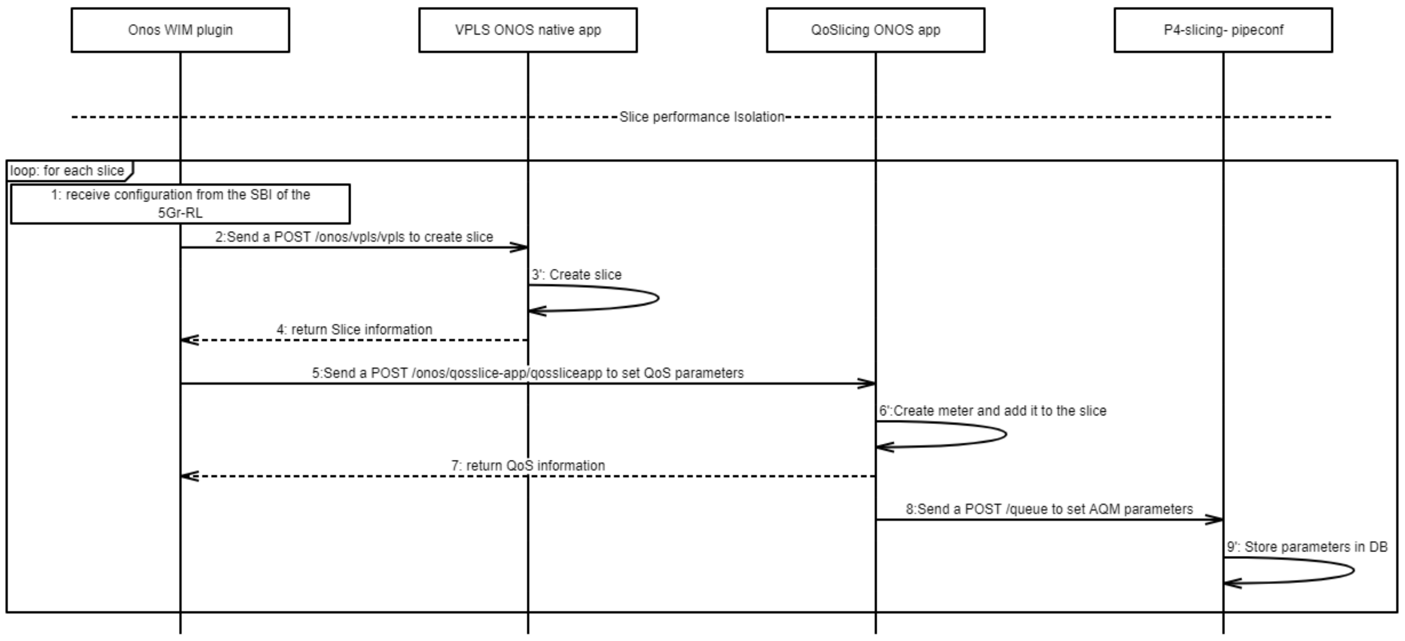

Onos WIM plugin
VPLS ONOS native app
QoSlicing ONOS app
P4-slicing- pipeconf
Slice performance Isolation
loop: for each slice
1: receive configuration from the SBI of the 5Gr-RL
2:Send a POST /onos/vpls/vpls to create slice
3': Create slice
4: return Slice information
5:Send a POST /onos/qosslice-app/qossliceapp to set QoS parameters
6':Create meter and add it to the slice
7: return QoS information
8:Send a POST /queue to set AQM parameters
9': Store parameters in DB

FIGURE 8: INTERACTIONS BETWEEN ONOS WIM PLUGIN AND ONOS CONTROLLER

The P4-slicing-pipeconf application contains a P4 interpreter to control and configure P4 switches with mutable pipelines, allowing the mapping between an ONOS treatment and specific actions to execute in the P4 switches, using the included ONOS Protocol Independent model framework [11].

To meet bandwidth and delay guarantees, virtual queues are introduced in the P4 pipeline. By using virtual queues to perform per-slice policing and leveraging P4's real queuing latency monitoring capability, the delay requirement of each slice can be met.

With respect to Release 1, the P4-slicing extends SBI [8] to enable delay guarantees, adding some new parameters to the POST request sent from the ONOS plugin to the P4-slicing-pipeconf application:

- M_DELAY: Marked Delay.
- T_DELAY: Transmission Delay.
- C_DELAY: Congestion Delay.

These parameters are explained in depth in D2.3 [1].

The Virtual queues are implemented by using the egress tables, which are added in this new release, so when packets match on some actions, some registers are updated to store the global timestamp and the virtual queue's size.

The source code of all these developments is available in the 5Gr-RL repository [17].

4.1.10. Kubernetes VIM plugin

The Kubernetes VIM plugin provides the IFA005 type interface for management of the specific VIM domain, including computing and communication resources. The main goal of the VIM plugin is to translate the RL requests to the Kubernetes API. In order to create PODs inside the Kubernetes cluster, the VIM plugin translates and passes 5Gr-RL's requests to the Kubernetes API (method POST) for the compute creation.

The initial version of the Kubernetes plugin for the 5Gr-RL used the default configuration of Kubernetes. The default configuration of the Kubernetes cluster does not support VLAN-based external connections to PODs. The container network interface (CNI) supports Kubernetes to use different network technologies. Specifically, Multus [13] is a CNI plugin for Kubernetes that allows attaching multiple network interfaces to the PODs and connects PODs to external WIM transport networks.

The Kubernetes VIM plugin was extended, and it provides the interface for managing the Multus CNI and supports VLAN-based networks. This plugin's extension allows the 5Gr-RL to create PODs connected to the external VLAN-based networks.

The VIM plugin also provides the range of available VLAN IDs to the 5Gr-RL (HTTP method GET). The 5Gr-RL uses this data in requests for network resources creation inside the VIM. The VIM plugin supports requests (HTTP method DELETE) for deletion of the pod and network inside the Kubernetes cluster. It also supports requests (HTTP method GET) to get status about pods inside the Kubernetes cluster.

The source code of the Kubernetes VIM plugin is available as open-source, under the Apache v2.0 license, on GitHub [12].

5. 5Gr-AIML Platform implementation

The 5Growth AIML (Artificial Intelligence and Machine Learning) Platform (5Gr-AIMLP) enables the exploitation of AI/ML models for the various decision-making processes necessary in a 5G management and orchestration stack, and is in charge of the ML model lifecycle management (including ML model uploading, catalogue building, and ML model training). The relationship between the 5Gr-AIMLP and the project innovations described in D2.3 [1] is as follows:

- It realizes I5, implementing an "AIML as a Service" platform;
- It contributes to I2, as it leverages the performance metrics collected through the 5Gr-VoMS to build data sets for model training/retraining. Performance metrics reported by the 5Gr-VoMS are also used for the decision-making process during the inference phase;
- It contributes to I4, as it can provide any 5Gr-entity with a trained or pre-trained AI/ML model that can then be run for closed-loop system stability;
- It contributes to I8, as it allows AI/ML-based algorithms for efficient, automated service and network management.

The models can be uploaded to the 5Gr-AIMLP by any authorized external user. Such models can be already trained and onboarded to then perform inference, or they can be untrained models.

In the latter case, if no dataset is uploaded along with the untrained model, the 5Gr-AIMLP can exploit the data collected through the 5Gr-VoMS platform about network/computing resource utilization, or performance. The configuration of the monitoring platform to gather the monitored data, its aggregation through Kafka and Spark Streaming, and their feeding as input for real-time model execution in the corresponding 5Growth building block also need to be properly set up. Models stored in the 5Gr-AIMLP can be accessed by any entity in the 5Growth architecture through a REST interface.

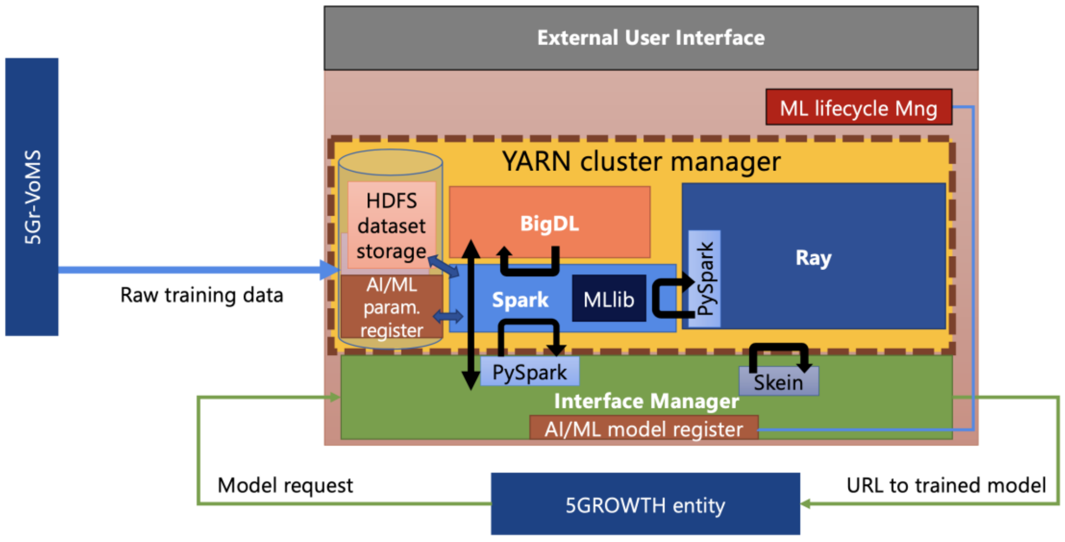


FIGURE 9: 5GR-AIMLP REPRESENTATION

The 5Gr-AIMLP has been implemented by developing the following components:

- External User Interface, i.e., a Graphical User Interface (GUI) through which users can upload the ML models, and the related datasets and inference classes;
- Model Registry, which records the models uploaded to the platform, their metadata, and pointers to the stored models and associated files;
- Lifecycle Manager, which is in charge of the model's lifecycle;
- Interface Manager, i.e., a REST API that processes the ML model requests coming from the architectural stack and forwards them to the proper block inside the computing cluster.
- Computing cluster, implemented through Apache Hadoop and leveraging Yet-Another-Resource-Negotiator (YARN) for the computing resources management, and the Hadoop Distributed File System (HDFS) for the storage of datasets and models. The YARN cluster nodes have access to different AI/ML frameworks, according to the requested model type, namely, Spark, BigDL, and Ray.

A graphical representation of the 5Gr-AIMLP is presented in Figure 9, while the above components are described in more detail below. The 5Gr-AIMLP is also available in the software repository at [18].

External User interface: To access the 5Gr-AIMLP, the user first fills in the sign-up page, specifying the login credentials and the entity along with some contact information, as shown in Figure 10.

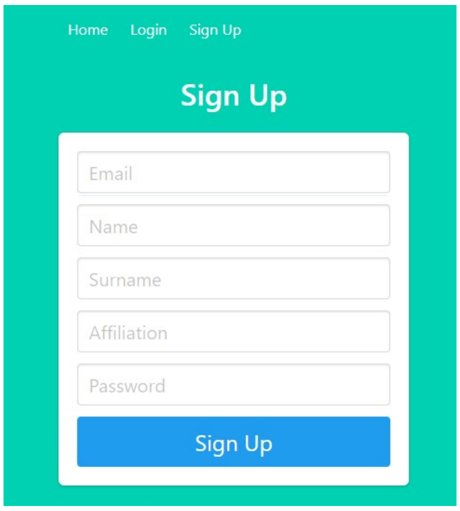


FIGURE 10: 5GR-AIMLP EXTERNAL USER INTERFACE – SIGN UP PAGE

The user can then login and access the main features of the 5Gr-AIMLP GUI, thanks to which new ML models can be uploaded. To do so, the user should select to either upload a new (already) trained model, or a new (yet to be trained) model, and fill in the necessary information accordingly, as shown in Figure 11.

