5G ExPerimentation Infrastructure hosting Cloud-nativE Network Applications for public proTection and disaster RElief

Innovation Action – ICT-41-2020 - 5G PPP – 5G
Innovations for verticals with third party services

D4.5: 5G-EPICENTRE experimentation facility final version

Delivery date: January 2024

Dissemination level: Public

Project Title:	**5G-EPICENTRE - 5G ExPerimentation Infrastructure hosting Cloud-nativE Network Applications for public proTection and disaster RElief**
Duration:	**1 January 2021 – 31 December 2023**
Project URL	https://www.5gepicentre.eu/

This project has received funding from the European Union's Horizon 2020 Innovation Action programme under Grant Agreement No 101016521.

www.5gepicentre.eu

Document Information

Deliverable	D4.5: 5G-EPICENTRE experimentation facility final version
Work Package	WP4: Platform integration and VNF development
Task(s)	T4.3: Cross-testbed federation & synchronization T4.4: End-to-end platform integration activities
Type	Report
Dissemination Level	Public
Due Date	M30, June 30, 2023
Submission Date	M31, July 31, 2023 M37, January 31, 2024 (Revision)
Document Lead	Konstantinos Apostolakis (FORTH) Manuel Requena Esteso (CTTC)
Contributors	Stefania Stamou (FORTH) Hamzeh Khalili (CTTC) Fatemehsadat Tabatabaeimehr (CTTC) Josep Mangues-Bafalluy (CTTC) Pedro Tomás (ONE) Luis Cordeiro (ONE) André Gomes (ONE) Jorge Marquez Ortega (UMA) Almudena Díaz Zayas (UMA) Apostolos Siokis (IQU) Luigi D'Addona (IST) Anna Maria Spagnolo (IST) Sozos Karageorgiou (EBOS) Daniel del Teso (NEM) Ivan González (NEM)
Internal Review	Daniel del Teso (NEM) Pedro Tomás (ONE)

Document history

Version	Date	Changes	Contributor(s)
V0.1	19/06/2023	Initial deliverable structure.	Konstantinos Apostolakis (FORTH)
V0.2	11/07/2023	50% of deliverable content.	Konstantinos Apostolakis (FORTH)
V0.3	14/07/2023	Assimilation of input on HSPF/Network Intrusion & Detection platform integration details (Section 4.5.2).	Pedro Tomás (ONE) Luis Cordeiro (ONE) André Gomes (ONE)
V0.4	17/07/2023	Assimilation of input on Back-end (Coordinator) and Infrastructure layer (5G Traffic simulation) component integration details (Sections 4.2.2 and 4.3.1).	Jorge Marquez Ortega (UMA) Almudena Díaz Zayas (UMA)
V0.5	18/07/2023	Assimilation of input on Network Service Repository APIs (IQU, Section 4.2.1); Analytics pipeline APIs (IST, Sections 4.4.2 and 4.5.3); and Northbound Configuration Dashboard and Configurator Network Application (EBOS, Section 4.5.1).	Apostolos Siokis (IQU) Anna Maria Spagnolo (IST) Luigi D'Addona (IST) Sozos Karageorgiou (EBOS)
V0.9	19/07/2023	Consolidation of final inputs, preparation of internal review version.	Konstantinos Apostolakis (FORTH)
V1.0	20/07/2023	Assimilation of input on Cross-testbed federation & synchronization approach (CTTC, Section 3) and Analytics pipeline APIs (IST, Sections 4.4.2 and 4.5.3).	Manuel Requena Esteso (CTTC) Hamzeh Khalili (CTTC) Fatemehsadat Tabatabaeimehr (CTTC) Josep Mangues-Bafalluy (CTTC) Anna Maria Spagnolo (IST) Luigi D'Addona (IST)
V1.1	27/07/2023	1st post internal review version with suggested revisions by ONE, and corrections by UMA and CTTC.	Pedro Tomás (ONE) Jorge Marquez Ortega (UMA) Manuel Requena Esteso (CTTC)
V1.2	28/07/2023	2nd post internal review version by NEM, with suggested revisions and corrections by IST, EBOS and CTTC. Version preparation for final quality review (CTTC).	Daniel del Teso (NEM) Ivan González (NEM) Anna Maria Spagnolo (IST) Sozos Karageorgiou (EBOS) Fatemehsadat Tabatabaeimehr (CTTC) Manuel Requena Esteso (CTTC)

V2.0	30/07/2023	Final revisions after quality review, including formatting and proof-reading - Final version for submission.	Konstantinos Apostolakis (FORTH)
V2.1	29/01/2024	Implemented revisions after project periodic review – marked changes for internal review.	Konstantinos Apostolakis (FORTH)
V2.2	29/01/2024	Updated version with suggested revisions by the appointed internal reviewers.	Daniel del Teso (NEM) Ivan Gónzalez (NEM) André Gomes (ONE)
V2.5	30/01/2024	Formatting and proof-reading	Konstantinos Apostolakis (FORTH)
V3.0	31/01/2024	Final revised version for submission	Konstantinos Apostolakis (FORTH)

Project Partners

Logo	Partner	Country	Short name
AIRBUS	AIRBUS DS SLC	France	**ADS**
NOVA	NOVA TELECOMMUNICATIONS SINGLE MEMBER S.A.	Greece	**NOVA**
altice labs	Altice Labs SA	Portugal	**ALB**
Fraunhofer HHI	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.	Germany	**HHI**
FORTH INSTITUTE OF COMPUTER SCIENCE	Foundation for Research and Technology Hellas	Greece	**FORTH**
UNIVERSIDAD DE MÁLAGA	Universidad de Malaga	Spain	**UMA**
CTTC	Centre Tecnològic de Telecomunicacions de Catalunya	Spain	**CTTC**
istella	Istella SpA	Italy	**IST**
ONE SOURCE Consultoria Informática Lda.	One Source Consultoria Informatica LDA	Portugal	**ONE**
i q	Iquadrat Informatica SL	Spain	**IQU**
nemergent solutions	Nemergent Solutions S.L.	Spain	**NEM**
eBOS Technologies	EBOS Technologies Limited	Cyprus	**EBOS**
athonet	*Athonet SRL* **(Participation ended)**	*Italy*	*ATH*
RedZinc	RedZinc Services Limited	Ireland	**RZ**
Opto Precision	OptoPrecision GmbH	Germany	**OPTO**
YOUBIQUO	Youbiquo SRL	Italy	**YBQ**
ORama VR	ORamaVR SA	Switzerland	**ORAMA**
Hewlett Packard Enterprise	Hewlett-Packard Italiana Srl	Italy	**HPE**

List of abbreviations

Abbreviation	Definition
5QI	5G QoS Indicator
AI	Artificial Intelligence
API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CPU	Central Processing Unit
DevOps	Development and Operations
GA	Grant Agreement
HSPF	Holistic Security and Privacy Framework
JSON	JavaScript Object Notation
K8s	Kubernetes
KPI	Key Performance Indicator
MANO	Management & Orchestration
MQTT	Message Queuing Telemetry Transport
NCD	Northbound Configuration Dashboard
NFV(I)	Network Functions Virtualisation (Infrastructure)
PaaS	Platform-as-a-Service
QoS	Quality of Service
TI	Testbed Instance
UC	Use Case
UE	User Equipment

UI	User Interface
VPN	Virtual Private Network
WP	Work Package

Executive summary

The present deliverable constitutes the updated version of D4.4 *"5G-EPICENTRE experimentation facility preliminary version"* and is the accompanying report to the integrated 5G-EPICENTRE experimentation facility, coinciding with project Milestone MS7 *"Platform tools and content ready for third-party user evaluation"* (due in M30). The document presents the thorough technical integration documentation produced in Task T4.4 *"End-to-end platform integration activities"*, which contains the developed application programming interfaces that support the information flows within the final version of the integrated 5G-EPICENTRE platform. It further compiles the final, detailed report on the implementation of the ambitious Karmada cross-testbed federation framework, integrated as part of activities in Task T4.3 *"Cross-testbed federation & synchronization"*.

This deliverable hence presents how the constituent components of an online 5G experiment e-ordering platform come together, enabling both first- (*i.e.*, Use Case owners), and third-party experimenters to upload their vertical application software (and make it available to the testbed administrators for the setup stage of the experimentation process), as well as book an experiment in one of the testbeds of the project, in order to execute the experiment and/or the respective use case, and to monitor and analyse the outcomes in real-time.

In this report we review the implementation perspective of the organizational structure of this 5G-EPICENTRE integrated system, as elaborated in D1.4 *"Experimentation requirements and architecture specification final version"*. The four independent testbed facilities (each characterized by different 5G features and exposed capabilities) are federated under a Karmada control plane architecture, each accommodating a K8s-based management architecture, to orchestrate and manage deployment and operation of containerized PPDR applications. A centralized, cloud-native experimentation platform is deployed on top of this federation, enabling access to each of the underlying testbeds' resources. It is comprised of a centralized Back-end layer architecture), in charge of orchestrating the commands necessary to facilitate the deployment of the vertical application under the specifically desired test conditions; and a Front-end layer, responsible for exposing a Platform-to-Consumer web application for experiment ordering. This system is thereby capable of facilitating the interactions between the two foreseen actors of the platform, *i.e.*, the vertical experimenters and the testbed administrators.

Table of Contents

List of Figures ... 10
List of Tables ... 11
1 Introduction ... 12
1.1 Mapping of project's outputs ... 13
1.2 Updates since the initial deliverable version ... 15
2 Platform integration overview ... 16
2.1 Integration environment ... 16
2.2 CI/CD process ... 16
2.3 Gitlab repository ... 18
3 Cross-testbed federation & synchronization approach ... 19
3.1 Initial approach ... 19
3.2 Developments ... 20
3.2.1 Cluster affinity plugin ... 21
3.2.2 Latency-aware plugin ... 21
3.3 Outcomes ... 22
3.4 Service deployment process in the federation ... 22
3.4.1 Accessing Karmada and the Kubernetes clusters ... 23
3.4.2 Setting up the federation ... 23
3.4.3 Deploying services through Karmada ... 24
4 Integrated 5G-EPICENTRE experimentation facility ... 26
4.1 Delegating vertical application components to the testbed operator ... 27
4.1.1 5G-EPICENTRE Portal - Network Service Repository integration ... 28
4.2 Scheduling an experiment ... 29
4.2.1 5G-EPICENTRE Portal – Network Service Repository ... 29
4.2.2 5G-EPICENTRE Portal – Experiment Coordinator ... 31
4.2.3 Experiment Coordinator – Network Service Repository ... 32
4.2.4 Experiment Coordinator – Karmada API Server ... 32
4.3 Experiment deployment ... 32
4.3.1 Experiment Coordinator – 5G Traffic Simulation Manager ... 33
4.3.2 Experiment Coordinator – Publisher ... 34
4.3.3 5G Traffic Simulation Manager - Remote iPerf Agent ... 37
4.4 Experiment monitoring during execution ... 38
4.4.1 5G-EPICENTRE Portal – Experiment Coordinator ... 39
4.4.2 Analytics Engine – Analytics Aggregator – 5G-EPICENTRE Portal ... 40
4.5 Platform Network Applications integration ... 43
4.5.1 Configurator and Network Configuration Dashboard ... 44
4.5.2 Network Intrusion & Detection ... 44
4.5.3 Analytics ... 45
5 Conclusions ... 48
References ... 49

List of Figures

Figure 1: The 5G-EPICENTRE CI/CD process 17

Figure 2: Multi-cluster K8s Federation across different 5G-EPICENTRE Testbeds Infrastructures (adapted from [1]) 20

Figure 3: a) Scheduling workflow [2] b) example of propagation policy........ 21

Figure 4: Federation built by Karmada with the different Testbed clusters 22

Figure 5: Kubeconfig files to access the clusters 23

Figure 6: Member clusters appearing in the federation after `kubectl get clusters.` The remaining clusters (one per 5G-EPICENTRE testbed) that appear in the list had joined the federation in the same way as those provided as example above........ 24

Figure 7: Sample propagation policy yaml used in UC2 24

Figure 8: Sample service deployment yaml........ 25

Figure 5: 5G-EPICENTRE platform overall functional architecture component diagram......... 26

Figure 6: 5G-EPICENTRE Platform-as-a-Service (PaaS) and Network Applications Layers (delegated at testbeds). 27

Figure 7: 5G-EPICENTRE component integrations supporting the "Scheduling an experiment" usage scenario. 29

Figure 8: 5G-EPICENTRE component integrations supporting the "Experiment deployment" usage scenario ... 32

Figure 9: 5G-EPICENTRE component integrations supporting the "Experiment monitoring during execution" usage scenario 39

Figure 10: 5G-EPICENTRE Upstream Information Flow orchestration 41

Figure 11: UCx, HSPF/Network Intrusion & Detection Network Application and 5G-EPICENTRE components. .. 46

List of Tables

Table 1: Adherence to 5G-EPICENTRE's GA Deliverable & Tasks Descriptions 13

Table 2: Deliverable updates since D4.4 15

Table 3: CI/CD process steps 17

Table 4: API for uploading a file to the Repository using the specified "filename" 28

Table 5: API for deleting a file from the Repository specified by "filename" 28

Table 6: API for returning the list of filenames in the repository... 30

Table 7: API for returning the file information and content specified by "filename"... 30

Table 8: API for creating and queueing a new experiment execution, based on the contents of an experiment descriptor 31

Table 9: API for marking the experiment for cancellation, specified by "id" 31

Table 10: API for starting traffic generation on a remote iPerf Agent specified by the "id" parameter inside "request_body" 33