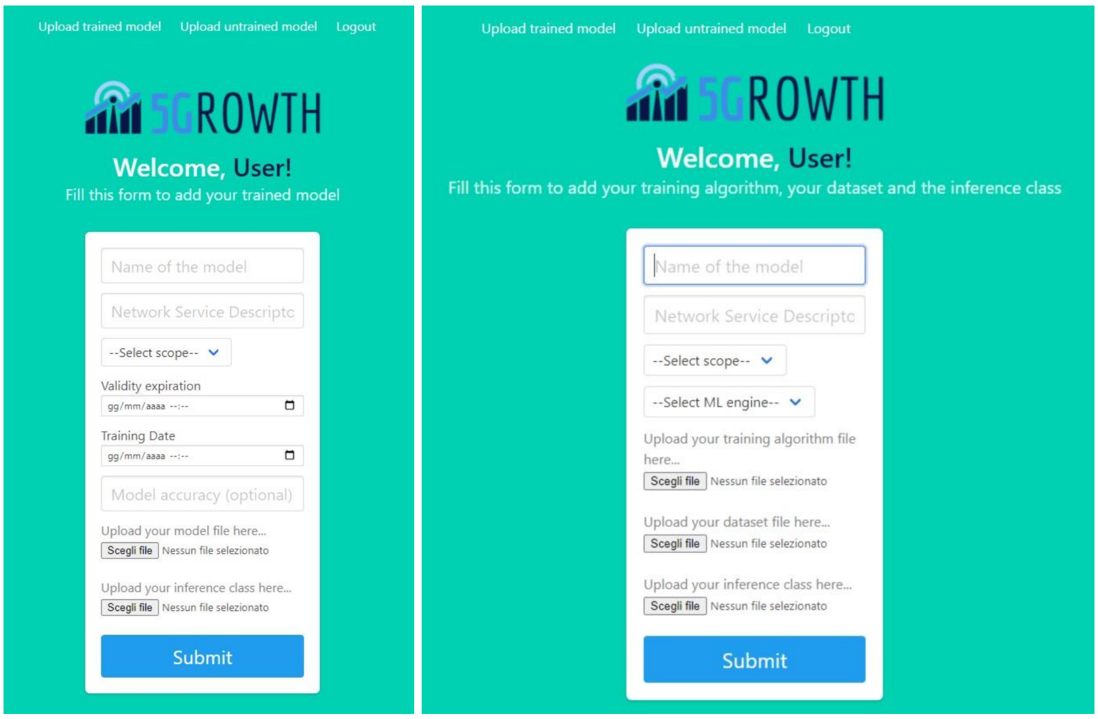

Upload trained model
Upload untrained model
Logout
5GROWTH
Welcome, User!
Fill this form to add your trained model
Name of the model
Network Service Descripto
--Select scope--
Validity expiration
gg/mm/aaaa --:--
Training Date
gg/mm/aaaa --:--
Model accuracy (optional)
Upload your model file here...
Scegli file Nessun file selezionato
Upload your inference class here...
Scegli file Nessun file selezionato
Submit
Upload trained model
Upload untrained model
Logout
5GROWTH
Welcome, User!
Fill this form to add your training algorithm, your dataset and the inference class
Name of the model
Network Service Descripto
--Select scope--
--Select ML engine--
Upload your training algorithm file here...
Scegli file Nessun file selezionato
Upload your dataset file here...
Scegli file Nessun file selezionato
Upload your inference class here...
Scegli file Nessun file selezionato
Submit

FIGURE 11: 5GR-AIMLP EXTERNAL USER INTERFACE - UPLOADING OF AN ALREADY TRAINED MODEL (LEFT) AND A YET-TO-BE-TRAINED MODEL (RIGHT)

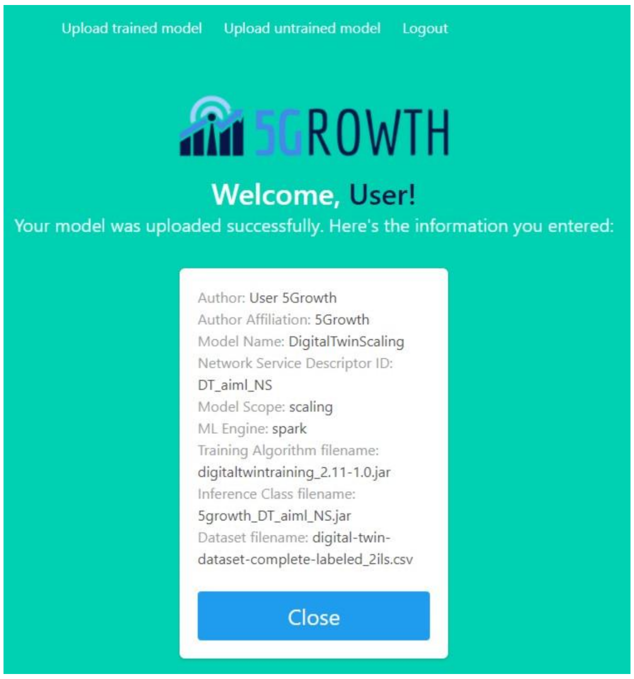
If the user wishes to load a model that has already been trained offline, the information to be provided is:

1. Name of the model
2. Network Service Descriptor ID
3. Scope of the model
4. Validity expiration and training dates
5. Model accuracy (optional)
6. The .jar files of the model and the inference class.

If instead the user wishes to load a model that has to be trained by the 5Gr-AIMLP autonomously, the information to be provided is:

1. Name of the model
2. Network Service Descriptor ID
3. Scope of the model
4. The engine to use in the training phase
5. The .jar files of the model and the inference class
6. The dataset to be used for the training phase.

Once the user has filled in and submitted the forms to the platform, a confirmation message is displayed (as shown in Figure 12, which refers to the correct uploading of a model to be trained).



Welcome, User!

Your model was uploaded successfully. Here's the information you entered:

Author: User 5Growth
Author Affiliation: 5Growth
Model Name: DigitalTwinScaling
Network Service Descriptor ID: DT_aiml_NS
Model Scope: scaling
ML Engine: spark
Training Algorithm filename: digitaltwintraining_2.11-1.0.jar
Inference Class filename: 5growth_DT_aiml_NS.jar
Dataset filename: digital-twin-dataset-complete-labeled_2ils.csv

FIGURE 12: 5GR-AIMLP EXTERNAL USER INTERFACE - CONFIRMATION MESSAGE

5.1. Exposed interfaces

Interface Manager. The REST interface offers the following functionalities to 5Growth entities:

1. Discover and download AI/ML models that are hosted and/or trained by the Platform;
2. Start and stop VNF metric collection upon the instantiation and termination of a Network Service. The metrics are then available as a training set, aggregated according to the Network Service Descriptor ID.

The basic workflow to use the 5Gr-AIMLP to retrieve a trained model is as follows:

1. Select the model of interest between those available in the 5Gr-AIMLP.
2. Given the *Model*, parse its "model_file_url" field to get the trained model file path to download. If a model has been recently uploaded or if an error was encountered during the training phase, the status may not be "trained". In this case, the *model_file_url* will be set to "null".

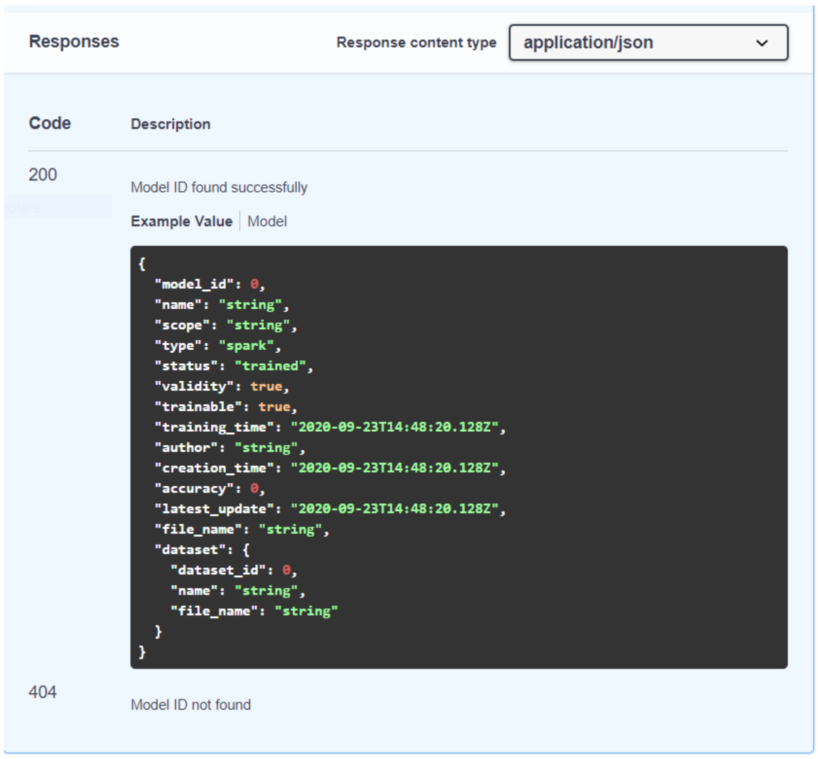
Requests and responses are in JSON format; no authentication is required as of now. The defined methods are reported in Table 5.

TABLE 5: 5GR-AIMLP API

GET /models	Retrieve the list of all *Model* objects hosted in the AIMLP. Optionally, to filter the list to a subset of all available models, it supports the parameters: ○ "scope": scope of the models to return ○ *"nsd_id"*: Network Service Descriptor ID of the model to return. Example: GET /models?scope=scaling&nsd_id=DT_aiml_NS Responses: ○ 200: [Model] Query is successful, a list of Model object is returned. The list may be empty
GET /models/{model_id}	Retrieve the *Model* with the specified "*model_id*". Responses: ○ 200: Model The found *Model* is returned. ○ 404 There is no model with the specified "*model_id*"
GET /models/{model_id}/file	Download the binary archive containing the trained model and the inference class. This operation can only be performed if the *status* of the Model is "trained". Responses: ○ 200: binary stream The binary file is a zip archive containing the inference class and the trained model. The trained model, according to the *ml_engine*, can also be a zip file. If the *ml_engine* is "spark", then it will be a zip archive named "model.zip" containing the folder in which the training algorithm saved the trained model (or pipeline). ○ 404 The *model_id* has not been found or the Model with the provided "*model_id*" is not in "trained" *status*, therefore the trained model cannot be downloaded.

GET /dataset_collectors/{kafka_topic}	Retrieve an active collector with the specified "kafka_topic". A collector is active if its status is not "terminated". Terminated collectors are not returned, a 410 status code is returned instead. Responses: o 200: Dataset Collector The active dataset collector that is reading from the specified "kafka_topic". o 404 No collector found with the specified "kafka_topic". o 410 The collector has already been terminated
PUT /dataset_collectors/{kafka_topic}	Start a new dataset collector upon Network Service instantiation. The collector will retrieve the service metrics from a dedicated Kafka topic and store them in the AIMLP. Parameter: *Dataset Collector* Responses: o 200: Dataset Collector The newly created Dataset Collector object is returned. Its status is "started". o 400 The request could not be parsed correctly
DELETE /dataset_collectors/{kafka_topic}	Terminate an active dataset collector. This method is effective only if the Dataset Collector status is "started". The Collector status will then change to "processing" while the AIMLP processes the collected metrics and finally to "terminated". Responses: o 200: Dataset Collector The updated Dataset Collector object is returned with a "processing" status. o 404 No collector found with the specified "kafka_topic". o 410 The collector has already been terminated.

The object structures are reported below, while an example of the Model object is shown in Figure 13.



Code	Description
200	Model ID found successfully
404	Model ID not found

Example Value | Model

```
{
  "model_id": 0,
  "name": "string",
  "scope": "string",
  "type": "spark",
  "status": "trained",
  "validity": true,
  "trainable": true,
  "training_time": "2020-09-23T14:48:20.128Z",
  "author": "string",
  "creation_time": "2020-09-23T14:48:20.128Z",
  "accuracy": 0,
  "latest_update": "2020-09-23T14:48:20.128Z",
  "file_name": "string",
  "dataset": {
    "dataset_id": 0,
    "name": "string",
    "file_name": "string"
  }
}
```

FIGURE 13: 5GR-AIMLP EXAMPLE OF A *MODEL* OBJECT

6. 5Gr-Forecasting Functional Block implementation

The 5Growth Forecasting Functional Block (5Gr-FFB) is a functional block, introduced in the final release of the platform, devoted to the computation of forecasting values for given metric(s) of a Network Service. It behaves like a probe, which, consuming the current data related to the selected metric(s), provides a forecasted data stream according to the used model.