Table 11: API for terminating traffic generation on a remote iPerf Agent 34

Table 12: API for fetching metrics from the Publisher, using the Prometheus client... 34

Table 13: API for adding metadata for an experiment to the Publisher 35

Table 14: API for removing metadata for an experiment from the Publisher 36

Table 15: API for executing iPerf Agent operations 37

Table 16: API for terminating an iPerf Agent operation... 38

Table 17: API for retrieving the result of the last iPerf Agent operation 38

Table 18: API for obtaining the logs generated in the execution of an experiment specified by "id"................. 40

Table 19: API for obtaining the experiment descriptor for the experiment specified by "id"............................. 40

Table 20: Asynchronous information flow exchanged between the Publisher and the Analytics Engine............ 41

Table 21: Asynchronous information flow exchanged between the Analytics Engine and the Analytics Aggregator 42

Table 22: Network Intrusion & Detection Network Application report message schema (left) and example (right). 44

Table 23: HSPF message fields - possible values and observations 45

Table 24: API for subscribing the analytics service based on the content of a KPI descriptor 46

Table 25: API for deleting the subscription to the analytics services identified by "subscription_id" 47

Table 26: API for returning the descriptor associated with "subscription_id" 47

1 Introduction

The purpose of this deliverable is two-fold, on the one hand recount the development and integration of the project's cross-orchestration solution over the past 12 months (Task T4.3 *"Cross-testbed federation & synchronization"*), as well as report on the means by which integration of the 5G-EPICENTRE experimentation facility has been achieved (Task T4.4 *"End-to-end platform integration activities"*). After identifying Karmada as the ideal solution for fulfilling the requirements and responsibilities assigned to the architectural Federation layer, work in T4.3 was spent on integrating the 5G-EPICENTRE platform with Karmada so to be able to register testbed Kubernetes (K8s) clusters in the envisioned federation. This addresses issues with resource fragmentation, aggregating all resources and offering them through a single point of access. On the other hand, work in T4.4 has focused on identifying and documenting the interdependencies among the different 5G-EPICENTRE components (between Months 1-12, reported in D4.1 *"Integration plan and framework"*); defining the individual components' interfaces over the identified reference points of the 5G-EPICENTRE architectural framework (between Months 13-18, reported in D4.4 *"5G-EPICENTRE experimentation facility preliminary version"*); and identifying and refining the platform's information flows (between Months 16-30, originally reported in D4.4 and technically elaborated in the present deliverable).

This deliverable receives input from the ongoing activities across all technical Work Packages (WPs) and Tasks, and their respective deliverables, and particularly, orchestrated into a comprehensive architectural stack described in D1.4 *"Experimentation requirements and architecture specification final version"*. More specifically:

- D3.1 *"5G EPICENTRE Northbound API"*, which laid the groundwork for the development of a high-level Application Programming Interface (API) simplifying access to infrastructure elements and network functionalities via programmable interfaces for the various applications (what would later become the role of the *Configurator* network application);
- D4.2 *"Network functions implementation"*, which further explores the Network Applications developed in the 5G-EPICENTRE framework;
- D2.7 *"Cloud-native security intermediate version"*, where the components of the security architecture are elaborated;
- D4.3 "Curated Network Application image repository", which defines how the Network Applications catalogue becomes accessible for the 5G-EPICENTRE architectural components;
- and finally (in tandem to this report), D3.2 *"5G EPICENTRE Frontend components"*, which describes the development of the 5G-EPICENTRE frontend components, recounts how the processes for experimenters to upload/ onboard their Experiments have been implemented; and delivers insight on how the upstream information flow, for presenting data generated at the infrastructures to the end-users, has been orchestrated.

With respect to other deliverables in WP2 and WP4, the present report feeds into the development of key components in the 5G-EPICENTRE architecture, *i.e.*, the 5G-EPICENTRE Experiment Coordinator (D2.5 "5G-EPICENTRE experiment execution"); 5G-EPICENTRE Analytics Engine (D2.6 "5G-EPICENTRE Analytics Engine"); and final version of the security components (D2.8 "Cloud-native security final version"), all of which shall provide intricate details of the components themselves, integrated in the means described in this present report. Finally, the deliverable anticipates the upcoming D4.7 document "Integration, Verification and Testing Report final version", where the activities regarding Task T4.5 ("Lab testing, prototyping and validation") will be coupled to the information in this document.

The remainder of this document is organized as follows:

- Section 2 follows up on the contents of D4.1 by establishing the microservices approach followed to deliver on the results reported in this deliverable.

- Section 3 describes the approach toward development of the testbeds' synchronization layer, and the mapping of features of the project-defined cross-testbed MANO API to the Karmada Kubernetes Management System elected for its implementation.
- Section 4 elaborates on the 5G-EPICENTRE platform component interactions, documenting all API developments and the information flows supported by the final version of the platform.
- Section 5 delivers conclusive remarks and outlook on future work.

1.1 Mapping of project's outputs

The purpose of this section is to map 5G-EPICENTRE Grant Agreement (GA) commitments, both within the formal Deliverable and Task description, against the project's respective outputs and work performed.

Table 1: Adherence to 5G-EPICENTRE's GA Deliverable & Tasks Descriptions

5G-EPICENTRE Task	Respective Document Chapters	Justification
T4.3: Cross-testbed federation & synchronization *"[…] define and develop a set of common, standardized interfaces that will intelligently combine the underlying testbed hardware and software components so as to enable the creation of new, virtual components that provide enhanced capabilities".*	Section 3.1 – Initial approach	Section 3.1 discusses how cross-testbed federation has been conceptualized within 5G-EPICENTRE, covering the *definition* aspect of the federation & synchronization approach.
T4.3: Cross-testbed federation & synchronization *"[…]. Key objectives of these interfaces are i) to allow testbeds to federate without losing control of their individual resources; ii) enable the calibration of individual testbed components from a singular control point; iii) allow experimenters to combine the available re-sources to achieve different experimentation conditions of varying scale and diversity; and iv) ensure these configurations are easily repeatable by supporting reproducible experimentation conditions".*	Section 3.2 – Developments	Section 3.2 elaborates on the scheduling component of the Karmada architecture, elaborating on the modes of scheduling that enable the specifically listed objectives (i)-(iv) to be achieved.
T4.3: Cross-testbed federation & synchronization	Section 3.3 – Outcomes	Section 3.3 describes how the cross-testbed MANO API is implemented in the project.

"The output of this Task will be a cross-testbed MANO API […], which will allow access to the many facilities federated under 5G-EPICENTRE by wrapping different aspects of each individual testbed API under a unified information model".		
T4.4: End-to-end platform integration activities *"[…]. It includes planning and realisation of all the 5G-EPICENTRE components' integration into one cloud-native, microservices oriented platform. To this end, 5G-EPICENTRE component instances will run inside containerised environments, such as Docker/K8s, able to tap into cloud-based support layer services (such as databases and middleware) that will also run as micro services inside their own containers".*	Section 2 - Platform integration overview	Section 2 describes the microservices architectural guidelines for the development and integration of the 5G-EPICENTRE architectural components. It describes the employed CI/CD approach and concludes by linking to the project's GitLab repository where the integrated components, modules and subsystems can be retrieved.
	Section 4 – Integrated 5G-EPICENTRE experimentation facility	Section 4 presents a comprehensive overview of the 5G-EPICENTRE platform integration activities, its developed APIs, and the information flows they support.
T4.4: End-to-end platform integration activities *"[…] tools for the continuous integration and continuous delivery of 5G-EPICENTRE will be employed in the integration routine, enabling the continuous delivery and deployment of the system."*	Section 2.2 – CI/CD process	Section 2.2 describes the *Continuous Integration / Continuous Delivery* (CI/CD) approach followed in the context of Task T4.4 *i.e.*, integrating code into a shared repository at regular intervals, and subsequent testing of the integrated code in order to automatically push changes into production.
T4.4: End-to-end platform integration activities *"[…]. Integration will hence be addressed using vertical methods in order to have functional entities and horizontal approaches so as to facilitate any necessary customization of the platform, which will iteratively integrate components resulting from technical WPs to deliver incremental releases of the 5G-EPICENTRE platform."*	Section 4 – Integrated 5G-EPICENTRE experimentation facility	This Section incorporates the 5G-EPICENTRE API specification in a homogenized format that makes API documentation straightforward to all technical partners in the Consortium. The Section allows the technical developers to find out how each API works, and thus generate code (for the remainder of the integration and implementation Task lifecycles), as well as test cases (in the context of Task T4.5).

1.2 Updates since the initial deliverable version

The present document is the second (and final) version of deliverable D4.4. The following Table (Table 2) lists all updates introduced in this latest version of the report, describing what material is new, or updated compared to the previous version.

Table 2: Deliverable updates since D4.4

Document Chapter	Updates since the initial deliverable version
Section 1	The Section has been updated to reflect upon the entire lifecycle of Tasks T4.3 and T4.4. A novel Section 1.2 has been added to highlight changes of the deliverable since its preliminary version (D4.4).
Section 2	There are no significant updates to this Section since D4.4.
Section 3	The Section has been updated to reflect the integration of the platforms in the cross-testbed federation since D4.4.
Section 4	The Section has been updated to address the latest platform specifications, as listed in D1.4. It delivers a similar structure of elaborating on component interdependencies based on information flow, but has been further broken down into the actual usage scenarios of the 5G-EPICENTRE platform in general, namely: 1) **Delegating vertical application components to the testbed operator**; 2) **Scheduling an experiment**; 3) **Experiment deployment**; and 4) **Experiment monitoring during execution**.
Section 5	Section 5 has been updated to reflect on the updates introduced in this document, and its positioning in the project work plan timing.

2 Platform integration overview

This section will outline the general process for the integration of individual components and services into the overall 5G-EPICENTRE integrated platform. As mentioned, 5G-EPICENTRE follows the CI approach to integration, enabling a multitude of developers to simultaneously work on parts of the integrated system in parallel. In this approach, as soon as a component development cycle is complete, it is integrated into the overall system and tested. Due to this workflow, it is important for the development team to focus on small changes and incremental additions to component functionality; otherwise, integration will become a bottleneck in the production of the solution.

2.1 Integration environment

5G-EPICENTRE adopts a microservices architecture, which essentially looks upon system components as a collection of loosely coupled services, communicating with one another through either synchronous (*i.e.*, Representational State Transfer – REST), or asynchronous (publish/subscribe messaging, *e.g.*, Message Queuing Telemetry Transport – MQTT) protocols implemented over well-defined interfaces. Adoption of this paradigm helps the 5G-EPICENTRE integration team to deliver on the large, complex experimentation facility architecture in a coordinated and rapid fashion.

Implementation of the project microservices architecture is based on a well-defined (D4.1) software lifecycle and Development and Operations (DevOps) platform in GitLab[1]. GitLab incorporates a variety of features to handle the end-to-end software development and operations workflows, most notably (and of relevance to the microservices architectural paradigm) the GitLab Container Registry. This extension integrates a private registry for Docker images, which contain everything needed to run an application, including source code and dependencies. Delivered images can then be deployed and executed as Docker containers, complete with hosting environment, dependencies, *etc*., enabling developers to build, push and deploy containers using GitLab CI for the purposes of testing an application as soon as changes are introduced to branches or tags.

For development and testing purposes, Kubernetes (K8s) is used as the staging environment, taking advantage of GitLab's integration with K8s. K8s is responsible for scheduling and running containerised applications over one or more containers, while simultaneously automating container operational tasks, such as creation, starting, organising, monitoring, and destruction. In addition, K8s can automatically restart containers after they have crashed, managing thus horizontal scalability. K8s can run on both a local cluster or be deployed in private or public Clouds. GitLab offers integration with K8s in multiple ways, of which the most relevant to the 5G-EPICENTRE integration plan is the capacity to build and deploy software from GitLab CI/CD pipelines to a K8s cluster.

2.2 CI/CD process

As 5G-EPICENTRE follows a microservices-based approach, developers are free to implement their components using the software stack of their choice. Developers are further responsible for upgrading the source code of the various components incrementally, and should write the proper automated tests that verify the implementation of a function or class inside a software module. Developers are encouraged to work on isolated, local development machines. Hence, the integrated prototype is maintained on a separate production environment, independent to the development environment.

The project developers will be responsible to incrementally improve and add features to the integrated 5G-EPICENTRE platform prototype, by adding functionality introduced in the most recent updated versions of the individual components in frequent integration and delivery steps, as shown in Figure 1.

[1] https://about.gitlab.com/

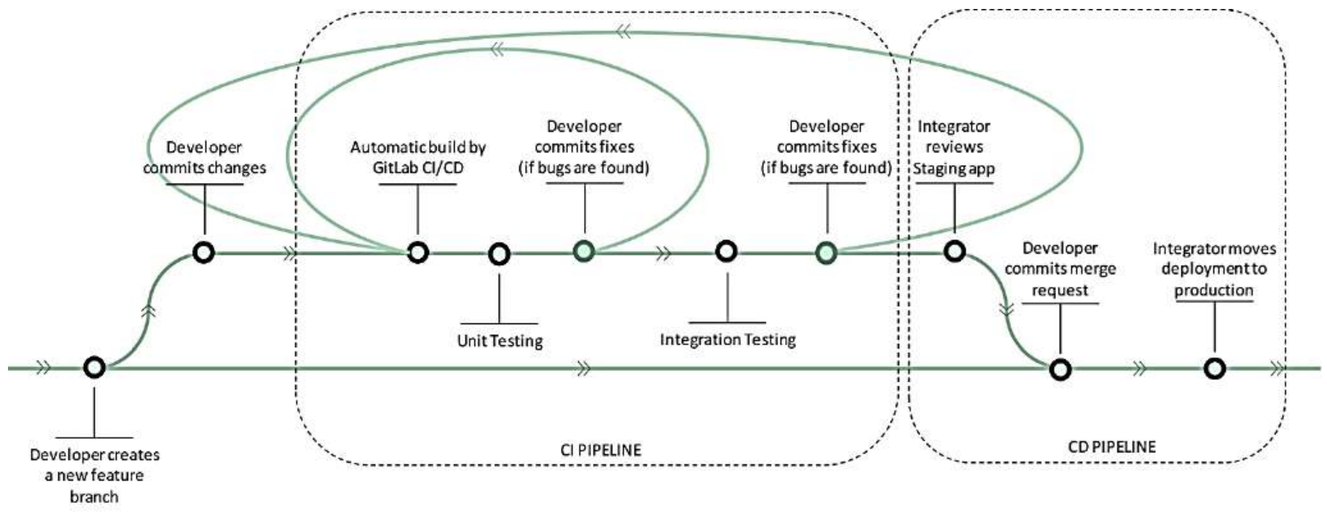

Developer commits changes
Automatic build by GitLab CI/CD
Developer commits fixes (if bugs are found)
Developer commits fixes (if bugs are found)
Integrator reviews Staging app
Developer commits merge request
Integrator moves deployment to production
Unit Testing
Integration Testing
Developer creates a new feature branch
CI PIPELINE
CD PIPELINE

Figure 1: The 5G-EPICENTRE CI/CD process

The individual steps depicted in this CI/CD process are described in Table 3.

Table 3: CI/CD process steps

Step number	Activities
1	**Developer creates a new feature branch:** Before any new developments on an individual component take place, it is important that the component developer obtains the latest version of the software currently in the production environment. On the component's GitLab repository, a new feature branch (*i.e.*, a clean copy of the current master initial branch) is created for the specific component indicating a new, independent line of development for the implementation of a particular added feature that the component should provide.
2	**Developer commits changes:** The local development activity takes place, introducing new code, updating older code, optimising existing code, and improving overall clarity of the source files. All work should be carried out on an independent, local developer machine. Following the CI/CD methodology, such changes should be committed to the branch in frequent intervals to avoid adding complexity.
3	**Automatic (test) build:** Utilising the GitLab CI/CD tool, the push should trigger a pipeline, configured to be executed with jobs running in separate, isolated Docker containers. A test app is deployed, which is used to verify changes against component specifications (unit testing) and interfacing with other components already in the production environment (integration testing).
4	**Unit testing:** The CI/CD pipelines invoked upon a push should contain an automated unit test job, whose purpose is to verify the code. Isolated parts of the built application are tested to indicate any errors or bugs introduced regarding unit functionality. Automated unit testing is performed by means of a specific unit testing framework applicable to the programming language used. Thorough information on the test planning is presented in deliverable D4.6.
5 (optional)	**Developer commits unit test fixes:** If the tests in the previous step should fail, the pipeline should fail as well. The developer should review the job logs and pinpoint the tests that have

	failed so that fixes can be introduced into the code. If no unit test was present for a particular discovered bug earlier, a unit test should be implemented in the upcoming bug fix commit. The fix is committed with a commit message that includes the bug number so as to facilitate the tracking of bug fixes. The new commit should trigger a new pipeline reverting the process back to step 3. If no tests fail during unit testing at step 4, this step is omitted altogether.
6	**Integration testing:** Integration testing should follow unit testing as an automated test job in the CI/CD pipeline and address the need for the new/updated component to work together with other components. Integration tests should therefore focus on component interface integrity. Whenever a component or system is not physically available for the test, a simulated (or "mock") response/command set is used to emulate its usage. Thorough information on the test planning is presented in deliverable D4.6.
7 (optional)	**Developer commits integration test fixes:** If integration testing fails to verify the compatibility of components in the integrated prototype, much like in step 5, the developer of the component should introduce appropriate fixes. As is the case with step 5, the fix is committed with a commit message that includes the bug number so as to facilitate the tracking of bug fixes. The new commit should trigger a new pipeline reverting the process back to step 3. If no tests fail during unit testing at step 6, this step is omitted altogether.
8	**Integrator reviews staging app:** The GitLab CI/CD pipeline should progress to the next job which should deploy the code into a staging environment prior to its release in production. The main purpose of this step is to facilitate a pre-release version of the system, with all new changes deployed in order for the integrators to preview them and ensure that they are working as expected.
9	**Developer commits merge request:** When the new component meets success criteria defined in its test plan (unit and integration testing), and the new feature is approved, the developer merges the branch changes into the main codebase (default branch). In the unlikely event of a conflict during the merge operation, the developer is tasked with resolving these conflicts manually. This includes downloading the latest changes to the code of the default branch, which should include any changes made since the feature branch was created. The developer should re-ensure that their version of the component works in a similar fashion to the previous default codebase, *i.e.*, repeat the CI/CD process. The merge request can then be re-attempted.
10	**Integrator moves deployment to production:** A new production build is deployed and all developers are notified. Every new feature should now trigger the CI/CD process at step 1, using the new codebase as the default branch.

2.3 Gitlab repository

As already mentioned, GitLab is utilised as the centralized source code repository and management system to keep track of every part of the 5G-EPICENTRE source code, contributed to by all the technical partners. Using GitLab as a distributed version control system tool, developers are able to clone the code on a local machine, creating a safe environment for experimentation without affecting functionality of the overall system. Once the desired features or changes have been implemented, the updated code is pushed back to the remote repository, making them available to the entirety of the development team.

A dedicated 5G-EPICENTRE group has been set up in GitLab, *i.e.*, https://gitlab.5gepicentre.eu.

3 Cross-testbed federation & synchronization approach

One of the main features of 5G that was not present in previous technologies is the virtualisation of its different elements. This offers versatility, dynamism and flexibility, both in its deployment and in the interactions with verticals, and opens the door to the intelligence that will be present in future network generations. For this reason, the adoption of microservices in the framework of 5G-EPICENTRE has been supported, both at the platform level and at the experimentation level. The platform must be ready for experimental needs and at the same time, align with technology trends to have a useful life beyond the lifetime of the project. However, the orchestration and management of multiple services is complex, and becomes even more complicated when elements are mixed, such as different testbeds, platform components, or the verticals themselves. This is why the project partners have opted for the advantages provided by ***Karmada***[2], which will serve as an abstraction layer to simplify the K8s orchestration processes.

3.1 Initial approach

The 5G-EPICENTRE project is developing a federated architecture for testing 5G applications. One of the key components of this architecture is the cross-testbed Management & Orchestration (MANO), which will provide a unified interface for managing resources across multiple testbeds.

The initial approach involves the concept of cross-testbed federation, which enables multi-clustered K8s orchestration across different domains. A K8s-based orchestrator manages and provides access to service resources for individual containerized workloads in testbeds, allowing for testbed federation and synchronization. The cross-testbed federation could be implemented using Karmada, a K8s management system that allows the federation of multiple K8s cluster.

Karmada is an open, multi-cloud, multi-cluster K8s orchestration platform, that facilitates the deployment of cloud-native applications across multiple K8s clusters without requiring application modifications. It leverages a centralized control plane to deploy and manage applications on multiple clusters, known as *member clusters*. Karmada also supports the federation of K8s resources in a multi-cluster environment. Initially, Karmada provides cluster federation across different testbed infrastructures, with the potential for further development to meet the architectural requirements of the 5G-EPICENTRE project (see also D1.4).

Karmada offers open and multi-cloud K8s orchestration by utilizing K8s-native APIs and advanced scheduling capabilities. Key features that are relevant to the project's cross-testbed MANO solution include:

- **Centralized management:** Karmada provides a centralized API for managing resources across multiple clusters. This simplifies the management of resources, and makes it easier to track the status of applications.
- **Multi-cluster scheduling:** Karmada can schedule applications across multiple clusters, which can improve the performance and availability of applications.
- **Fault tolerance:** Karmada can tolerate failures of individual clusters, which ensures that applications continue to run even if one of the clusters fails.

The architecture of Karmada resembles that of a single K8s cluster, with components such as a control plane, API server, scheduler, and controllers. Figure 2 illustrates these components. They play a crucial role in facilitating communication, policy enforcement, and workload management within the cross-testbed MANO framework.

The API server provides K8s native APIs and policy APIs, while the scheduler implements sophisticated multi-dimensional, multi-weight, and multi-cluster scheduling policies. Karmada employs an agent-based pull mode for member cluster synchronization, enabling efficient management of large-scale, multi-cluster resource pools.

[2] https://karmada.io/

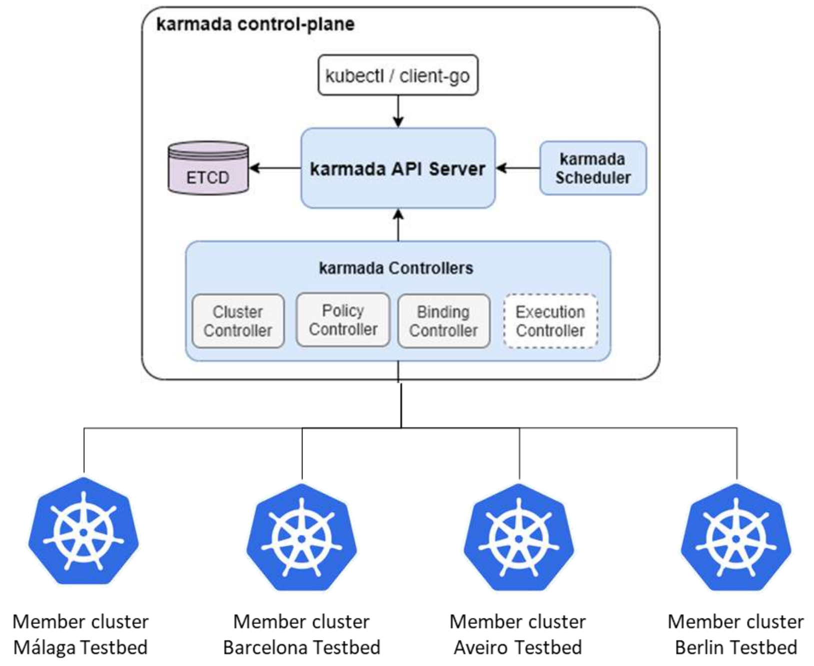


Figure 2: Multi-cluster K8s Federation across different 5G-EPICENTRE Testbeds Infrastructures (adapted from [1])

Additionally, it offers support for Execution Controller and KubeEdge integration, enabling seamless management of K8s clusters in diverse network environments, including public, private, and edge clouds. The execution provided by Karmada ensure isolation of access permissions and resources across multiple clusters.

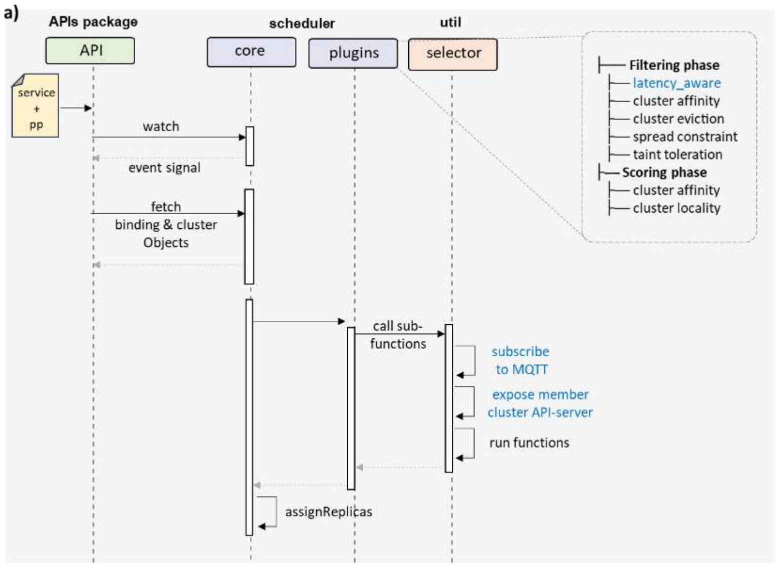
The configuring federated resources in Karmada involves three fundamental concepts: ***template***, ***placement***, and ***override***. The resource *template* defines attributes shared among member clusters, while *placement* determines the selection of a member cluster for a federated resource. *Override* allows for cluster-specific configurations, tailored to selected member clusters.

By adopting Karmada as a cross-testbed federation solution, the 5G-EPICENTRE project can achieve efficient resource management, synchronization and 5G applications deployment across diverse testbed infrastructures.