The 5Gr-FFB has been designed to interact with other components of the 5Growth architecture, i.e., the 5Gr-SO, the 5Gr-VoMS and the 5Gr-AIMLP. It has been implemented in python, relying on different standard libraries.

The relationship between the 5Gr-FFB and the project innovations described in D2.3 [1] is as follows:

- It realizes and implements I10, for the forecasting of service-related metrics;
- It contributes to I4, as it can provide forecasting metrics to be consumed in the closed-loop system stability.

More details on the operations performed by the forecasting functional block and the interaction workflow with other modules are reported on D2.3 [1].

In general, the 5Gr-FFB interacts with other 5Growth orchestration modules:

1. 5Gr-SO: receiving as input all the data required, to activate a forecasting job. The input data includes: the NS details, the metric(s) to be forecasted and the current metric(s) objecID(s).
2. 5Gr-AIMLP: to retrieve the model to be used for the forecasting. It can be a pre-trained model (pre-uploaded) or it can be trained using current data.
3. 5Gr-VoMS: to gather the current monitored data, stored in Prometheus, and to enable Prometheus jobs to poll forecasting data. In this perspective, the 5Gr-FFB is able to activate ad-hoc scraper jobs, in order to enable the streaming of the current monitored data (serving as input for the forecasting) on a specific topic in Kafka.

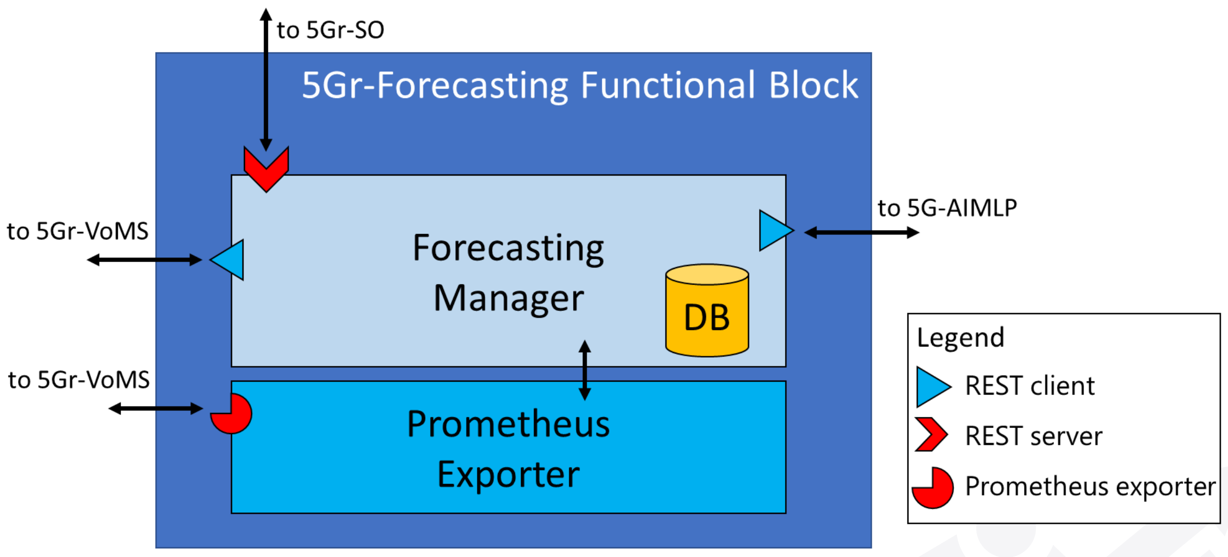


FIGURE 14. 5GR-FFB HIGH LEVEL SOFTWARE ARCHITECTURE

The 5Gr-FFB software architecture is shown in Figure 14 and it is composed by the following main blocks:

- **Forecasting Manager**: this block is the core of the 5Gr-FFB. It consists in a python-based project, that includes different modules with different roles:
    - The basis of the module is based on a flask REST server that implements the APIs to other 5Growth orchestration stack functional blocks. The details of the exposed/implemented APIs are detailed in Section 6.1. In general, the APIs allow to both start and stop a forecasting job, according to the input parameters, and to retrieve information regarding the active jobs.
    - A specific module has been defined in order to implement a forecasting job. The module is based on Keras Tensoflow library, where the forecasting algorithms designed and studied under the Innovation 10 (see Section 3.2.3 in D2.3 [1]). Each forecasting job presents an instance of Kafka Consumer, able to retrieve current data related to the performance metric to be forecasted, streamed on a dedicated Kafka topic. Each forecasting job runs a specific model. In general, the model is trained at the 5Gr-AIMLP and downloaded on the forecasting purpose. However, 5Gr-FFB can exploit forecasting algorithms based on classical time series analysis (i.e., double exponential smoothing, DES, and triple exponential smoothing, TES), not AI/ML-based, that can be run directly without resorting to the 5Gr-AIMLP
    - The overall state of the module is maintained in a dedicated database. In particular, all the details of the instantiated forecasting jobs are kept up-to-date, including the job_id, the input parameters, used for the forecasting job activation, the ad-hoc created kafka_topic, the selected model name and the downloaded model file.
- **Prometheus exporter**: the 5Gr-FFB exposes a dedicated API (to the 5Gr-VoMS) in order to provide the forecasting data. The API implements a Prometheus exporter capable of expose forecasting data. More specifically, for each forecasting job a dedicated Prometheus job is enabled at the 5Gr-VoMS, periodically querying the forecasting exporter for new forecasting data. When a new GET request is received for a given forecasting job, the forecasting model is run, computing the related forecasting data.

An example of workflow involving the 5Gr-FFBs shown in Figure 15 and described below:

1. The NSD at the 5Gr-SO includes the information related to the metric (i.e., Metric) to be forecasted. The 5Gr-SO configures the monitoring job at the 5Gr-VoMS related to the Metric.
2. Once enabled, the 5Gr-VoMS sends the id (i.e., monID) related to the enabled monitoring job.
3. The 5Gr-SO triggers the 5Gr-FFB process passing as input parameters: nsId. vnfdId, Metric, nsdId and instantiation level.
4. The 5Gr-FFB requests a trained model from the 5Gr-AIMLP (in general it can ask for training a model if needed).
5. The 5Gr-AIMLP provides the trained model.

6. 5Gr-FFB creates a new Kafka topic (i.e., FTopic) on the Kafka cluster, dedicated to the transmission of the values related to current data of Metric.
7. The corresponding reply is returned.
8. A new scraper job, filtering the data according to input parameters (i.e., nsId. vnfdId, Metric), is activated, pushing the data on FTopic.
9. The scraper job id is returned.
10. An instance of a Kafka consumer is activated, receiving current data of Metric on FTopic.
11. The forecasting job (i.e., FjobID) is started, running the stored model and using as input parameters the metric values received by the Kafka consumer.
12. A Prometheus job is configured at the 5Gr-VoMS in order to periodically get the forecasting data related to the enabled forecasting job (identified by FjobID), using the Prometheus exporter interface, acting as a standard probe.
13. The Prometheus job id is returned (i.e., fmonID).
14. The 5Gr-FBB notifies the 5Gr-SO of the enabled forecasting job ID (FjobID).

The 5Gr-VoMS will then periodically retrieve forecasted metric through Prometheus. Such values will be stored locally and can be accessed by the 5Gr-SO to include them in the decisions to be taken.

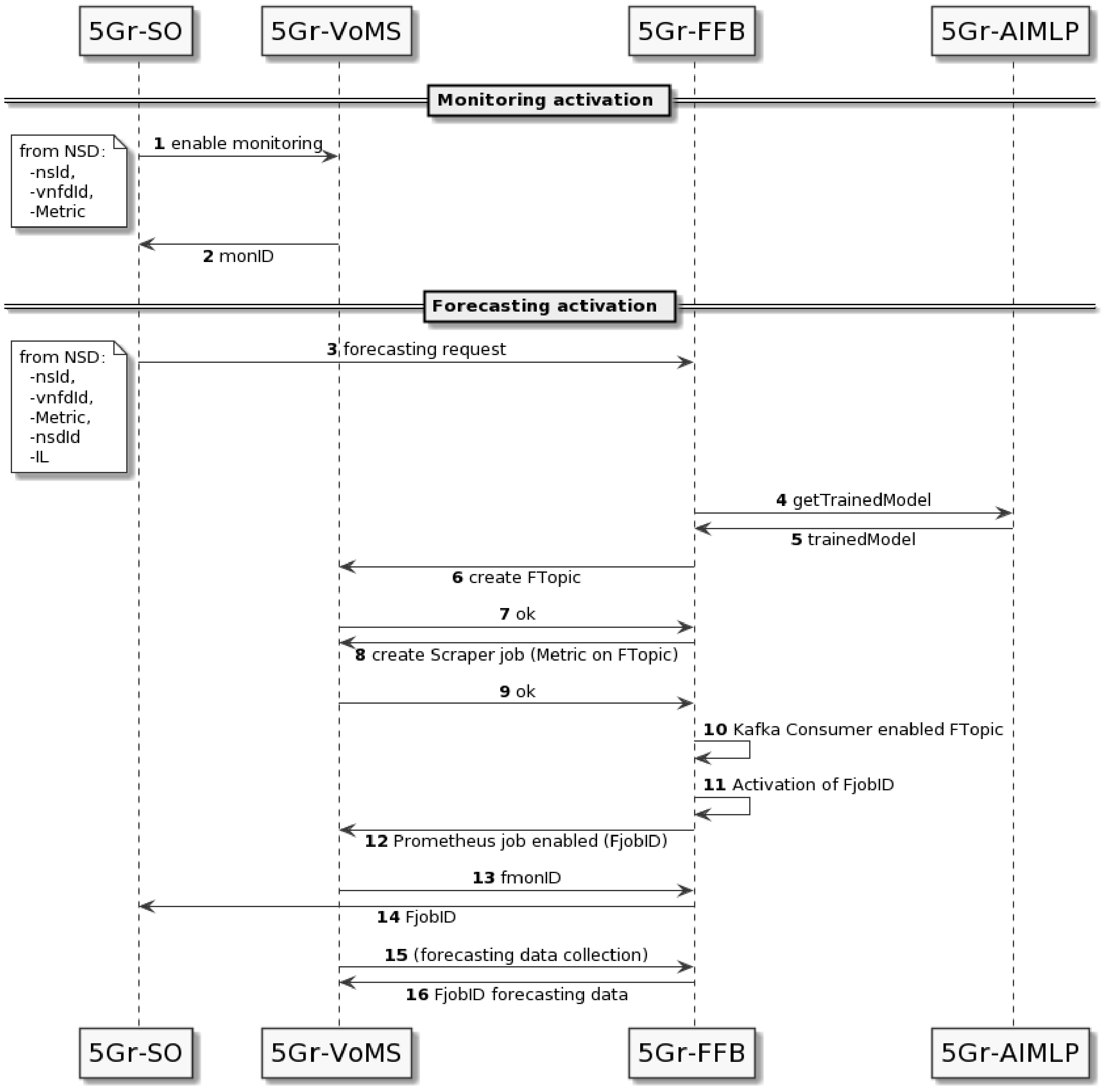
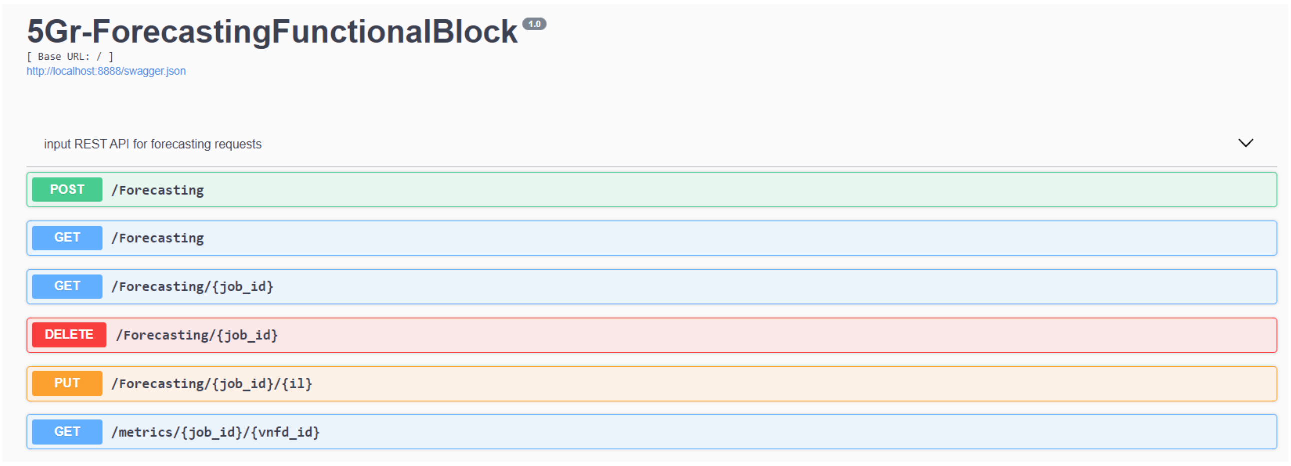

5Gr-SO
5Gr-VoMS
5Gr-FFB
5Gr-AIMLP
Monitoring activation
from NSD: -nsId, -vnfdId, -Metric
1 enable monitoring
2 monID
Forecasting activation
from NSD: -nsId, -vnfdId, -Metric, -nsdId -IL
3 forecasting request
4 getTrainedModel
5 trainedModel
6 create FTopic
7 ok
8 create Scraper job (Metric on FTopic)
9 ok
10 Kafka Consumer enabled FTopic
11 Activation of FjobID
12 Prometheus job enabled (FjobID)
13 fmonID
14 FjobID
15 (forecasting data collection)
16 FjobID forecasting data

FIGURE 15: 5GR-FFB EXAMPLE WORKFLOW

The source code of the 5Gr-FFB is available in the 5Growth GitHub repository [35].