3.2 Developments

Karmada incorporates a Policy API as multi-cloud scheduling policy, or *"Propagation Policy"* (Figure 3b), to manage the propagation of services across multiple K8s clusters. This term serves as a declarative mechanism for defining how the resources submitted to system should be deployed into a single, or group of clusters, based on attribute or label. All services must include the propagation policy within deployment in which the user identifies the deployment policies. The presented Figure 3a succinctly illustrates the workflow of the scheduler, providing an overview of the key stages. Upon submission, the binding controller receives this information and proceeds to populate the necessary details within the binding object. Karmada scheduler uses a watch method to API server, and is notified about the changes of those binding resources [2]. Once the scheduler is triggered, the process of scheduling starts based on the plugin enabled in the scheduler images. Karmada scheduler has a pluggable framework, allowing the definition of diverse sub-functions without altering the core of the scheduler. It means that the Karmada scheduler can easily incorporate new scheduling algorithms or policies as plug-ins, without requiring significant modifications to the underlying system. For the moment, the built-in plugin in Karmada mainly are: *cluster affinity*; *API enablement*; *cluster eviction*; *cluster locality*; *spread constraint*; *taint*; and *toleration*, and uses the role of resource selector to apply the plugins to the specified resources. In the context of the 5G-EPICENTRE project, the deployment mainly uses **cluster affinity** (see Section 3.2.1) and **latency-aware** (see Section 3.2.2) plugins for the purpose of testing.

a)



b)

```
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: name of the service ( or object)
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: service1
  placement:
    clusterAffinity:
      clusterNames:
        - Cluster 1
        - Cluster 2
spreadConstraints:
      - spreadByField: cluster
        maxGroups: 1
    replicaScheduling:
      replicaSchedulingType: Duplicated or Divided
```

Figure 3: a) Scheduling workflow [2] b) example of propagation policy

Propagation policy also encompasses the capability of scheduling replicas using two strategies, *i.e.*, *Duplicate* and *Divide*. In the Duplicate strategy, all replicas are consolidated within a single cluster, while the Divide strategy distributes the replicas among all identified clusters, aggregating them in a single cluster, or proportionally allocating them based on the weight assigned to each cluster. This dynamic allocation mechanism specified in the Propagation Policy offers users the flexibility to choose the most suitable strategy that aligns with their specific workload and resource requirements.

3.2.1 Cluster affinity plugin

Cluster affinity plugin is a built-in plugin of Karmada's scheduler, which allows Verticals to influence the scheduling and placement of Vertical, or Network Applications onto specific cluster. It enables fine-grained control over service placement, by defining rules that express the desired association between service and clusters based on cluster attributes. Cluster affinity enhances better resource utilization, by strategically distributing services across clusters with the most suitable resources. It has four fields to set in propagation policy: label selector, field selector, cluster names, and exclude clusters that can be defined as affinity in propagation policy to filter out and score the clusters [1].

3.2.2 Latency-aware plugin

The latency-aware plugin is a customized scheduler plugin to guarantee the fulfilment of required Key Performance Indicators (KPIs) by computing the optimal placement for a given service, while the cluster resources (like Central Processing Unit - CPU) is used efficiently. To ensure that the latency is fulfilled, the scheduler subscribes to a RabbitMQ broker to obtain the measured ping time as latency of each member cluster. If the latency is fulfilled, the service is deployed in cluster with the best conditions. If the latency is not fulfilled, the service will be load balanced among clusters with less latency. In this regard, the scheduler utilizes a set of client sets to expose the member clusters' API to retrieve the aggregated available CPU resources of each cluster.

The functionality provided by the latency-aware plugin is found to be incompatible with affinity rules in propagation policy, since the key parameter is the requested KPI from the vertical. Once the user submits the deployment, the cluster list should be empty (the list of clusters in Figure 3b), thereby allowing the function to potentially utilize all available resources comprehensively.

3.3 Outcomes

The federation is composed of 4 testbeds, each hosting its own K8s clusters. In the implementation of the federation built by Karmada, Figure 4 showcases the connectivity and relationships between these components:

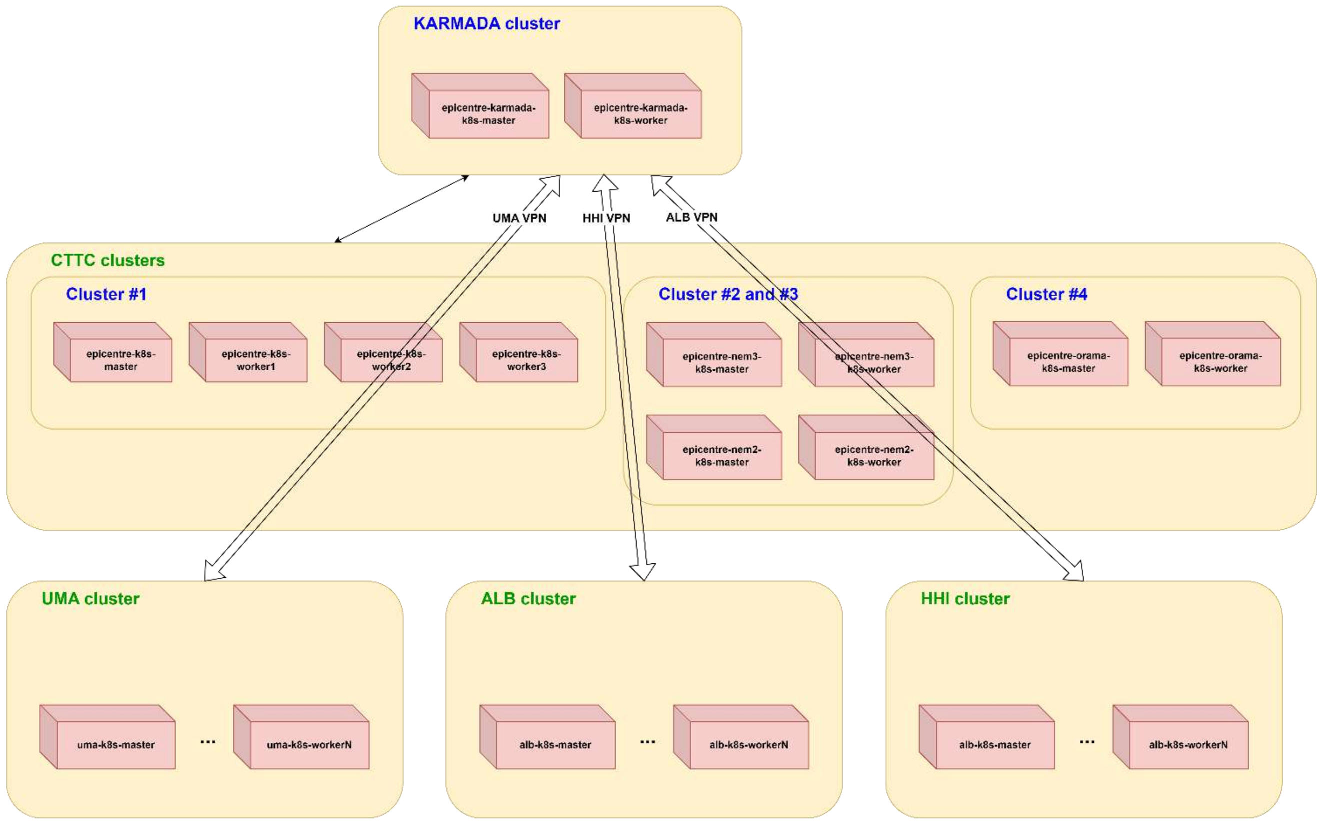


Figure 4: Federation built by Karmada with the different Testbed clusters

To begin, Karmada is installed within a single K8s cluster located at CTTC. This serves as the central control plane for the federation. To establish communication and coordination between CTTC and each testbed, a Virtual Private Network (VPN) is set up. Using the established VPN connections, the K8s cluster hosted within each testbed can be seamlessly joined to the federation created by Karmada. This means that the resources and capabilities offered by these individual testbeds become part of the larger federation.

CTTC not only serves as the provider of the overall federation infrastructure, but also contributes with several K8s clusters for the various use cases it hosts. These locally provided K8s clusters within CTTC are also integrated into the federation, enhancing the federation's collective resources, and enabling a wider range of use cases to be supported.

Overall, the federation, orchestrated by Karmada and interconnected through VPNs, enables the collaboration and coordination of resources across the different testbeds and their respective K8s clusters. This integration promotes efficient resource utilization, increased scalability, and the ability to execute diverse applications and experiments within the federation.

3.4 Service deployment process in the federation

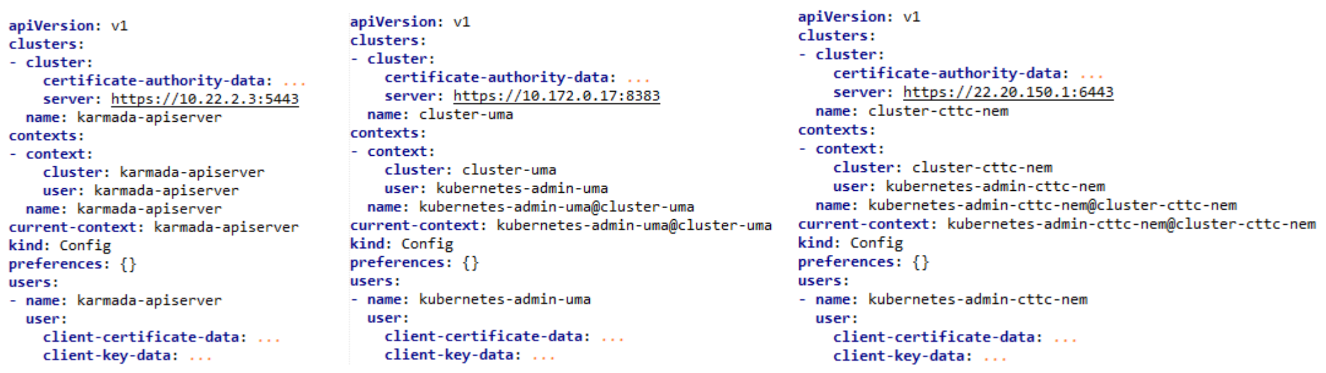
The 5G-EPICENTRE architecture is designed in a modular way so that the services offered to other building blocks of the architecture are clearly defined. In this sense, the operation of the federation layer is transparent to the experimenter. That is, the experimenter interacts with the 5G-EPICENTRE experimentation framework through the Portal (see D3.2), where they select the testbed in which the service needs to be deployed. Then, the Experiment Coordinator (see D2.5) maps this selection to the corresponding field of the propagation policy that is needed by Karmada, and sends the request to Karmada. According to this operation, the service is automatically deployed by Karmada in the corresponding testbed.

In summary, the federation layer allows exposing all testbeds of the project as an aggregated experimentation platform that is consumed in a unified way by the building blocks of the Backend Layer of the 5G-EPICENTRE architecture (see D1.4). Therefore, the Backend Layer consumes the federation layer API on the southbound, and interacts with the experimenter on the northbound.

However, before the federation acts transparently to the end-user in coordination with the backend-layer, the federation manager needs to set it up by following the steps below.

3.4.1 Accessing Karmada and the Kubernetes clusters

Karmada itself is deployed in a K8s cluster. To access it, kubeconfig files need to be configured, being the main parameter to configure the IP address, port, user, and credentials, through which the cluster will be accessible. Figure 9 presents an example, where the leftmost file corresponds to the cluster where Karmada is deployed, and the other two correspond to two clusters that will be part of the federation (others may join as well in the same way).



```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: ...
    server: https://10.22.2.3:5443
  name: karmada-apiserver
contexts:
- context:
    cluster: karmada-apiserver
    user: karmada-apiserver
  name: karmada-apiserver
current-context: karmada-apiserver
kind: Config
preferences: {}
users:
- name: karmada-apiserver
  user:
    client-certificate-data: ...
    client-key-data: ...
```

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: ...
    server: https://10.172.0.17:8383
  name: cluster-uma
contexts:
- context:
    cluster: cluster-uma
    user: kubernetes-admin-uma
  name: kubernetes-admin-uma@cluster-uma
current-context: kubernetes-admin-uma@cluster-uma
kind: Config
preferences: {}
users:
- name: kubernetes-admin-uma
  user:
    client-certificate-data: ...
    client-key-data: ...
```

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: ...
    server: https://22.20.150.1:6443
  name: cluster-cttc-nem
contexts:
- context:
    cluster: cluster-cttc-nem
    user: kubernetes-admin-cttc-nem
  name: kubernetes-admin-cttc-nem@cluster-cttc-nem
current-context: kubernetes-admin-cttc-nem@cluster-cttc-nem
kind: Config
preferences: {}
users:
- name: kubernetes-admin-cttc-nem
  user:
    client-certificate-data: ...
    client-key-data: ...
```

Figure 5: Kubeconfig files to access the clusters

3.4.2 Setting up the federation

The next step is to build the federation, by making each cluster join it as a member cluster. Following the above example, we have two member clusters, with names *cluster-uma* and *cluster-cttc-nem*, that join the federation through the following commands:

```
kubectl-karmada join cluster-uma --kubeconfig=$HOME/clusters_config/karmada-apiserver.config --clus-
ter-kubeconfig=$HOME/clusters_config/member-uma.config --cluster-context=cluster-uma
```

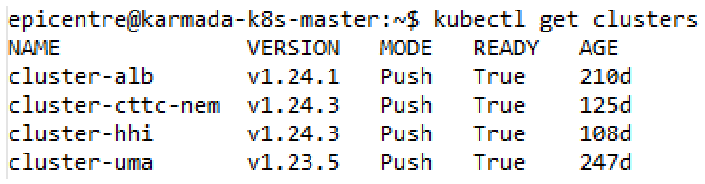
```
kubectl-karmada join cluster-cttc-nem --kubeconfig=$HOME/clusters_config/karmada-apiserver.config --
cluster-kubeconfig=$HOME/clusters_config/member-cttc-nem.config --cluster-context=cluster-cttc-nem
```

In addition to the cluster name, the parameters provided in the command are the kubeconfig file of the Karmada API server, that will act as entry point of the Karmada control plane of the federation; and the kubeconfig file of the member cluster.

We can confirm that they currently joined the federation through the following command:

```
kubectl get clusters
```

which yields the results depicted in Figure 6.



```
epicentre@karmada-k8s-master:~$ kubectl get clusters
NAME              VERSION   MODE   READY   AGE
cluster-alb       v1.24.1   Push   True    210d
cluster-cttc-nem  v1.24.3   Push   True    125d
cluster-hhi       v1.24.3   Push   True    108d
cluster-uma       v1.23.5   Push   True    247d
```

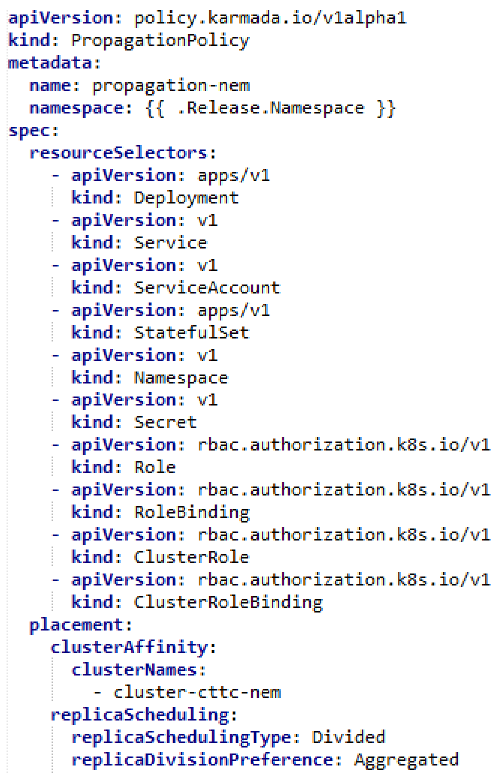
Figure 6: Member clusters appearing in the federation after `kubectl get clusters`. The remaining clusters (one per 5G-EPICENTRE testbed) that appear in the list had joined the federation in the same way as those provided as example above.

3.4.3 Deploying services through Karmada

Once the federation is set up, one can deploy services in the federation in (almost) the same way as done in a regular K8s cluster, including using the same commands. The main difference is that an additional configuration yaml file needs to be provided, to specify the constraints of the deployment that will be applied to the service, as far as the multiple clusters that compose the federation are concerned. This is done by applying what is referred to as *propagation policy*, by executing the following command:

```
kubectl apply -f propagationpolicy.yaml
```

This includes, for instance, *clusterAffinity*, which defines what clusters can be used to deploy a given service (and so, which cannot) or *replicaScheduling*, which defines the philosophy followed to distribute the replicas of a given service throughout the cluster, in case there are replicas. See the example in Figure 7.



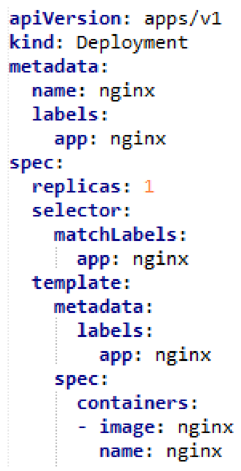
```
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: propagation-nem
  namespace: {{ .Release.Namespace }}
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
    - apiVersion: v1
      kind: Service
    - apiVersion: v1
      kind: ServiceAccount
    - apiVersion: apps/v1
      kind: StatefulSet
    - apiVersion: v1
      kind: Namespace
    - apiVersion: v1
      kind: Secret
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: Role
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: RoleBinding
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRole
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRoleBinding
  placement:
    clusterAffinity:
      clusterNames:
        - cluster-cttc-nem
    replicaScheduling:
      replicaSchedulingType: Divided
      replicaDivisionPreference: Aggregated
```

Figure 7: Sample propagation policy yaml used in UC2

Once the deployment constraints are defined, the service can be deployed in the same way as in a regular K8s cluster, through the following command:

```
kubectl apply -f deployment.yaml
```