6.1. Exposed Interfaces

Figure 16 shows a swagger view of the implemented APIs at the 5Gr Forecasting Functional Block. Following the details of the interfaces are shown.



5Gr-ForecastingFunctionalBlock 1.0

[Base URL: /]
http://localhost:8888/swagger.json

input REST API for forecasting requests

POST /Forecasting
GET /Forecasting
GET /Forecasting/{job_id}
DELETE /Forecasting/{job_id}
PUT /Forecasting/{job_id}/{il}
GET /metrics/{job_id}/{vnfd_id}

FIGURE 16: 5GR-FFB API

Forecasting Manager. The APIs exposed by this module enable the creation and the deletion of forecasting jobs. All the requests and responses follow the JSON format. The implemented methods are reported in Table 6.

TABLE 6: 5GR-FFB FORECASTING MANAGER APIS

POST /Forecasting/	Activate a new forecasting job. Input parameters: ○ nsId. ○ vnfdId ○ Performance metric ○ nsdId ○ IL Responses: ○ 200: The job_id is returned. ○ 404: Forecasting job not started.
PUT /Forecasting/(job_id)/(IL)	Updating the IL of the model running on job_id Responses: ○ 200: The IL of job_id is updated. ○ 404: Forecasting job not found.
GET /Forecasting/	Retrieve the list of active forecasting jobs. Responses: ○ *200: The list of the forecasting jobs ("job_id") is returned.*
GET /Forecasting/(job_id}	Retrieve the forecasting job details with the specified "job_id". Responses: ○ 200: The details related to the forecasting job with "job_id" are returned. ○ 404: Forecasting job not found.
DELETE /Forecasting/(job_id}	Stop the forecasting job identified with the job_id: ○ 200**:** The forecasting job with "*job_id*" is stopped. ○ 404: Forecasting job not found.

Prometheus Exporter. The module is used for forecasting data polling from the 5Gr-VoMS, using the job_id as input parameter, by means of the API detailed in Table 7.

TABLE 7: 5GR-FFB PROMETHEUS EXPORTER API

GET /metrics/(job_id)/(vnfd_id)	Retrieve the forecasting data computed by the forecasting job with job_id. Responses: ○ 200: The Prometheus-like output, according to the metric(s) forecasted. ○ 404: Forecasting job with the specified "*job_id*" not found.

7. 5Gr-VoMS implementation

The 5Growth Vertical-oriented Monitoring System (5Gr-VoMS) has been developed and extended during the whole project. The 5Gr-VoMS Release 1 (released at M12 and described in D2.2 [8]) was dynamically exporter (monitoring probe) installation oriented. The 5Gr-VoMS Release 2 (described in this deliverable) implements additional features such as gathering application metrics and collecting logs from VNFs. All the code of the 5Gr-VoMS Reference Implementation is open-source. In particular, the new components developed as part of the Config Managers have been released under Apache 2.0 license and they are available in the project git repository [22] as well as in the public GitHub repository [23]. The final version of the 5Gr-VoMS architecture is described in Figure 17.

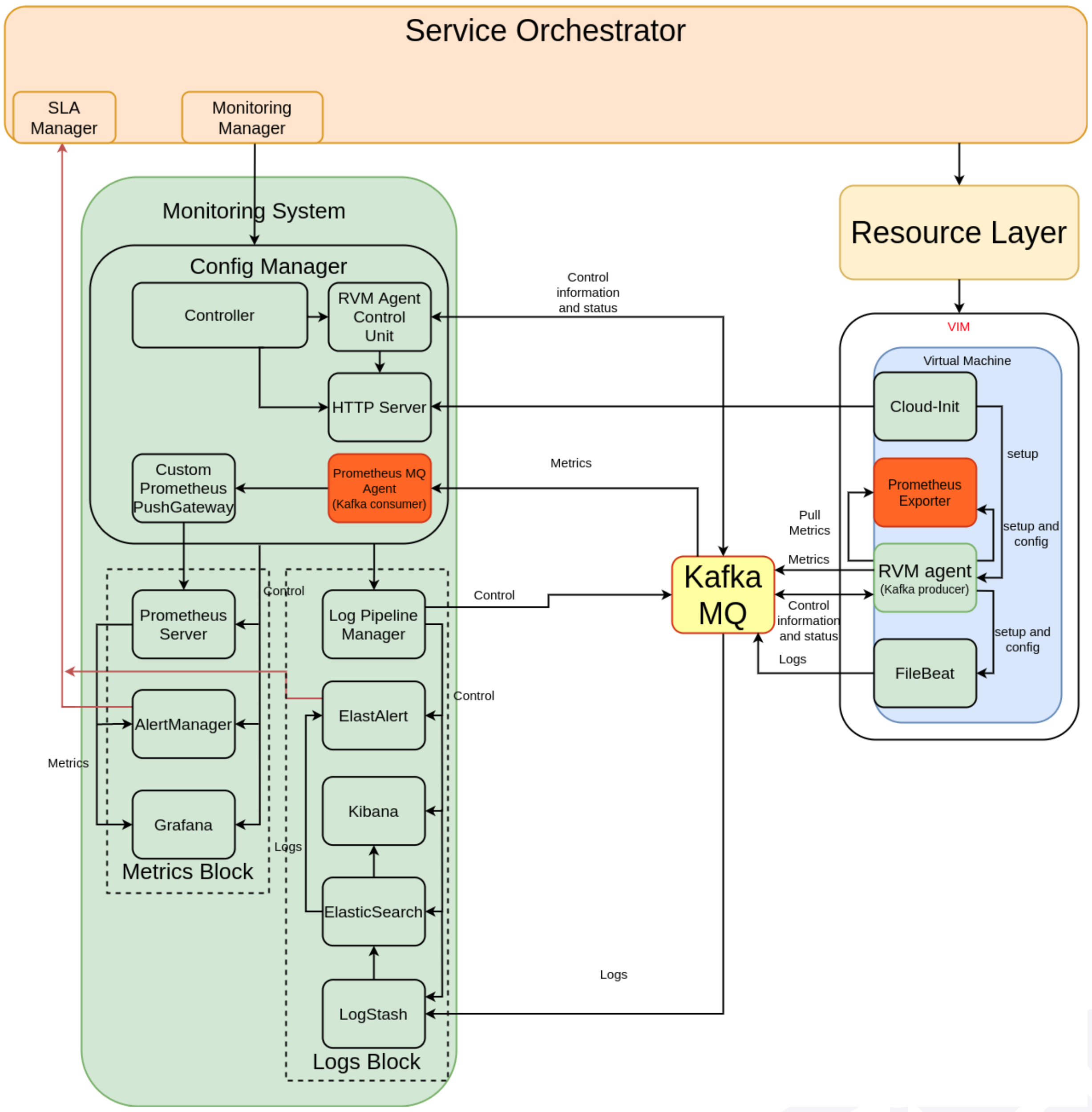


FIGURE 17: VERTICAL-ORIENTED MONITORING SYSTEM ARCHITECTURE

The 5Gr-SO interacts with the 5Gr-VoMS through the Monitoring Manager and the SLA Manager, which are part of the 5Gr-SO. The Monitoring Manager is the component in the 5Gr-SO translating requests for high-level monitoring jobs referred to NFV service instances into low-level requests related to the monitoring of resource-level parameters, describing which exact data should be collected by the 5Gr-VoMS.

The SLA Manager deals with SLA assurance. This component is keeping track of monitoring parameters like KPIs or real-time measurements mirroring the load of the resources. This allows indicating potential SLA breaches or anomalous behaviour that may lead to system under-performance. The SLA Manager interacts with the 5Gr-VoMS following a subscription-notification paradigm, to promptly react to any alert associated with the target monitoring data. In particular, the SLA Manager subscribes with the Config Manager, registering monitoring parameter queries (which may include arbitrarily complex expressions on many different time series) and threshold values for notifications. Such subscription is translated into the appropriate configuration of the Prometheus Alert Manager or ElastAlert so that whenever the given threshold is passed, a notification is sent to the SLA Manager, which will then trigger the needed reactions at the 5Gr-SO [25].

The 5Gr-VoMS is responsible for the next functionalities of the 5Growth stack:

- VNF logs and metrics collection;
- holding metrics and logs;
- tracking metrics and logs;
- providing metrics and logs to other systems (AI/ML platform, forecasting functional block, etc.);
- checking if the metrics data is corresponding to KPIs thresholds.

7.1. Description of building blocks

The 5Gr-VoMS consists of three blocks, Config Manager, Metrics Block and Logs Block, each of them detailed in the following subsections.

7.1.1. Config Manager

The **Config Manager** is the main block of the Monitoring System. This block includes: northbound interface (NBI), Controller, RVM Agent, RVM Agent Control Unit, HTTP Server, Custom Prometheus Push Gateway and Prometheus MQ Agent. The description of these modules is the following:

- The **Config Manager** has a **NBI** for receiving REST API requests from the 5Gr-SO, 5Gr-VS, and 5Gr-RL. These requests contain information on what metrics or logs should be tracked, how this information should be visualized, which unit should be informed if metrics are out of range. The requests can contain information on which monitoring probe should be installed at a specific VNF. NBI process requests for exposing metrics with other systems. One method of providing metrics is to publish the required metrics to a specific Kafka topic.

- The **Controller** is a unit of Config Manager that manages all parts of the 5Gr-VoMS. It receives requests from NBI, processes them, and prepares responses.
- The **RVM Agent** is a software component that was developed for the 5Gr-VoMS. It uses Kafka as a Message Queue for interaction with the 5Gr-VoMS. The RVM agent can execute bash and python scripts at a remote virtual machine. This gives the possibility to change the VNF configuration and install extra software. The RVM agent is used for monitoring probes (exporters) dynamic installation and configuration. Also, the RVM agent enables metrics to be pushed from the VNF side to the Prometheus server through the queue (Kafka) instead to be pulled by the Prometheus server from the VNF. The RVM agent can receive instruction about the creation of a "Prometheus collector" from the 5Gr-VoMS. The software entity "Prometheus collector" can be created inside the RVM agent, located on the VNF. Prometheus collector receives the configuration from the 5Gr-VoMS, it performs requests to installed Prometheus Exporter for metrics and publishes these metrics to Kafka Topic. The advantage of this RVM agent architecture, in comparison with existing configuration management systems like Puppet, Salt, or Ansible, is that it does not require a dedicated management network for interacting with the 5Gr-VoMS, as the RVM agent has control and data plane communication, performed through the Kafka broker.
- The **RVM Agent Control Unit** is responsible for the installation of RVM agents at remote VMs and interaction with them through Kafka Topics. The functions of this unit are:
  - RVM agent installation init-scripts generation and their distribution with the help of the HTTP server.
  - Sending control information to RVM agents through Kafka Topic and receiving responses from RVM agents.
  - Monitoring the current state of RVM agents by using a keepalive mechanism.
  - Executing bash and python scripts at RVM agent and receiving responses about execution status.
  - Installing monitoring probes through RVM agents.
- The **HTTP Server** is responsible for the distribution of the RVM agent installation init-scripts, RVM agent itself, and monitoring probes.
- **Custom Prometheus Push Gateway** is responsible for transferring the received metrics into the Prometheus server. Initially, the Open-Source Prometheus Push Gateway [25] was used as part of the 5Gr-VoMS but it has disadvantages. Native Open-Source Prometheus Push Gateway component has the same scrape interval for all metrics, scrapping interval cannot be changed individually for each metric, and it is unable to detect whether the Prometheus exporter is unavailable. To solve this limitation, Custom Prometheus Push Gateway has been developed.
- **Prometheus MQAgent** is a part of the config manager that gathers metrics from Kafka Topics and sends them to Custom Prometheus Push Gateway.