Figure 8 presents a simple example of the deployment file of a service consisting of a nginx pod.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
```

Figure 8: Sample service deployment yaml

Once these steps are followed, Karmada will deploy the service in the corresponding member cluster(s), by sending them the corresponding commands transparently.

As mentioned above, these commands are explained here for completeness, though they are transparent to the experimenter, since the backend layer and the federation have been integrated, so that the deployment characteristics specified through the Portal are automatically mapped to these commands. More specifically, the Experiment Coordinator building block of the Backend Layer is the one sending these commands to Karmada.

After these commands are run, the service is fully operational in the federation. For more details on the behaviour/performance of services in testbeds once they are deployed, see deliverable D5.2.

4 Integrated 5G-EPICENTRE experimentation facility

This Section elaborates on the integration of the 5G-EPICENTRE experimentation facility. The integration ultimate goal is to deliver a concrete implementation of the reference architecture defined in D1.4. As specified in D4.4, the system implements both *downstream* (*i.e.*, usage of front-end tools to setup and calibrate infrastructure components) and *upstream* (*i.e.*, infrastructure components generating data that is presented to the user via a user interface) information flows (illustrated in Figure 9). These aim at supporting the platform usage scenarios, as defined in D1.4 (Section 3.7 Information view):

- **Delegating vertical application components to the testbed operator** (downstream).
- **Scheduling an experiment** (downstream).
- **Experiment deployment** (downstream).
- **Experiment monitoring during execution** (upstream).

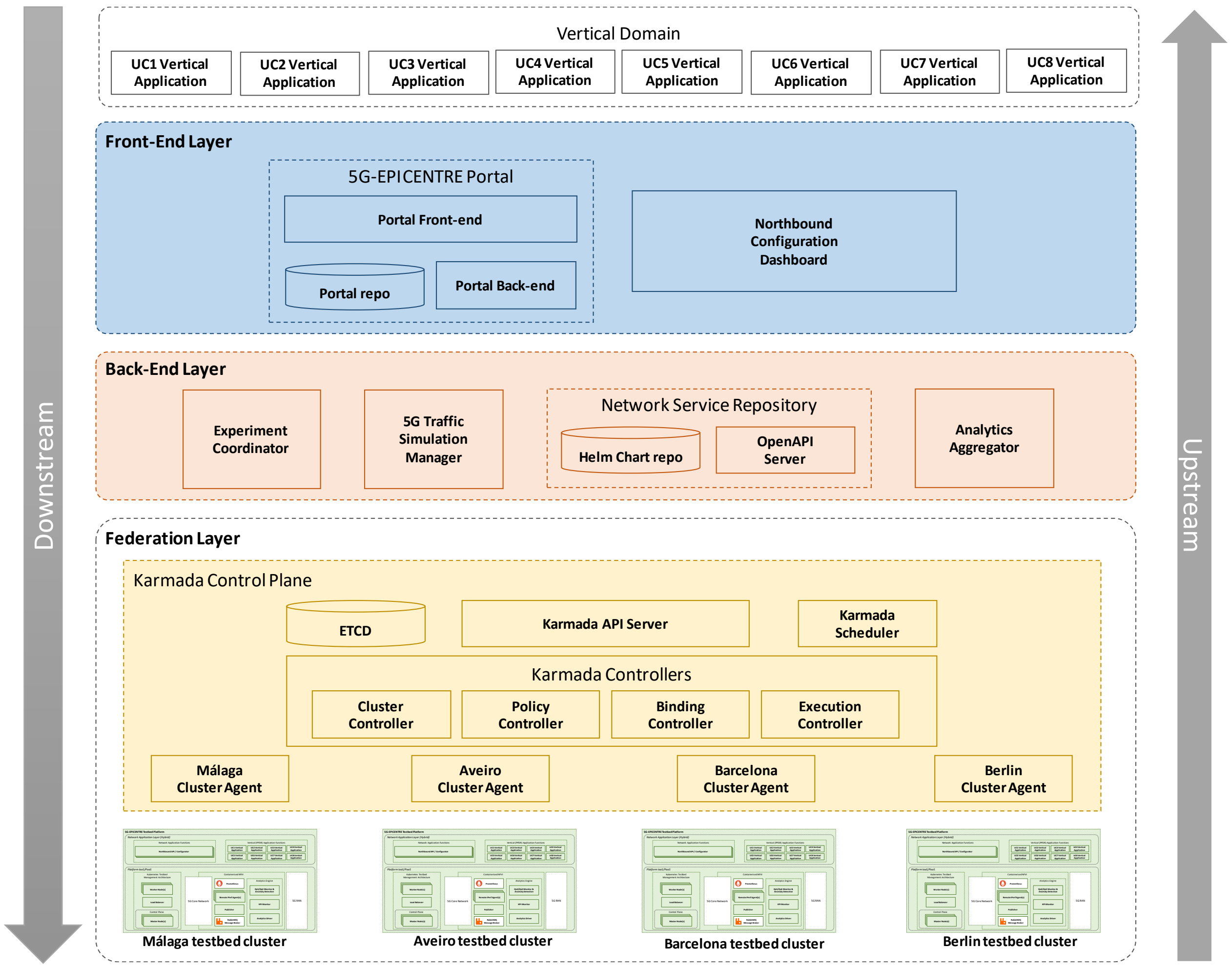


Figure 9: 5G-EPICENTRE platform overall functional architecture component diagram.

Integration further alludes to the developments necessary to homogenize the testbeds in terms of supporting the functionalities that the 5G-EPICENTRE platform and underlying infrastructures have to offer, such as analytics calculation and traffic simulation. This augmentation architecture is summarized in Figure 10.

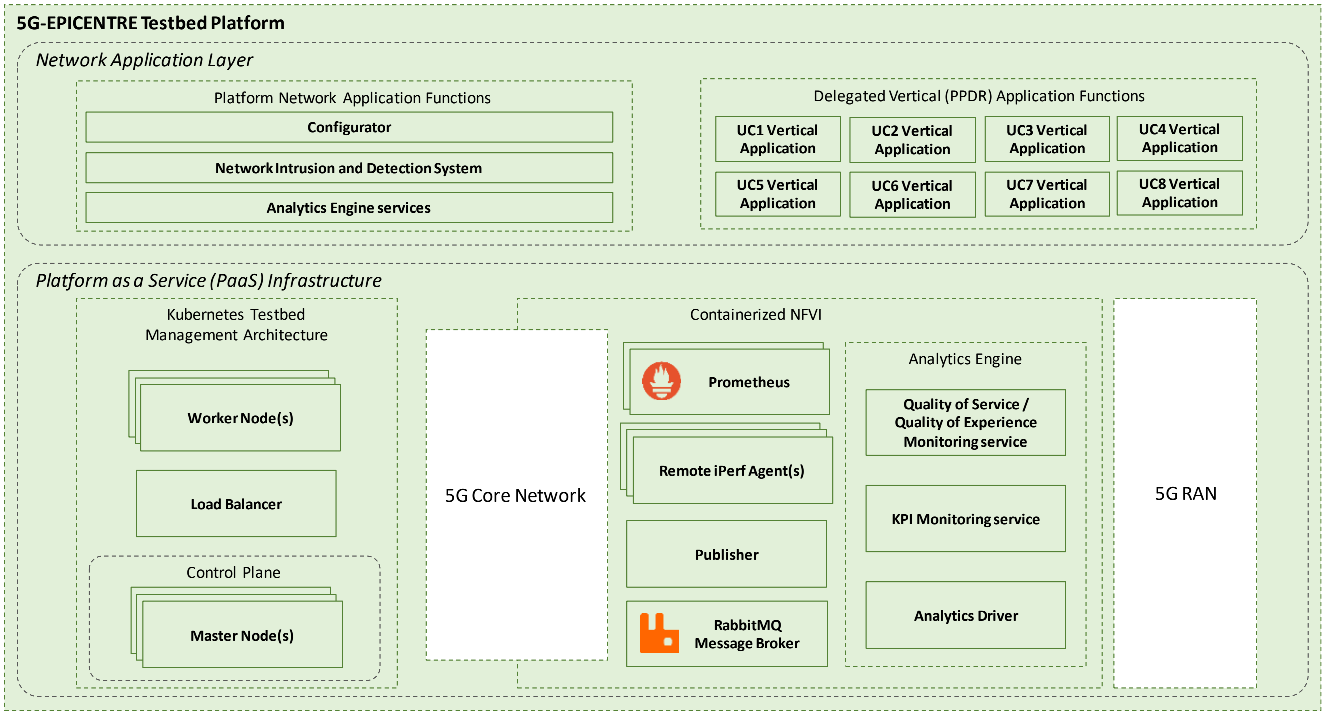


Figure 10: 5G-EPICENTRE Platform-as-a-Service (PaaS) and Network Applications Layers (delegated at testbeds).

Finally, integration corresponds to the specification of the information flow throughout the system; which components should be involved in the different usage scenarios; and how information is orchestrated and flows throughout the system during its usage. Integration is hence addressed through the definition and implementation of well-defined *interfaces*, with concretely established inputs and outputs. A component thus exposes methods to other elements, or consumes methods exposed by other elements in the architecture, so that exchange of data can be facilitated via pre-specified sets of parameters. The outcome of this entire process, which corresponds to activities in Task 4.4, is the concrete definition of all necessary *Application Programming Interfaces* (APIs) that implement these procedures. In 5G-EPICENTRE, two components are considered "integrated" when either one is able to consume the APIs exposed by the other.

The remainder of this Section will elaborate on the integration of the individual components and technologies utilized towards delivering on the 5G-EPICENTRE experimentation facility usage scenarios.

4.1 Delegating vertical application components to the testbed operator

In this scenario, the experimenter transfers (*i.e.*, *delegates*) the vertical experiment application components to the testbed administrator(s), for the latter to get the experiment environment up and running. The transfer is based on the Helm chart template, to leverage its simplicity and automation benefits for K8s deployments [3]. After the delegated Helm chart package has been inspected and verified by a testbed administrator, it is eventually hosted on the Network Service Repository, becoming available to the Experiment Coordinator component (see Section 4.2).

The downstream information flow supporting this usage scenario involves integration of Front-end (*i.e.*, 5G-EPICENTRE Portal) and Back-end Layer components (*i.e.*, Network Service Repository), and should support different variants, such as being able to *delete* vertical application Helm charts from the repository.

The following paragraphs describe all APIs used to connect the various entities in the 5G-EPICENTRE architecture for facilitating this usage scenario.

4.1.1 5G-EPICENTRE Portal - Network Service Repository integration

Towards supporting the operations described in this usage scenario, the **5G-EPICENTRE Portal** ("*Portal*", described in D3.2) consumes APIs exposed to it by the **Network Service Repository** (*"Repository"*) OpenAPI Server sub-module.

4.1.1.1 Put File API

This API is consumed by the *Portal* to upload a file to the *Repository*. It uses the PUT request methods to either create a brand-new resource, or replace a previous one with the target resource name. The API documentation is summarised in Table 4. If the query is not successful, the response contains the corresponding error code and message.

Table 4: API for uploading a file to the Repository using the specified "filename"

Put File	
Method	PUT
Endpoint	/:filename
Request Headers	**Authorization** [String] REQUIRED Basic Username and Password based authentication scheme for the user making the request, containing the Basic word followed by a space and a base64-encoded username:password string.
Request Body	**filedata** [Blob] REQUIRED The file to be uploaded to the repository, formatted as 'Blob' immutable raw data.
Response	**200** The query was successful.
	500 The server cannot process the request for an unknown reason.

4.1.1.2 Delete File API

This API is consumed by the *Portal* to delete a file from the *Repository*. It uses the DELETE request method to permanently remove the specified resource. The API documentation is summarised in Table 5. If the query is not successful, the response contains the corresponding error code and message.

Table 5: API for deleting a file from the Repository specified by "filename"

Delete File	
Method	DELETE
Endpoint	/:filename
Request Headers	**Authorization** [String] REQUIRED Basic Username and Password based authentication scheme for the user making the request, containing the Basic word followed by a space and a base64-encoded username:password string.
Request Body	None
Response	**200** The query was successful.

	404 The user is able to communicate with the server but it is unable to locate the requested file or resource for deletion.
	500 The server cannot process the request for an unknown reason.

4.2 Scheduling an experiment

In this scenario, the experimenter requests the scheduling and execution of a vertical application experiment, by selecting the prior delegated vertical application components (see Section 4.1). The process involves exposure of calibration parameters to specify the experiment environment, *e.g.*, simulation of varying types of network traffic, or definition of vertical-specific KPIs to be collected and monitored during the experiment. The information is stored in an *experiment descriptor* file, *i.e.*, a structure exchanged between the Portal and the Backend layer experiment coordination components, toward describing the creation of all necessary K8s objects to be deployed in the designated testbed cluster (see also D3.2). Upon acceptance of the experiment by the testbed administrator, the Experiment Coordinator automatically applies the deployment YAML to the Karmada API server (see Section 3) at the designated time window where the execution was scheduled.

The downstream information flow (see Figure 11) supporting this usage scenario involves primarily integration of Front-end (*i.e.*, 5G-EPICENTRE Portal) and Back-end Layer components (*i.e.*, Network Service Repository, Experiment Coordinator), and should support different variants, such as being able to *cancel* an experiment upon request by the experimenter.

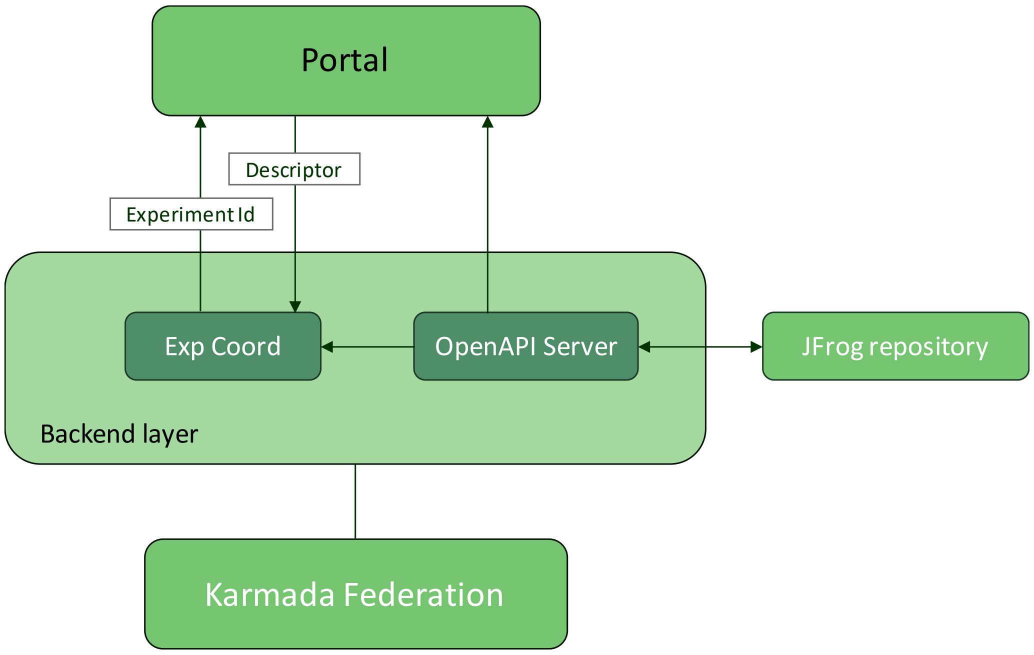


Figure 11: 5G-EPICENTRE component integrations supporting the "Scheduling an experiment" usage scenario

The following paragraphs describe all APIs used to connect the various entities in the 5G-EPICENTRE architecture for facilitating this usage scenario.

4.2.1 5G-EPICENTRE Portal – Network Service Repository

Towards supporting the operations described in this usage scenario, the **5G-EPICENTRE Portal** ("*Portal*", described in D3.2) consumes APIs exposed to it by the **Network Service Repository** (*"Repository"*) OpenAPI Server**.**

4.2.1.1 Get All API

This API is consumed by the *Portal* to retrieve the complete list of filenames in the *Repository*, for the experimenter to obtain their own vertical application artefacts for experimentation. The API documentation is summarised in Table 6. If the query is not successful, the response contains the corresponding error code and message.

Table 6: API for returning the list of filenames in the repository

Get All	
Method	GET
Endpoint	/
Request Headers	**Authorization** [String] REQUIRED Basic Username and Password based authentication scheme for the user making the request, containing the "Basic" word, followed by a space and a base64-encoded username:password string.
Request Body	None
Response	**200** The query was successful. The response will contain a list of filenames in the repository, or an empty list, if no filenames are found.
	500 The server cannot process the request for an unknown reason.

4.2.1.2 Get File API

This API is consumed by the *Portal* to retrieve the specified file information and metadata from the *Repository*, which can be used to further elaborate experimentation parameters with the vertical application's microservices. The API documentation is summarised in Table 7. If the query is not successful, the response contains the corresponding error code and message.

Table 7: API for returning the file information and content specified by "filename"

Get File	
Method	GET
Endpoint	/:filename
Request Headers	**Authorization** [String] REQUIRED Basic Username and Password based authentication scheme for the user making the request, containing the Basic word followed by a space and a base64-encoded username:password string.
Request Body	None
Response	**200** The query was successful. The response will contain the contents of "Chart.yaml", if the file is in an archive file format.
	404 The user is able to communicate with the server, but it is unable to locate the requested file or resource.
	500 The server cannot process the request for an unknown reason.

4.2.2 5G-EPICENTRE Portal – Experiment Coordinator

Towards supporting the operations described in this usage scenario, the **5G-EPICENTRE Portal** ("*Portal*", described in D3.2) consumes APIs exposed to it by the **Experiment Coordinator** ("*Coordinator*")**.**

4.2.2.1 Experiment Run API

This API is used to send an experiment descriptor to the *Coordinator*. The API documentation is summarised in Table 8. The response for this API contains the execution identifier of the experiment that has been queued for execution. If the query is not successful, the response contains a 404 error.

Table 8: API for creating and queueing a new experiment execution, based on the contents of an experiment descriptor

Experiment Run	
Method	POST
Endpoint	/experiment/run
Request Headers	None
Request Body	**descriptor** [Object] REQUIRED A JavaScript Object Notation (JSON) structure exchanged between the *Portal* and the *Coordinator* to create and queue a new experiment execution in one of the testbed infrastructures. Its fields are elaborated in D3.2 (Section 4.2.2 in that document).
Response	**200** The requested experiment was created and queued at the Coordinator. The response will contain the execution id of the experiment queued.
	404 A connection error has occurred with the Experiment Coordinator.

4.2.2.2 Experiment Cancel API

This API us used for the elimination of an experiment in execution, identified by its id. It is used to mark the selected execution for cancellation. The execution will be cancelled after finalization of the current task. The API documentation is summarised in Table 9. The response for this API contains a copy of the Experiment Descriptor (see D3.2) that was used to define the execution of the experiment. If the query is not successful, the response contains the corresponding error code and message.

Table 9: API for marking the experiment for cancellation, specified by "id"

Experiment Cancel	
Method	DELETE
Endpoint	/experiment/:id/cancel
Request Headers	None
Request Body	None
Response	**200** The query was successful.
	404 Communication with the Experiment Coordinator has not been possible.
	500 The experiment Id in execution or in standby was not found.

4.2.3 Experiment Coordinator – Network Service Repository

Once it receives the instructions from the Portal in the form of a descriptor (see Section 4.2.2.1), the Coordinator receives the necessary artefacts from Repository's OpenAPI (proxy) server, using the Get File APIs described in Section 4.2.1.2.

4.2.4 Experiment Coordinator – Karmada API Server

After receiving the Helm chart, the Coordinator communicates with the Karmada federation, for deployment of the artefacts on each testbed (see Section 3).

4.3 Experiment deployment

In the 'Experiment deployment' scenario, the Experiment Coordinator has created and queued a new experiment execution in one of the testbed infrastructures using the information obtained by the 5G-EPICENTRE Portal in the descriptor exchange (Section 4.2.2.1). Alongside information on the artefacts and configuration parameters facilitating the deployment of the vertical application under the specifically desired test conditions, a time window (*i.e.*, a start and end date for the experiment) is specified for each experiment request, in which the Experiment Coordinator's internal scheduling components (see D2.5) query the testbed for resources' availability. If the Coordinator is unable to schedule the experiment between the start date and the end date, it will terminate the experiment.

Architecture update: to address problems when communicating the Coordinator with the different Remote iPerf Agents (probes) deployed in the different testbeds (*i.e.*, all the probes of the different testbeds should be stored in a unique 5G Traffic Simulation Manager ("Traffic Manager"), and each time the IP assigned to one of those probes changes it must also be notified to the Traffic Manager, which can be problematic). By having each testbed host an instance of the Traffic Manager instead of the Backend Layer, the Experiment Coordinator will only have to set up connections to the different Traffic Manager instances, and the communication with the probes (*e.g.*, UEs, *etc.*) will instead depend on each testbed.

This downstream information flow (see Figure 12) supporting this usage scenario involves primarily the integration of the Back-end Layer (*i.e.*, Experiment Coordinator) and of the Infrastructure Layer components (*i.e.*, Traffic Simulation Manager, Publisher).

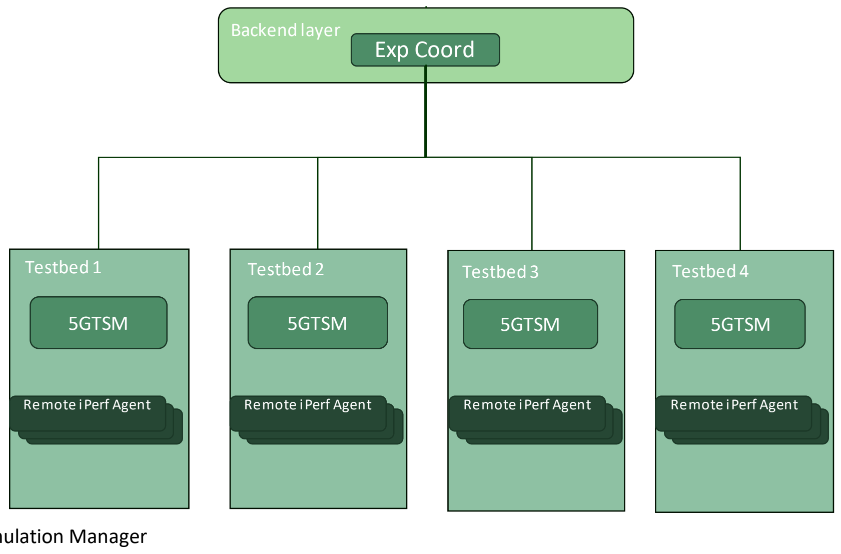


Figure 12: 5G-EPICENTRE component integrations supporting the "Experiment deployment" usage scenario

The following paragraphs describe all APIs used to connect the various entities in the 5G-EPICENTRE architecture for facilitating this usage scenario.

4.3.1 Experiment Coordinator – 5G Traffic Simulation Manager

The **Experiment Coordinator** (“*Coordinator*”, described in D2.5) consumes APIs exposed to it by the **5G Traffic Simulation Manager** (“*Traffic Manager*”, described in D2.5) exposed on each testbed, to pass the parameters needed for traffic generation.

4.3.1.1 Start API

This API executes an iPerf command with its parameters on the Agent identified by the “id” field in the “request_body” parameter. This is intended to be the normal way to start a server/client on a remote agent. The API documentation is summarised in Table 10. The response for this API contains a JSON with the metadata required, as well as the iPerf parameters for traffic generation. If the query is not successful, the response contains the corresponding error code and message.

Table 10: API for starting traffic generation on a remote iPerf Agent specified by the “id” parameter inside “request_body”

Start	
Method	POST
Endpoint	/start
Request Headers	None
Request Body	**netapp_id** [String] OPTIONAL Id for vertical system under test identification.
	origin [String] REQUIRED Origin of the measures. Potential acceptable values are "UE", "RAN", "5GC", "EPC", "main data server", or "edge".
	userId [String] REQUIRED User identification.
	experiment_id [Number] REQUIRED Experiment identifier, provided by the Experiment Coordinator.
	publish [Boolean] REQUIRED Indicates whether the generated traffic measurements should be published in the RabbitMQ queue or not.
	request_body [Object] REQUIRED The request body contains the data necessary to identify and configure the agents. Its contents are described in Table 15.
Response	**200** The response will contain a JSON Object with metadata and iPerf parameters for traffic generation.

	404 Indicates that the connection to the 5G Traffic Simulator Manager has not been possible.
	500 Indicates that there was an error in the input format.

4.3.1.2 Stop API

This API stops an iPerf server running on a remote Agent, identified by a dictionary with the keyword "agent_id" and value "id". The API documentation is summarised in Table 11. The response for this API contains a JSON with the necessary information for the identification of the agent to be stopped. If the query is not successful, the response contains the corresponding error code and message.

Table 11: API for terminating traffic generation on a remote iPerf Agent

Stop	
Method	POST
Endpoint	/stop
Request Headers	None
Request Body	**userId** [String] REQUIRED User identification.
	request_body [Object] REQUIRED The request body contains the necessary to identify the agent to be stopped. Its contents are described in Table 15.
Response	**200** The Agent specified by the "agent_id" keyword has stopped executing.
	404 Indicates that the connection to the 5G Traffic Simulator Manager has not been possible.
	500 Indicates that there was an error in the input format.

4.3.2 Experiment Coordinator – Publisher

The **Experiment Coordinator** ("*Coordinator*", described in D2.5) consumes APIs exposed to it by the **Publisher** architectural block exposed on each testbed, for metadata assignment.

4.3.2.1 Fetch Metrics API