7.1.2. Metrics block

The **Metrics block** is used to process metrics and it includes Prometheus Server, Alert Manager and Grafana. The description of these modules is the following:

- **Prometheus Server** is an open-source platform that collects metrics from monitored targets. Prometheus data is stored in the form of Time Series Database (TSDB). Each metric has a name that is used for referencing and querying it. Prometheus stores data locally on disk, which helps for fast data storage and fast querying. Prometheus has PromQL - the query language used to create dashboards and alerts.
- **Alert Manager** processes Prometheus alerts. Alert Manager takes care of deduplicating, grouping, and routing alerts to the correct receiver. It also takes care of silencing and inhibition of alerts.
- **Grafana** provides graphs and visualizations of metrics. Grafana is multi-platform open-source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources. It is expandable through a plug-in system. End users can create complex monitoring dashboards using interactive query builders.

7.1.3. Logs block

The **Logs block**[2] includes units Log Pipeline Manager, ElastAlert, Kibana, Elasticsearch and Logstash. These blocks process logs, and they have been completely remodelled for this Release 2. Figure 18 presents the microservices architecture followed for these blocks, which are highlighted with two blue boxes:

[2] The code related to this block can be found in the 5Gr-VoMS repositories [23], in the folder called "log_monitoring_pipeline"

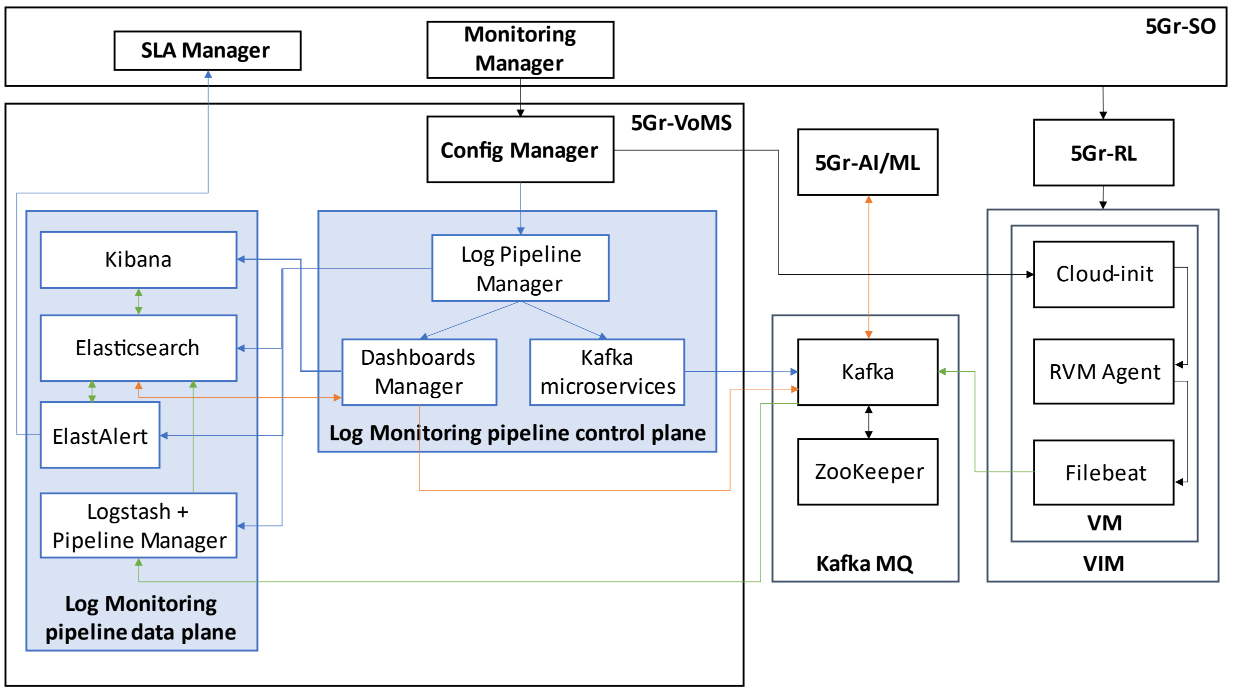


FIGURE 18: DETAILED ARCHITECTURE OF THE MONITORING PIPELINE

In this architecture, three different workflows can be observed: the data plane workflow (blue arrows), the control plane workflow (green arrows) and the interaction with the AI/ML platform (orange arrows). The first two workflows will be further explained in Section 7.2, and the last one will be described in Section 7.3. Regarding the components and the arrows with black colour that are not related to this pipeline, they have been included in the architecture for the sake of completeness.
The description of these modules is the following:

- **Log Pipeline Manager** is a new component in the 5Gr-VoMS platform, being in charge of managing the configuration lifecycle of the components related to the log monitoring process. This feature has been provided in this separate block, instead of including it directly into the Config Manager, due to the complex internal structure on which the Logs block is based on using microservices-based architecture for full control of the configuration lifecycle. As a result, the Log Pipeline Manager defines a NBI for receiving REST API requests from the Config Manager, related to the monitoring of logs for specific VNFs. Then, the Log Pipeline Manager triggers the interaction between microservices to properly configure all the components of the Logs block, based on the Elastic Stack. In particular, the Log Pipeline manager performs the following actions:
    - Create/delete the Kafka topics in the Kafka MQ in which the logs will be published by Filebeat. The Network Service ID (nsId) is used as a name for Kafka topic, so that all the logs monitored for a given Network Service will be published by Filebeat in the same Kafka topic. Although all the data is sent to the same Kafka topic, it can be extracted, from the message sent by Filebeat, the VNF that is generating a given message, or the log file that is being monitored. The interaction with Kafka is

implemented by using specific **Kafka microservices**, directly triggered by the Log Pipeline Manager.

  - Create/delete the Logstash pipelines that will subscribe to the corresponding Kafka topic, having a one-to-one mapping between the Kafka topic and the Logstash pipeline. The Logstash pipeline just defines a procedure to receive the data from Kafka, extract the fields of the source message sent by Filebeat (based on JSON) and forward the processed message to a given Elasticsearch index.
  - Create/delete the Elasticsearch indexes that will contain the data pushed by Logstash, making it available for visualization or searching operations. Again, there is a one-to-one mapping between the Logstash pipeline and the Elasticsearch index.
  - Create/delete the Kibana index pattern that allows the visualization of the data saved in a given Elasticsearch index. Again, the relationship between these two entities is one-to-one.
  - Create/delete the Kibana dashboard that displays the data saved in a given Elasticsearch index by using the corresponding Kibana index pattern. In principle, only one Kibana dashboard will be created for each Network Service. This action and the previous one is performed by the module called **Dashboards Manager**, a Java application that is also used by the Log Pipeline Manager to configure the dashboards in Kibana.
  - Create/delete alerts by interacting with ElastAlert, a component that monitors Elasticsearch in order to detect events that trigger the actions defined in the alerts, also interacting with the SLA Manager component.

- **Logstash** is the data collection pipeline tool. It collects data inputs and feeds them into Elasticsearch. It aggregates all types of data from different sources and makes it available for further use. In this case, Logstash receives data from Kafka. It also contains a logic called **Pipeline Manager**, which configures the Logstash pipelines based on the information sent by the Log Pipeline Manager.
- **Elasticsearch** allows storing, searching, and analyzing big volumes of data. It is mostly used in these applications as the underlying engine for implementing search task functionality. It has been adopted in search engine platforms for modern web and mobile applications. Apart from a quick search, the tool also offers complex analytics and many advanced features.
- **Kibana** is used for visualizing Elasticsearch documents and helps developers to have a quick insight into it. Kibana dashboard offers various interactive diagrams, geospatial data, and graphs to visualize complex queries. It can be used for search, view, and interact with data stored in Elasticsearch directories. Kibana helps to perform advanced data analysis and visualize the data in a variety of tables, charts, and maps.
- **ElastAlert** is a simple framework for alerting on anomalies, spikes, or other patterns of interest from data in Elasticsearch. It works by combining Elasticsearch with two types of components, rule types and alerts. Elasticsearch is periodically queried, and the data is passed to the rule type, which determines when a match is found. When a match occurs, it is given

to one or more alerts, which act based on the match. This is configured by a set of rules, each of which defines a query, a rule type, and a set of alerts.

Together with these three blocks, a **Kafka MQ** component is used as a bus for metrics, logs, and control information for RVM agents

7.2. System workflows

In the following, we explain the process how the 5Gr-SO creates a VM and manages the associated RVM agent. Note that the 5GR-SO analyses the NSD, calculates how many instances, and in which PoPs instances should be created by using the placement algorithm. Figure 19 shows the RVM agent installation and managing workflow.