This API is used to fetch new Prometheus client data, if available. Data will be fetched until the "end-time" is reached or until the Coordinator stops the search. The API documentation is summarised in Table 12. The response for this API contains a JSON, reporting the measures requested from the Prometheus client. If the query is not successful, the response contains the corresponding error code and message.

Table 12: API for fetching metrics from the Publisher, using the Prometheus client

Fetch Metrics	
Method	POST
Endpoint	/fetch_metrics

Request Headers	None
Request Body	**metrics** [List] REQUIRED Contains a list of the parameters needed to obtain the metrics. These parameters are: • **query** [String] REQUIRED A valid query for the Prometheus client. • **unit** [String] OPTIONAL Unit in which the measures are to be received. • **origin** [String] REQUIRED Indicates the origin of the measurements.
	end_time [Date] OPTIONAL Indicates until when to receive measurements.
	interval [Int] OPTIONAL Indicates at what interval to receive measurements.
Response	**200** The operation was successfully executed, the response containing the measures requested from the Prometheus client.
	404 Indicates that the connection to the Publisher has not been possible.
	500 Indicates that there was an error in the input format.

4.3.2.2 Add Experiment API

This API adds a JSON containing metadata for the Publisher to store. When the corresponding experiment results are read from the MQTT, they will be filled with this metadata. The API documentation is summarised in Table 13. The response for this API contains a message confirming that the experiment metadata has been added. If the query is not successful, the response contains the corresponding error code and message.

Table 13: API for adding metadata for an experiment to the Publisher

Add Experiment	
Method	POST
Endpoint	/add_experiment
Request Headers	None
Request Body	**experiment_id** [Number] REQUIRED The id of the experiment.
	testbed_id [Number] REQUIRED The id of the testbed where it is going to be executed.
	scenario_id [Number] REQUIRED The id of the scenario where the experiment is to be run.

	use_case_id [Number] REQUIRED The id of the use case to which the experiment belongs.
	netapp_id [String] REQUIRED The id of the vertical system under test to which the experiment Belongs.
Response	**200** The operation was successfully executed. The response will contain a message saying that the experiment metadata has been added.
	404 Indicates that the connection to the Publisher has not been possible.
	500 Indicates that there was an error in the input format.

4.3.2.3 Remove Experiment API

This API removes the metadata of a given experiment stored in the Publisher. The API documentation is summarised in Table 14. The response for this API contains a message confirming that the experiment metadata has been removed. If the query is not successful, the response contains the corresponding error code and message.

Table 14: API for removing metadata for an experiment from the Publisher

Remove Experiment	
Method	POST
Endpoint	/remove_experiment
Request Headers	None
Request Body	**experiment_id** [Number] REQUIRED The id of the experiment.
	testbed_id [Number] REQUIRED The id of the testbed where it is going to be executed.
	scenario_id [Number] REQUIRED The id of the scenario where the experiment is to be run.
	use_case_id [Number] REQUIRED The id of the use case to which the experiment belongs.
	netapp_id [String] REQUIRED The id of the vertical system under test to which the experiment Belongs.
Response	**200** The operation was successfully executed. The response will contain a message saying that the experiment metadata has been removed.
	404 Indicates that the connection to the Publisher has not been possible.
	500 Indicates that there was an error in the input format.

4.3.3 5G Traffic Simulation Manager - Remote iPerf Agent

The 5G Traffic Simulation Manager (“*Traffic Manager*”) consumes APIs exposed to it by one or more **Remote iPerf Agents** (*“Agents”*), which are responsible for simulated traffic generation.

4.3.3.1 iPerf API

This API executes iPerf process with the specific parameters specified in the request body in JSON format. The API documentation is summarised in Table 15. The response for this API contains a JSON Object, reporting the success of the execution request. If the query is not successful, the response contains the corresponding error code and message.

Table 15: API for executing iPerf Agent operations

iPerf	
Method	POST
Endpoint	/Iperf
Request Headers	None
Request Body	**agent_id** [String] REQUIRED Identifier of the agent involved in traffic generation.
	action [String] REQUIRED Action to be performed by the agent.
	parameters [Object] REQUIRED A JSON Object which includes the agent configuration parameters based on the iPerf tool. There will be one entry for each configuration flag.

Response	**200** The operation was successfully executed. The response will contain a JSON, reporting the success of the execution, and a message, *e.g.*:
	`{`
	`“Status”:”Success”,`
	`“Message”:”Successfully executed iPerf <mode>”`
	`}`

	404 Indicates that the connection to the Remote iPerf Agent has not been possible.
	500 Indicates that there was an error in the input format.

4.3.3.2 Close API

This API terminates an iPerf process. The API documentation is summarised in Table 16. The response for this API contains a JSON Object, reporting the success of the termination request. If there has been a connection failure, the response contains a 404 error.

Table 16: API for terminating an iPerf Agent operation

Close	
Method	GET
Endpoint	/Close
Request Headers	None
Request Body	None

Response	**200** The query was successful. The response will contain a JSON reporting the success of the execution and a message:
	`{`
	`“Status”:”Success”,`
	`“Message”:”Successfully closed iPerf”`
	`}`

	404 Indicates that the connection to the Remote iPerf Agent has not been possible.

4.3.3.3 Last JSON Result API

This API is used to retrieve the result of the previous iPerf execution. The API documentation is summarised in Table 17. The response for this API contains a JSON Object with the list of desired results. If there has been a connection failure, the response contains a 404 error.

Table 17: API for retrieving the result of the last iPerf Agent operation

Retrieve	
Method	GET
Endpoint	/LastJsonResult
Request Headers	None
Request Body	None

Response	**200** The query was successful. The response will contain a JSON reporting the success of the retrieval, a message and a list of Results (dictionary with parsed results):
	`{`
	`“Status”:”Success”,`
	`“Message”:”Successfully retrieved last raw result”,`
	`“Result”:<Parameters Dict>`
	`}`

	404 Indicates that the connection to the Remote iPerf Agent has not been possible.

4.4 Experiment monitoring during execution

In this scenario, the experimenter can monitor platform-generated metrics and statistical information in real time, and through explanatory data visualizations, generated automatically at the start of the vertical application deployment. Insights generated and charted into graphics include experimental condition indicators, network traffic metrics, KPIs and detected anomalies.

The upstream information flow (see Figure 13) supporting this usage scenario involves integration of Front-end (*i.e.*, 5G-EPICENTRE Portal, Back-end, (*i.e.*, Analytics Aggregator, Experiment Coordinator), and Infrastructure Layer components (*i.e.*, Analytics Engine, Publisher, testbed infrastructure).

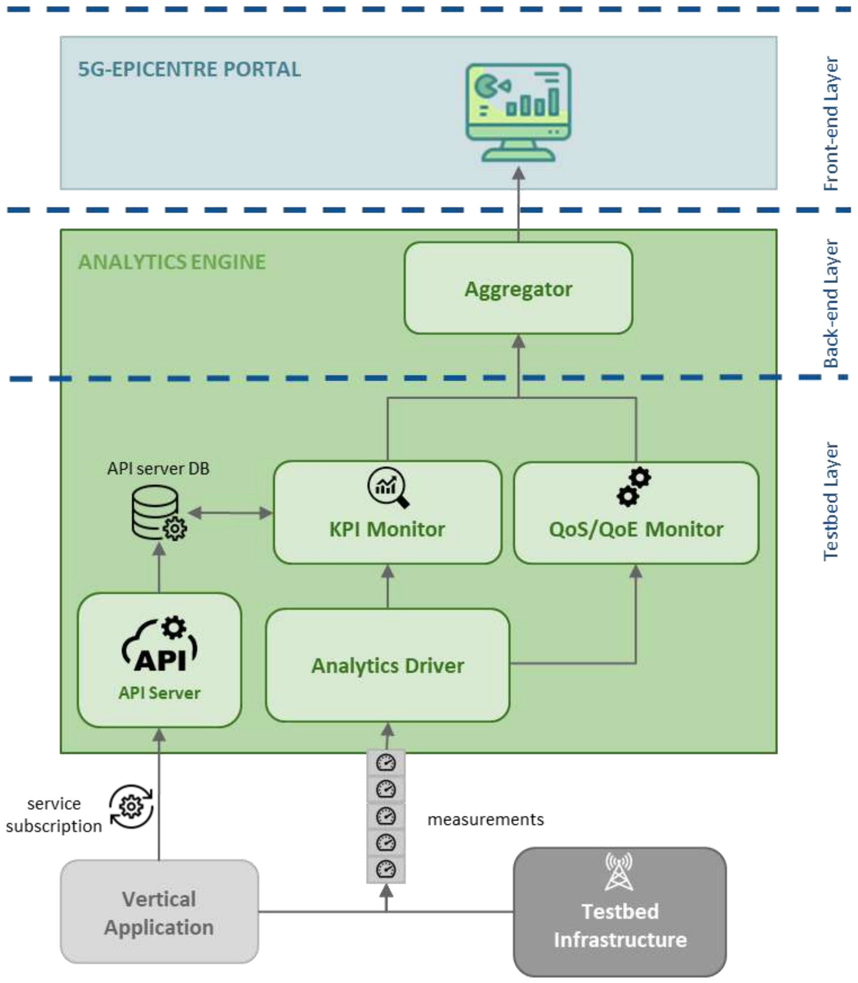


Figure 13: 5G-EPICENTRE component integrations supporting the "Experiment monitoring during execution" usage scenario

The following paragraphs describe all APIs used to connect the various entities in the 5G-EPICENTRE architecture for facilitating this usage scenario.

4.4.1 5G-EPICENTRE Portal – Experiment Coordinator

Towards supporting the operations described in this usage scenario, the **5G-EPICENTRE Portal** ("*Portal*", described in D3.2) consumes APIs exposed to it by the **Experiment Coordinator** ("*Coordinator*")**.**

4.4.1.1 Get Logs API

This API is used to obtain the logs generated in the execution of an experiment identified by its id. The API documentation is summarised in Table 18. The response for this API contains a JavaScript Object Notation Object (JSON) that contains all the log messages generated by the execution, separated by stage, as follows (refer to D2.5 for more information on the different stages):

- **"Status":** Either *"success"* or *"Not Found"*;
- **"PreRun":** Messages generated during the *Pre-Run* stage;
- **"Execution":** Messages generated during the *Run* stage;
- **"PostRun":** Messages generated during *Post-Run* stage.

If the query is not successful, the response contains a 404 error.

Table 18: API for obtaining the logs generated in the execution of an experiment specified by "id"

Get Logs	
Method	GET
Endpoint	/experiment/:id/logs
Request Headers	None
Request Body	None
Response	**200** The query was successful. The response will contain a JSON containing all log messages.
	404 Indicates that the connection to the Experiment Coordinator has not been possible.

4.4.1.2 Get Descriptor API

With this API it is possible to obtain the descriptor that was used to define the experiment, identified by its id. The API documentation is summarised in Table 19. The response for this API contains a copy of the Experiment Descriptor (see D3.2) that was used to define the execution of the experiment, or a 404 error, if the query is not successful.

Table 19: API for obtaining the experiment descriptor for the experiment specified by "id"

Get Descriptor API	
Method	GET
Endpoint	/experiment/:id/descriptor
Request Headers	None
Request Body	None
Response	**200** The query was successful. The response will contain the Experiment Descriptor for this experiment.
	404 Indicates that the connection to the Experiment Coordinator has not been possible.

4.4.2 Analytics Engine – Analytics Aggregator – 5G-EPICENTRE Portal

The 5G-EPICENTRE platform includes the Analytics Engine module, which is in charge of collecting and analysing data coming from the infrastructure and from the vertical applications under test. The module performs KPI calculation on experimental data, statistical analysis and Artificial Intelligence (AI)-based functionalities, such as anomaly detection.

According to the 5G-EPICENTRE standard validation methodology (described in deliverable D1.6 *"Experiment evaluation strategy and experimentation plan"*), KPI calculation is performed on several measurements collected over a specified number of iterations. It integrates a statistical analysis service, which provides indicators of the overall experiment (*i.e.*, minimum, maximum, median, standard deviation and quartiles). The outputs of the Analytics Engine are then used by the 5G-EPICENTRE Portal, to provide a visual representation of the experimentation results.

This upstream information flow corresponds to the orchestration described in D3.2 and is integrated within the platform by means of asynchronous messaging communication based on the MQTT protocol and a RabbitMQ message-oriented middleware. The orchestration of this flow is illustrated in Figure 14.

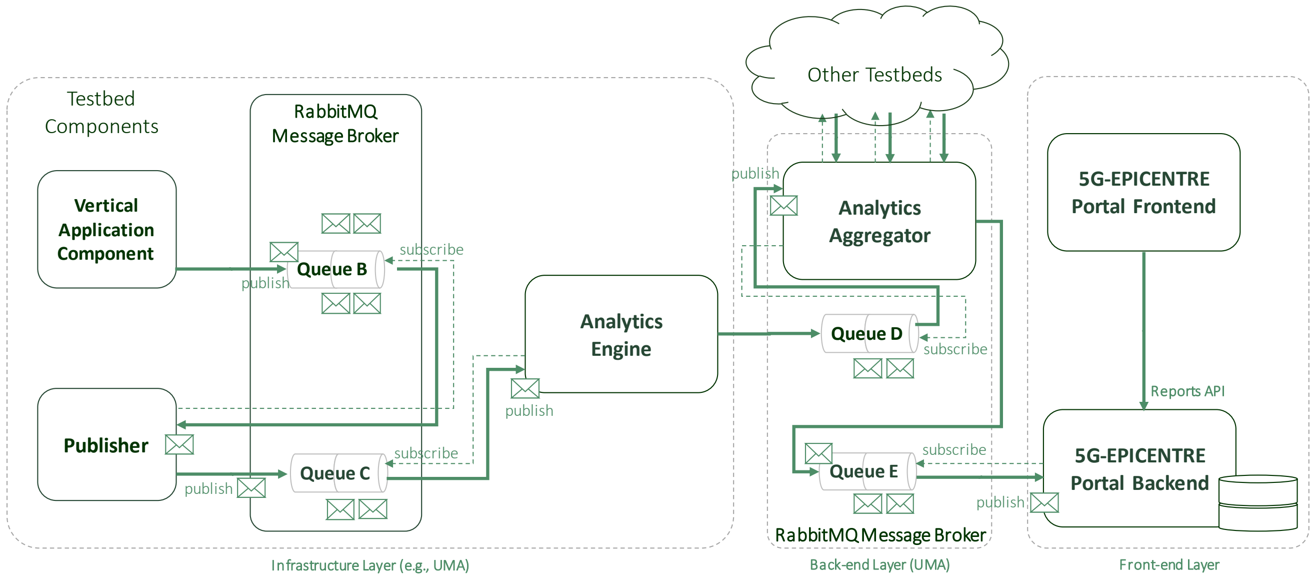


Figure 14: 5G-EPICENTRE Upstream Information Flow orchestration

4.4.2.1 Publisher – Analytics Engine information flow

The Publisher dispatches messages containing network and experiment metrics towards the Analytics Engine components at each testbed using an asynchronous communication mode. The interface implements an MQTT client for the exchange of messages including a JSON payload. Table 20 below lists the information flow exchanged between the Publisher and the Analytics Engine during the metrics publishing operation.

Table 20: Asynchronous information flow exchanged between the Publisher and the Analytics Engine

Publisher routed messages consumed by the Analytics Engine	
Method	Publish
Direction	Publisher → Analytics Engine
Message standard	MQTT
Message format	**category** [String] REQUIRED The category of the metrics the Publisher sends to the Analytics engine, *e.g.*, *"5g_network"* (network measurement); *"experiment"* (experiment measurements).
	testbed_id [Integer] REQUIRED

	The testbed identifier (as specified in the federation) where the metrics are coming from.
	scenario_id [Integer] OPTIONAL The specific scenario for which metrics are collected.
	use_case_id [Integer] OPTIONAL The specific use case for which metrics are collected.
	experiment_id [Integer] OPTIONAL The specific experiment for which metrics are collected.
	netapp_id [String] OPTIONAL The specific vertical system under test, for which metrics are collected.
	device_id [String] OPTIONAL The identifier of the UE providing the metrics.
	data [Object] REQUIRED Array of data blocks containing collected measurements. These data blocks will include a set of mandatory fields (*e.g.*, *type*, *timestamp*, *origin*, *etc*.) and custom fields (*e.g.*, key-value pairs from native metrics).

4.4.2.2 Analytics Engine – Analytics Aggregator information flow

The Analytics Engine dispatches messages containing various types of data (calculated KPIs, detected anomalies, statistics over experiments/infrastructure metrics and specific Network Application metrics – see Section 4.5.2) towards the Analytics Aggregator module deployed at the Back-end Layer via a dedicated queue. The interface between the modules implements an MQTT client for the exchange of messages including a JSON payload. Table 21 below lists the information flow exchanged between the Analytics Engine and the Analytics Aggregator.

Table 21: Asynchronous information flow exchanged between the Analytics Engine and the Analytics Aggregator

Analytics Engine routed messages consumed by the Analytics Aggregator	
Method	Publish
Direction	Analytics Engine → Analytics Aggregator
Message standard	MQTT
Message format	**category** [String] REQUIRED The category of the metrics the Publisher sends to the Analytics engine, *e.g.*, *"5g_network"* (network measurement); *"experiment"* (experiment measurements).
	testbed_id [Integer] REQUIRED The testbed identifier (as specified in the federation) where the metrics are coming from.
	scenario_id [Integer] OPTIONAL The specific scenario for which metrics are collected.

	use_case_id [Integer] OPTIONAL The specific use case for which metrics are collected.
	experiment_id [Integer] OPTIONAL The specific experiment for which metrics are collected.
	netapp_id [String] OPTIONAL The specific vertical system under test, for which metrics are collected.
	device_id [String] OPTIONAL The identifier of the UE providing the metrics.
	data [Object] REQUIRED Array of data blocks containing collected measurements. These data blocks will include a set of mandatory fields (*e.g.*, *type*, *timestamp*, *origin*, *etc*.) and custom fields (*e.g.*, key-value pairs from native metrics).
	type [String] REQUIRED If *'params',* the message contains statistics on measurements coming from the testbed infrastructure. If *'kpis',* the message contains KPIs calculated for the experiments. If *'anomalies',* the message contains detected anomalies. Finally, if *'hspf',* the message contains metrics coming from the Network Intrusion & Detection Network Application (see Section 4.5.2).

The Analytics Aggregator dispatches messages received from the Analytics Engine instances on all the testbeds towards the 5G-EPICENTRE Portal Backend via a dedicated queue. The interface between the modules implements an MQTT client for the exchange of messages including a JSON payload (see also D3.2). The information flow exchanged between the Analytics Aggregator and the Portal Backend is the one described in Table 21.

4.5 Platform Network Applications integration

Whilst scheduling an experiment, the 5G-EPICENTRE platform optionally exposes platform Network Applications for exerting control over network control plane functions, testbed entities and (security) policies. Each Network Application creates additional, optional parameters in the experiment descriptor, declaring specification for deploying additional Helm charts alongside the vertical application one. Thereby, experimenters can easily and effortlessly configure the network to fit their needs during the experiment execution. For a complete list of available Network Applications, Network and Application Functions, see D4.2 *"Network functions implementation"*. With use of these tools, developers can easily and effortlessly chain their application to a variety of pre-deployed services offered to them, as well as configure the network to fit their needs during the experiment execution. In the below paragraphs, we elaborate on the integration of the D1.4-listed *'Platform Network Applications'*. This refers to:

- Prioritising traffic flows and/or guarantee QoS (**Configurator Network Application**, Section 4.5.1);
- Detecting and reacting to outside malicious interference (**Network Intrusion Detection Network Application**, Section 4.5.2);
- Obtaining custom visual reports of experimenter-specific KPIs directly on the Portal (**Analytics Services Network Application**, Section 4.5.3);

4.5.1 Configurator and Network Configuration Dashboard

The Northbound Configuration Dashboard (NCD) is a network application specifically designed to empower experimenters with the ability to request and configure specific Quality of Service (QoS) parameters based on the 5QI (5G QoS Indicator) standards for running their experiments. The NCD consumes the Npcf_PolicyAuthorization Service APIs[3], that are exposed by the testbed 5G Core components.

The approach on mapping of request body parameters is to map the selected 5QI value to the ‘afAppId’ in request body, *e.g.*, ‘afAppId = Id_X’ for 5QI X. Testbed partners will then map the afAppId value with specific network configurations on the testbed.

4.5.2 Network Intrusion & Detection

The Holistic Security and Privacy Framework (HSPF) corresponds to the security approach developed in the context of 5G-EPICENTRE, and aims at providing security to any Vertical, or Network Application being executed in a K8s environment (see D2.7 for further details). This Network Application (*Network Intrusion & Detection*) has been integrated with several 5G-EPICENTRE components, as described hereafter.

Table 22 presents the reporting message sent by the Network Intrusion & Detection Network Application components to the Analytics Engine components every time a classification routine is executed. Two similar messages are sent: one reporting on the number of **total flows** and another on the number of **malicious flows** detected.

Table 22: Network Intrusion & Detection Network Application report message schema (left) and example (right).