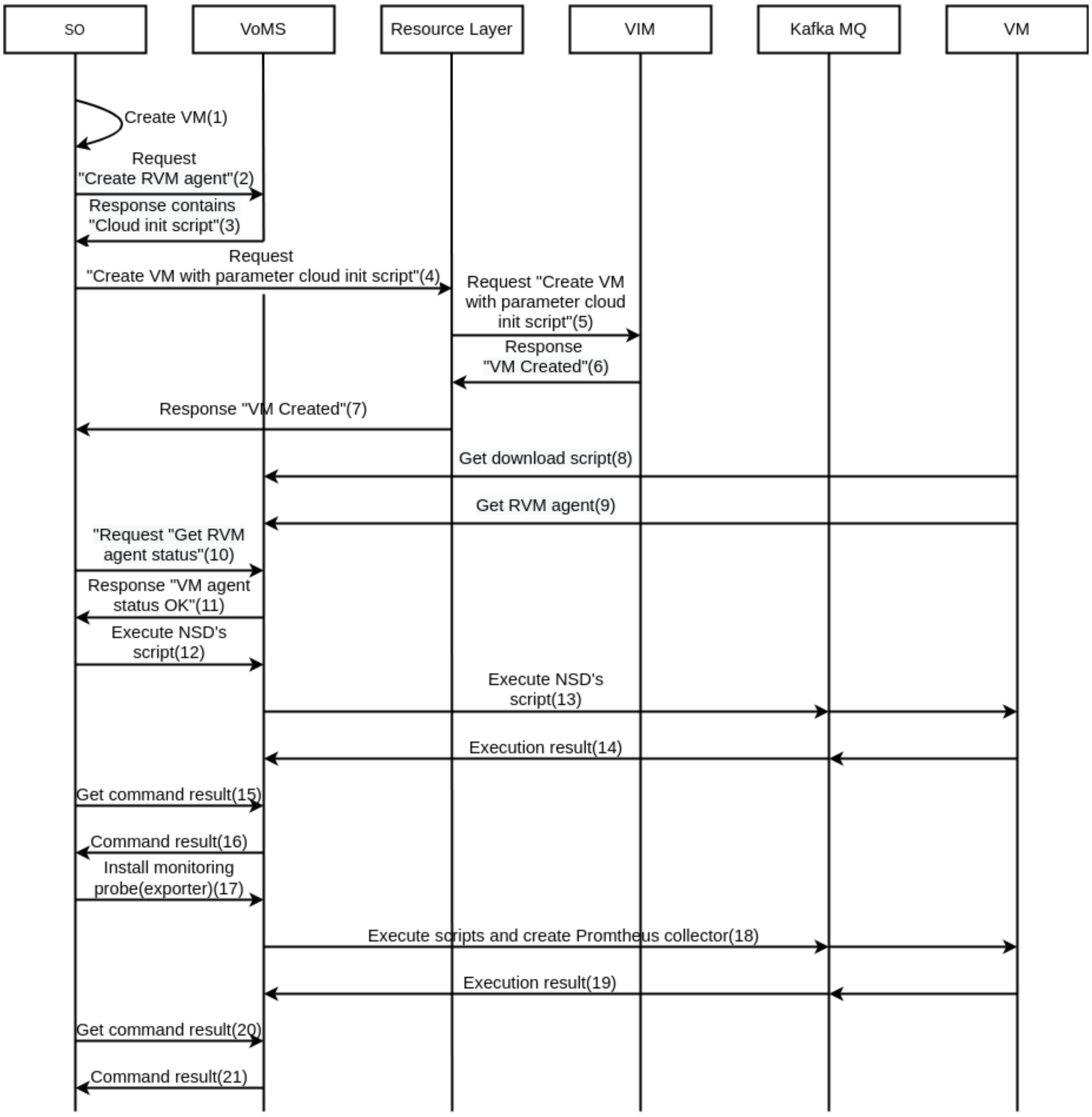

SO
VoMS
Resource Layer
VIM
Kafka MQ
VM
Create VM(1)
Request "Create RVM agent"(2)
Response contains "Cloud init script"(3)
Request "Create VM with parameter cloud init script"(4)
Request "Create VM with parameter cloud init script"(5)
Response "VM Created"(6)
Response "VM Created"(7)
Get download script(8)
Get RVM agent(9)
"Request "Get RVM agent status"(10)
Response "VM agent status OK"(11)
Execute NSD's script(12)
Execute NSD's script(13)
Execution result(14)
Get command result(15)
Command result(16)
Install monitoring probe(exporter)(17)
Execute scripts and create Promtheus collector(18)
Execution result(19)
Get command result(20)
Command result(21)

FIGURE 19: AN RVM AGENT INSTALLATION AND MANAGING WORKFLOW

An RVM agent installation and management workflow steps are the following:

1. The 5Gr-SO analyzes the NSD and takes the task “Create VM”.
2. The 5Gr-SO makes a request "Create RVM agent" to the 5Gr-VoMS.
3. The 5Gr-VoMS returns the response to the 5Gr-SO with a cloud-init script.
4. The 5Gr-SO makes a request "Create VM" with parameter cloud-init to the Resource Layer.
5. The Resource Layer sends the request to VIM (Virtual Infrastructure Manager)
6. The VIM creates a VM and returns a response "VM created" to the Resource Layer
7. Resource Layer transmits the response to the 5Gr-SO "VM created".
8. When Virtual Machine starts, it loads the script (from the HTTP server – a part of 5Gr-VoMS).

9. The script detects the type of the operating system, downloads (from the HTTP server – a part of 5Gr-VoMS), and starts the necessary agent.
10. The 5Gr-SO repeats the request "Get VM agent status" to the 5Gr-VoMS.
11. The 5Gr-SO repeats the previous request until it receives the response "VM agent status OK" from the 5Gr-VoMS. This time interval includes the following processes: get VM created, get the Operation system started, and get the VM agent started until it sends keep-alive messages to the 5Gr-VoMS.
12. The 5Gr-SO makes a "execute command" request to the VoMS. This request contains a script from NSD and RVM agent ID. The script from NSD is executed at the created VM.
13. The 5Gr-VoMS passes request "executes command" thorough Kafka broker to specific RVM agent on created VM.
14. The RVM agent executes the script on VM and returns the command execution status.
15. The 5Gr-SO repeats the request "Get command result" to the 5Gr-VoMS.
16. The 5Gr-SO repeats the previous request until it receives the response "command result" from the 5Gr-VoMS.
17. The 5Gr-SO makes a request "Install monitoring probe(exporter)" to the 5Gr-VoMS. This request contains an exporter identifier and RVM agent ID. The 5Gr-VoMS translates this request to a set of commands for exporter installation and request for the creation object "Prometheus collector" at the RVM agent side. Prometheus collector is a software unit which does requests metrics from Prometheus exporter and publishes received metrics to Kafka Topic.
18. The 5Gr-VoMS sends request "execute command" for exporter installation on VM and sends the request for creation object "Prometheus collector" at VM agent through Kafka Topic.
19. The RVM agent executes the script on VM and returns the command execution status.
20. The 5Gr-SO repeats the request "Get command result" to the 5Gr-VoMS.
21. The 5Gr-SO repeats the previous request until it receives the response "command result" from the 5Gr-VoMS.

Moreover, as the log monitoring procedure implemented in the 5Gr-VoMS has been completely updated from previous releases, it is worth presenting here the full workflow to be followed by the Logs block in order to introduce how the configuration lifecycle of all the components related to this stack is performed. This is divided in three different workflows:

1. The first workflow is related to the **control plane configuration**, in which all the entities of the Log Monitoring pipeline are configured according to the data provided by the Config Manager. This workflow is divided in four steps, which are executed in the order described in Figure 20: (1) the job configuration (messages from 1 to 5), which implies the creation of the Kafka topic, the Elasticsearch index and the Logstash pipeline related to the logs to be monitored for a given network service, (2) the dashboard configuration (messages 6 to 8'), which creates the Kibana dashboard, returning the URL to see the dashboard in an on-line basis, (3) the alert configuration (messages 9 to 11), creating the alert in ElastAlert, and (4) the log scraper configuration (messages 12 to 16), which enables the Log Scraper through

the Dashboards Manager entity. After this, the Log Scraper is continuously checking the data in Elasticsearch, and when retrieving new data, it is sent to Kafka.

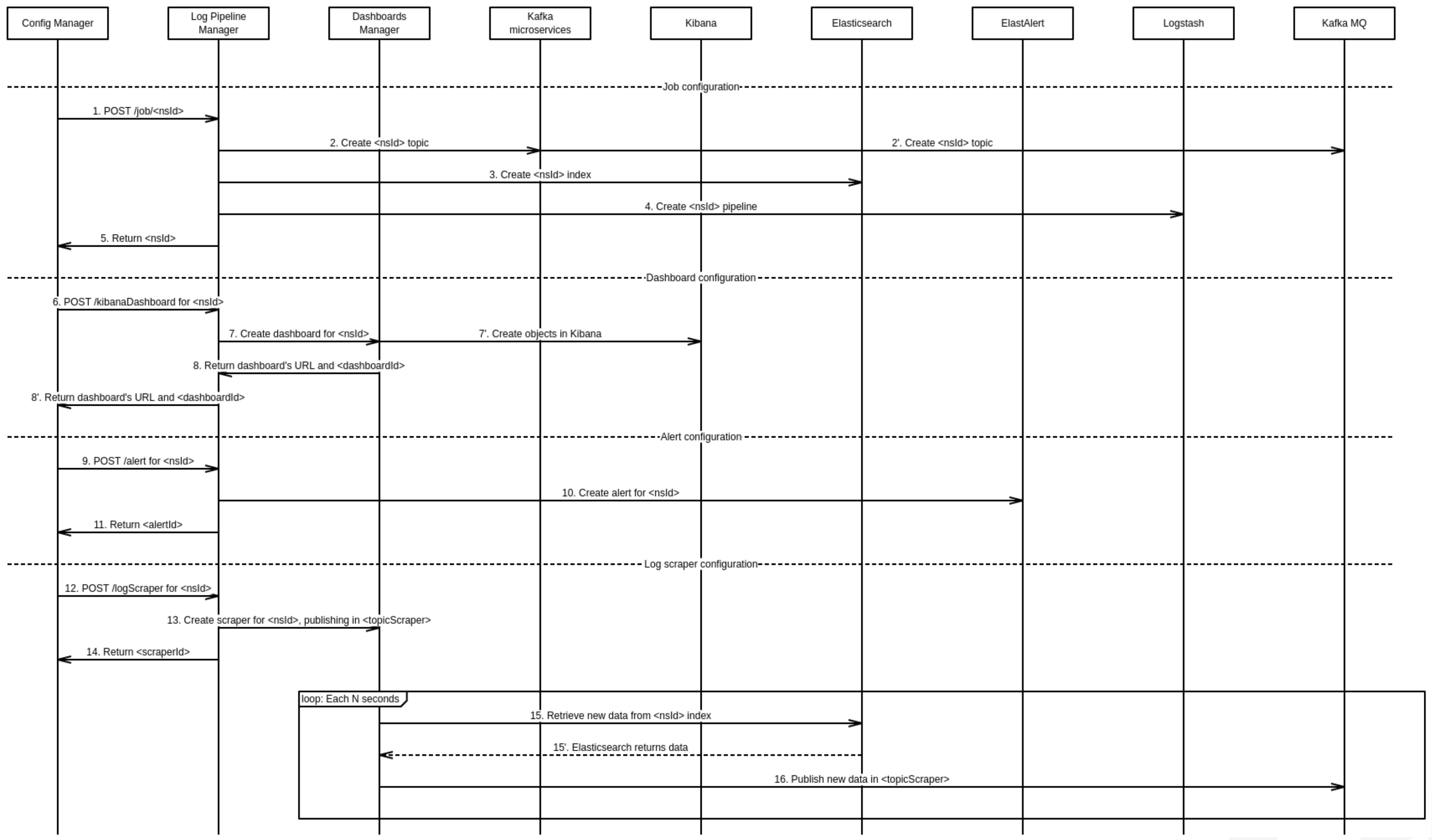

Config Manager
Log Pipeline Manager
Dashboards Manager
Kafka microservices
Kibana
Elasticsearch
ElastAlert
Logstash
Kafka MQ
Job configuration
1. POST /job/<nsId>
2. Create <nsId> topic
2'. Create <nsId> topic
3. Create <nsId> index
4. Create <nsId> pipeline
5. Return <nsId>
Dashboard configuration
6. POST /kibanaDashboard for <nsId>
7. Create dashboard for <nsId>
7'. Create objects in Kibana
8. Return dashboard's URL and <dashboardId>
8'. Return dashboard's URL and <dashboardId>
Alert configuration
9. POST /alert for <nsId>
10. Create alert for <nsId>
11. Return <alertId>
Log scraper configuration
12. POST /logScraper for <nsId>
13. Create scraper for <nsId>, publishing in <topicScraper>
14. Return <scraperId>
loop: Each N seconds
15. Retrieve new data from <nsId> index
15'. Elasticsearch returns data
16. Publish new data in <topicScraper>

FIGURE 20: LOG MONITORING PIPELINE. CONTROL PLANE - CONFIGURATION WORKFLOW

2. The **data plane workflow**, depicted in Figure 21, presents the path followed by the logs provided by Logstash, which are firstly sent to Kafka (message 1), then being delivered to Logstash (message 2), which saves them in Elasticsearch (message 3), then allowing Kibana to present them through the dashboard (message 4). If ElastAlert detects that the alert condition has been met, it notifies the SLA Manager consequently (message 5).

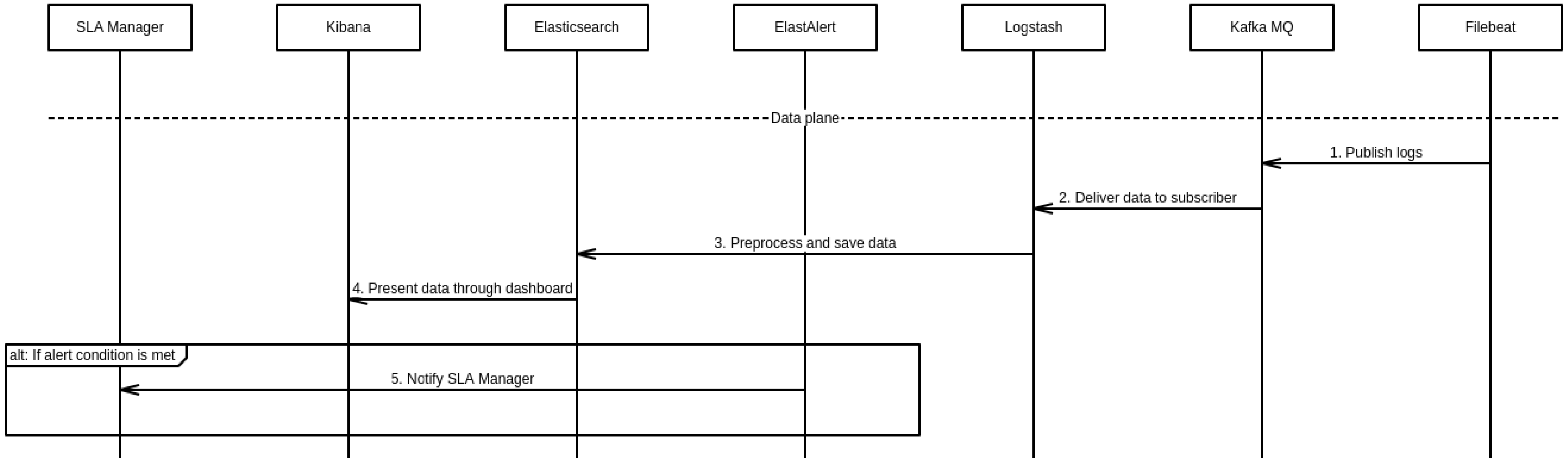

SLA Manager
Kibana
Elasticsearch
ElastAlert
Logstash
Kafka MQ
Filebeat
Data plane
1. Publish logs
2. Deliver data to subscriber
3. Preprocess and save data
4. Present data through dashboard
alt: If alert condition is met
5. Notify SLA Manager

FIGURE 21: LOG MONITORING PIPELINE. DATA PLANE WORKFLOW

3. Finally, when the execution is finished, the resources allocated in the control plane configuration phase can be removed by following the control plane withdrawal workflow, presented in Figure 22. In particular, it implies the opposite operations and order than the one presented in Figure 20: (1) the Log Scraper is removed (messages 1 to 3), then (2) the alert rule (messages 4 to 6), followed by (3) the Kibana dashboard (messages 7 to 9'), and finally, (4) the configuration related to the job (messages 10 to 14), related to Kafka, Logstash and Elasticsearch configuration.

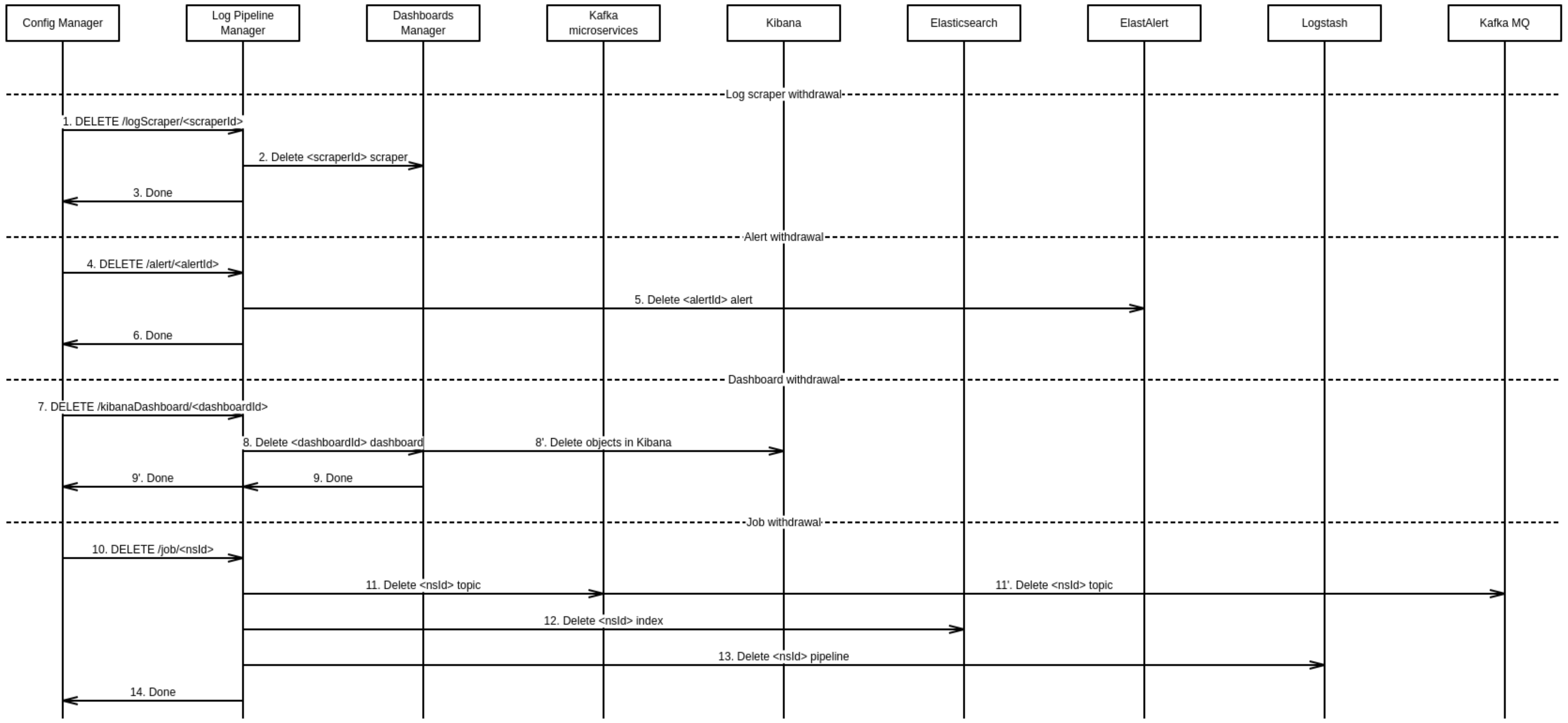


FIGURE 22: LOG MONITORING PIPELINE. CONTROL PLANE - WITHDRAWAL WORKFLOW

7.3. Value-added capabilities

The 5Gr-VoMS can provide metrics and logs to other systems by using native Prometheus and Elasticsearch interfaces or through the Kafka topics, which may be also used for other systems as a source of metrics data from the 5Gr-VoMS. According to requirements that were received from the 5Gr-AIMLP, the following software units were developed:

- **Prometheus Scraper:** it is a dynamically-created software unit as a request from the 5Gr-SO. The function of the Prometheus Scraper is to gather the last metrics from a Prometheus Server by using a Prometheus query as a periodic request from the 5Gr-SO request, and by publishing it as a Kafka Topic. The Kafka Topic identifier is also defined in the 5Gr-SO request. This Kafka Topic is tracked by the 5Gr-AIMLP, which uses this information for models training. The following figure shows the usage of the Prometheus Scraper. The Prometheus scrapper usage schema is provided in Figure 23.

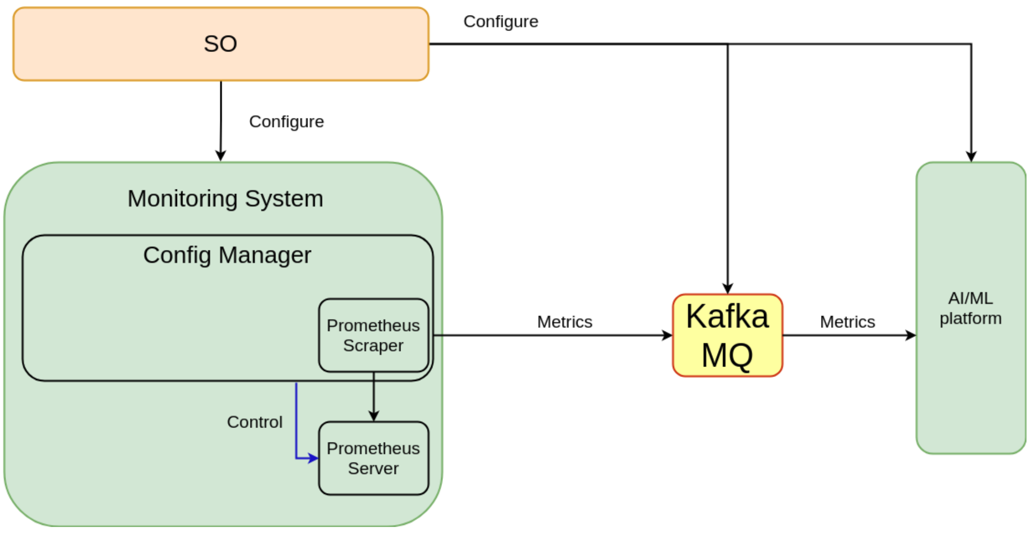


FIGURE 23: THE PROMETHEUS SCRAPER USAGE SCHEMA

- **Log Scraper:** this module oversees gathering the data saved in Elasticsearch for a specific log that is periodically monitored in each Network Service, and publishing it to a Kafka topic in the same way that is done by the Prometheus Scraper. The schema in which this component is based can be seen in the previous Figure 18, in the workflow depicted with the orange arrows. In this way, the Dashboards Manager retrieves the data from Elasticsearch, then creating a Kafka producer for publishing the data in the corresponding Kafka topic, from which the AI/ML platform will be listening to the data.
  The data model to be followed by the data sent to Kafka is as described in Figure 24, using a list to send the different data that matches the expression defined in the descriptors:

```
{"record":[{"agent":"<agent_name>","log_path":"<log_file_path>","host":"<host_n
ame>","message":"<log_message>","timestamp":"<log_timestamp>"}]}
```

FIGURE 24: LOG SCRAPER DATA MODEL

7.4. Final release features

Table 8 summarizes the functionalities implemented in Release 2. Release 1 focused on the implementation of a dynamic exporter (monitoring probe) installation. Conversely, Release 2 focuses on extending different functionalities of the 5Gr-VoMS, such as log collection from VNFs, collection of application metrics, monitoring probes process installation enhancement, or the integration of server-side probes.

TABLE 8: 5GR-VOMS RELEASES

Release 1	Release 2
RVM Agent: • Initially developed and added into the Monitoring Platform • Bash scripts execution • Keep-alive support • Prometheus collector's support • Configuration reliability (avoid loss after restart)	**RVM Agent:** • Python script run support • Filebeat installation and configuration
Configuration Manager: • REST API developed to support RVM agent installation • Functionality for generating RVM agent installation script • HTTP server integrated for storing RVM agent archives • Support bash script execution via RVM agent • Keep-alive support • Prometheus collectors support for RVM agent • REST API ELK stack support • Kibana auto dashboard instantiation extension for the configuration manager. • Automatic configuration of Kafka topics in the logstash pipelines to insert data in the Elastic	**Configuration Manager:** • Python scripts run supported via RVM agent • KPI measurement support • Custom Prometheus PushGateway • Filebeat installation and configuration
Service Orchestrator: • RVM agent installation • Remote command execution through RVM agent • Prometheus collectors creation/deletion	**Service Orchestrator:** • Exporters (monitoring probes) installation
Network Service Descriptor: • Definition of Config manager additional functions	
Log Monitoring Architecture: • ELK stack preliminary integration • Filebeat probe with Kafka • Logstash log parser from kafka	**Log Monitoring Architecture:** • Integration of the Log Pipeline Manager entity, together with the Dashboards Manager and the Kafka microservices

• Kibana visualization preliminary tool integration • ElastAlert preliminary integration • Two new network monitoring probes, IPFIX Collector and Packetbeat.	• Integration of Log Scraping capabilities. • Update of all the components of the ELK Stack to fit in the new architecture • Deployment of the whole stack with Kubernetes.
	Additional components: • Native and 3rd-party client-side monitoring probes • server-side probe • Prometheus exporter for collecting customer metrics (PECCM)

The relationship between the 5Gr-VoMS and the project innovations described in D2.3 [1] and looks like as 5Gr-VoMS provides provide metrics and logs from VNFs and infrastructure for the following innovations:

- Innovation 4 (I4): Control and Management. Control-loops stability
- Innovation 5 (I5): Control and Management. AI/ML Support
- Innovation 10 (I10): Forecasting and inference

7.4.1. RVM Agent

In Release 2, the RVM agent has been enhanced to execute Python scripts at VNFs. In this way, NSDs can include, not only bash scripts, but also python scripts. These scripts are used for initial configuration and changes of the VNFs configuration during scaling operation. In addition, the RVM agent can install and configure Filebeat to gather logs from VNFs.

7.4.2. Configuration Manager

In Release 2, the functionality of the Configuration Manager was extended in the following way:

- Custom Prometheus Push Gateway was added - it supports Kafka Broker as a source of metrics and gives the possibility for the Prometheus server to use storage space for metrics more efficiently in comparison with native open-source Prometheus push gateway [1].
- The API and functionality of the Configuration Manager were extended for executing Python scripts at VNFs.

The Configuration Manager has tools for measuring the Core KPIs defined in D4.1 [33]:

- CKPI-1 End-to-end Latency,
- CKPI-2 Packet Loss,
- CKPI-3 Guaranteed Data Rate,
- CKPI-5 Availability,
- CKPI-6 Slice Creation Time,
- CKPI-9 Jitter.

7.4.3. Service Orchestrator

In Release 2, the specification of NSD was extended by defining the type of monitoring probe hence the 5Gr-SO got the possibility to parse this new parameter and send the request to the 5Gr-VoMS for installation of monitoring probes at VNFs.

7.4.4. Log Monitoring Architecture

This architecture, related to the Logs block described in Section 7.1, has been completely re-designed for Release 2, compared to Release 1, in which only the ELK Stack with limited features for its configuration was provided.

As a result, Release 2 covers the full configuration of the ELK Stack with the inclusion of a set of control plane entities, such as the Log Pipeline Manager, the Dashboards Manager and the Kafka micro-services, enabling the automation of the configuration of monitoring processes related to log monitoring.

7.4.5. Additional components

In Release 2, additional components were developed for application metrics collection and evaluation (the detailed concept can be found in D2.3 [1]). The additional components are the following:

- Native and 3rd-party client-side monitoring probes - are entities that collect application metrics at the client-side and send these metrics to Prometheus exporter for collecting customer metrics (PECCM).
- Prometheus exporter (PECCM) - this element collects application metrics and passes them to the Prometheus server.
- Server-side monitoring probes - are based on BlackBox Prometheus exporter [26]. The Server-side probe, in a nutshell, is a dedicated VNF that does requests to the virtual server as a client and reports to the monitoring platform about service status. These probes give the possibility to evaluate CKPI-5 Availability.

8. 5Growth CI/CD

Continuous Integration/Continuous Development (CI/CD) is a software development concept and a set of practices targeted to automate and simplify the software development and deployment procedures. The 5Growth CI/CD consists of two parts – Continuous Integration (CI) responsible for automated testing of a new code, and Continuous Deployment (CD) responsible for simple, repeatable, and reliable deployment. 5Growth CI/CD is built around and uses repositories, compute resources and network infrastructure of the project and it is integrated into it. For more details regarding 5Growth CI/CD, its architecture, flows and infrastructure please refer to D2.3 [1] section 3.3.2.

The high-level CI/CD representation is described on Figure 25.

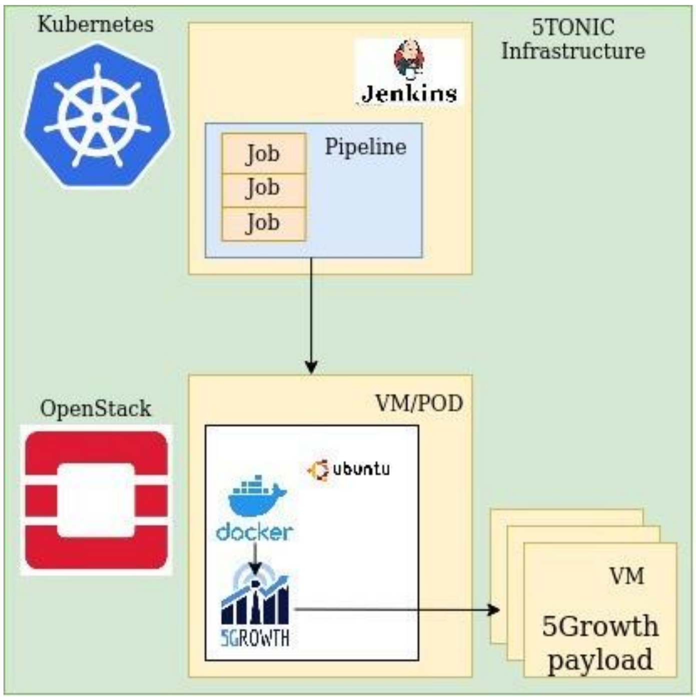


FIGURE 25: 5GROWTH CI/CD

5Growth CI/CD cycle allows to reduce time and resources, required for 5Growth platform deployment. 5Growth evaluation in detail can be found in section 4 of D2.3 [1]. CD is also able to package and ship the entire 5Growth platform as a whole.

The 5Growth CI/CD consists of the following components:

- 5Growth git repositories are storage for the project codebase and a version control system, that allows project developers to contribute and collaborate. Jenkins continuously monitors 5Growth repositories for code changes and when it happens, the pipeline is being triggered.
- CI/CD tool (Jenkins) has a set of preconfigured Pipelines, providing automation for stages in a CI/CD workflow like environment preparation, verification, containerization, etc.
- Infrastructure orchestrator instantiates environment according to blueprint, provided by a Pipeline.

- Series of tests examine the deployed platform and generate a report. The pipeline analyses test reports and make a decision. Successfully verified code changes are being built, containerized, and stored for next usage, while failed changes are reported to the developers.
- Infrastructure itself, OpenStack or Kubernetes, responsible for hosting test, demo, and development environments, CI/CD infrastructure, and 5Growth platform payload.

8.1. Final release features

Table 9 summarizes 5Growth CI/CD release 2 features. Release 1 was concentrated around creating core of CI/CD, main Jobs and Pipeline's skeletons. Release 2 is mostly concentrating on options extension and deepening functionality, like extending test coverage, parametrizing deployment, extending infrastructure interactions.

TABLE 9: 5GROWTH CI/CD RELEASES

Release 1	Release 2	Notes
Containerization: • 5Growth component containerization (Docker)	**Containerization**: • Containerization extension to Kubernetes deployment	
Environment management: • Reworked LCM pipeline for environment management	**Environment management:** • Standardized developer's environment (minikube on top of devstack) • Infrastructure as Code approach for describing environments (develop, testbed, demo) • Automation for Kubernetes deployment	
Quality Assurance (QA): • Automated testing pipeline	**QA:** • Extend testing	

8.1.1. Containerization

In Release 2, the CI/CD framework was extended with advanced containerization capabilities. The previously containerized platform is now adapted for Kubernetes deployment. This enabled significant reduction of deployment time, but required infrastructure extensions and deployment process adaptation. The New continuous integration pipelines are able to store validated images in a docker registry. The 5TONIC lab environment, detailed in D3.2 [34], was extended with a docker registry and Kubernetes cluster. There were new deployment pipelines developed for Kubernetes. The deployment process for Kubernetes requires a different approach for the configuration of 5Growth platform components and its interactions.

8.1.2. Environment management

Stable and repeatable environment is crucial for development and deployment processes automation. Environment repeatability is achieved via Infrastructure as Code (IaC) approach. IaC is developed in testbeds for platform code validation (at CI stage), for platform deployment (at CD stage) and in specially designed environment for development that may be automatically deployed on developer's workstation. Moreover, Kubernetes support into CI/CD workflows was introduced using pipelines for automated Kubernetes deployment both on 5Tonic infrastructure and developer's workstations.

Due to IaC approach and extended automation every environment created by any CI/CD pipeline is similar to each other and to developer's environment.

8.1.3. Quality Assurance (QA)

In the code validation phase, automated tests check code functionality, regression and integration perspectives and deployment readiness.

9. 5Growth end-to-end platform repositories

In Table 10 we summarize the 5Growth platform repositories and their licenses.

TABLE 10: 5GROWTH END-TO-END SERVICE PLATFORM REPOSITORIES

Name	URL	License
5Gr-VS	https://github.com/5growth/5gr-vs	Apache 2.0
5Gr-SO	https://github.com/5growth/5gr-so	Apache 2.0
5Gr-RL	https://github.com/5growth/5gr-rl	GPL Apache 2.0 (ONOS WIM plugin for Openflow)
5Gr-AIMLP	https://github.com/5growth/aimlp	Apache 2.0
5Gr-VoMS	https://github.com/5growth/5gr-mon	Apache 2.0
5Gr-FFB	https://github.com/5growth/5gr-FFB	Apache 2.0
5Gr-CI/CD	https://github.com/5growth/5gr-ci	Apache 2.0

10. References

[1] 5Growth D2.3 – "Final Design and Evaluation of the innovations of the 5G End-to-End Service Platform", May 2021

[2] 5Gr-SO GitHub repository available at the repository: https://github.com/5growth/5gr-so

[3] OSM Python client: https://osm.etsi.org/wikipub/index.php/OSM_client

[4] Cloudify official website: https://cloudify.co/

[5] ETSI GS NFV-IFA 005, "Network Functions Virtualisation (NFV); Management and Orchestration; Or-Vi reference point - Interface and Information Model Specification", v2.3.1, August 2017

[6] Open Source MANO website: https://osm.etsi.org/

[7] ETSI GR NFV-IFA 022, Management and Orchestration; Report on Management and Connectivity for Multi-Site Services", v0.8.2, 2018

[8] 5Growth D2.2 – "Initial implementation of 5G End-to-End Service Platform", May 2020

[9] 5Growth D2.1 – "Initial Design of 5G End-to-End Service Platform", December 2019

[10] 5Gr-RL ONOS plugin repository https://github.com/5growth/5gr-rl/tree/master/rl/plugins/WIM/ONOS-OpenFlow-Slicing/onos-plugin

[11] Carmelo Cascone, "ONOS support for P4". MTS, ONF, December 2018: https://opennetworking.org/wp-content/uploads/2018/12/ONOS-support-for-P4.pdf

[12] 5Gr-RL Kubernetes plugin repository https://github.com/5growth/5gr-rl/tree/master/rl/plugins/VIM/kubernetes

[13] Kubernetes Multus plugin website https://github.com/k8snetworkplumbingwg/multus-cni

[14] Nextworks slicer-catalogue repository, https://github.com/nextworks-it/slicer-catalogue

[15] Nextworks slicer-identity repository, https://github.com/nextworks-it/slicer-identity-mgmt

[16] 5Gr-VS GitHub repository available at https://github.com/5growth/5gr-vs

[17] 5Gr-RL GitHub repository available at https://github.com/5growth/5gr-rl

[18] 5Gr-AIMLP GitHub repository available at https://github.com/5growth/aimlp

[19] 5Growth D3.3 – "First version of software implementation for the platform", November 2020

[20] 5Gr-SO project repository available at the repository: https://github.com/5growth/5gr-so

[21] "ETSI GR NFV-IFA 022, Management and Orchestration; Report on Management and Connectivity for Multi-Site Services", v0.8.2, 2018

[22] 5Gr-VoMS project repository available at the repository: https://5growth.eu/git/5growth.5gr-mon

[23] 5Gr-VoMS project repository available at the repository: https://github.com/5growth/5gr-mon

[24] Official Prometheus Pushgateway https://github.com/prometheus/pushgateway

[25] 5G-TRANSFORMER, "Final design and implementation report on service orchestration, federation, and monitoring platform", Deliverable 5G-TRANSFORMER D4.3, May 2019.

[26] BlackBox Prometheus Exporter https://github.com/prometheus/blackbox_exporter

[27] 5Gr-RL Resource Allocation Server repository, https://github.com/5growth/5gr-rl/blob/master/rl/RL_RA_Server_R2/API/5gr_rl_ra_api_v1.2.0.json

[28] Swagger Editor available at https://editor.swagger.io/

[29] ETSI GS MEC 010-2, "Multi-access Edge Computing (MEC); MEC Management; Part 2: Application lifecycle, rules and requirements management", v2.1.1, 2019

[30] Guava: Google core Libraries for Java available at https://github.com/google/guava

[31] MySQL database software available at https://www.mysql.com/it/

[32] 5Growth D3.4 – "Plan for pilot deployments", November 2020

[33] 5Growth D4.1 – "Service and Core KPI Specification", May 2020

[34] 5Growth D3.2 – "Specification of ICT17 in-house deployment", April 2020

[35] 5Gr-FFB GitHub repository, https://github.com/5growth/5gr-FFB

[36] 5Growth D5.4 – "Report on WP5 Progress and Update of CoDEP", May 2021

[37] 5Growth contribution to ONOS https://gerrit.onosproject.org/c/onos/+/23479