```
{
    “category”: “String”,
    “testbed_id”: “Number”,
    “netapp_id”: “String”,
    “data”: [
        {
            “type”: “String”,
            “origin”: “String”,
            “value”: “Number”,
            “network_application”: “String”,
            “micro_service”: “String”,
            “hspf_agent_id”: “String”,
            “timestamp”: “Number”,
        }
    ]
}
```

```
{
    “category”: “experiment”,
    “testbed_id”: 1,
    “netapp_id”: “hspf_netapp”,
    “data”: [
        {
            “type”: “malicious_flows”,
            “origin”: “main_data_server”,
            “value”: 15,
            “network_application”: “UC4”,
            “micro_service”: “message-broker”,
            “hspf_agent_id”: “UC4-message-broker”,
            “timestamp”: 1687532092599,
        }
    ]
}
```

As with any other Network Application messages (*i.e.*, KPI messages), all the HSPF messages are published to the *application* topic of the Publisher component, then consumed by the Analytics Engine, and ultimately reach the Portal, where the implemented visualization tools represent the underlying values (see also D3.2).

Table 23 provides further information on the message content, by identifying the possible values and some observations for each of the fields.

[3] https://jdegre.github.io/editor/?url=https://raw.githubusercontent.com/jdegre/5GC_APIs/master/TS29514_Npcf_PolicyAuthorization.yaml

Table 23: HSPF message fields - possible values and observations

Field	Possible values	Observations
category	“experiment”	This field is required for the Publisher to be able to handle these messages.
testbed_id	[1,2,3,4]	This value is used to identify the testbed where the HSPF components are being executed.
netapp_id	“hspf_netapp”	This field allows the remaining 5G-EPICENTRE components to clearly identify the origin of this messages. This value has been agreed with the respective partners.
data.type	[“malicious_flows”, “total_flows”]	This field represents the type of traffic that the message reports on.
data.origin	“main_data_server”	This field is part of the mandatory fields of the “experiment” kind of messages, as previously defined (namely in D4.1).
data.value	[0-∞]	This value corresponds to the number of flows that the message reports on.
data.network_application	“UC[0-8]” “TP[0-∞]”	Used to identify the underlying UC. May also be used to identify a third-party experimenter solution.
data.micro_service	“microservice_name”	Identifies the micro-service that the HSPF components are protecting.
data.hspf_agent_id	“UC[0-8]-micro-service_name” “TP[0-∞]-micro-service_name”	Field used to uniquely identify the HSPF agent originating the respective message.
data.timestamp	<timestamp_value>	UNIX format timestamp used to identify the origin time of the message.

Figure 15 provides a high-level overview of the interaction between an arbitrary Vertical, or Network Application; the Network Intrusion & Detection Network Application; and the 5G-EPICENTRE components. This Figure displays the User Equipment (UEs) and custom solutions of a generic Use Case (UC), or third-party experimenter’s vertical application, their interaction with the K8s cluster (located in one of the four 5G-EPICENTRE testbeds) and how the services of the aforementioned solution are monitored and protected by the HSPF components. The HSPF then generates reporting information, which is shared with the Analytics Engine and the Portal.

Additional information and more messages can be reported by the HSPF components, as described in D2.7.

4.5.3 Analytics

The vertical application can consume the APIs exposed by the Analytics Network Application, in order to subscribe to the aforementioned (see Section 4.4.2) services for KPI calculation and statistical analysis.

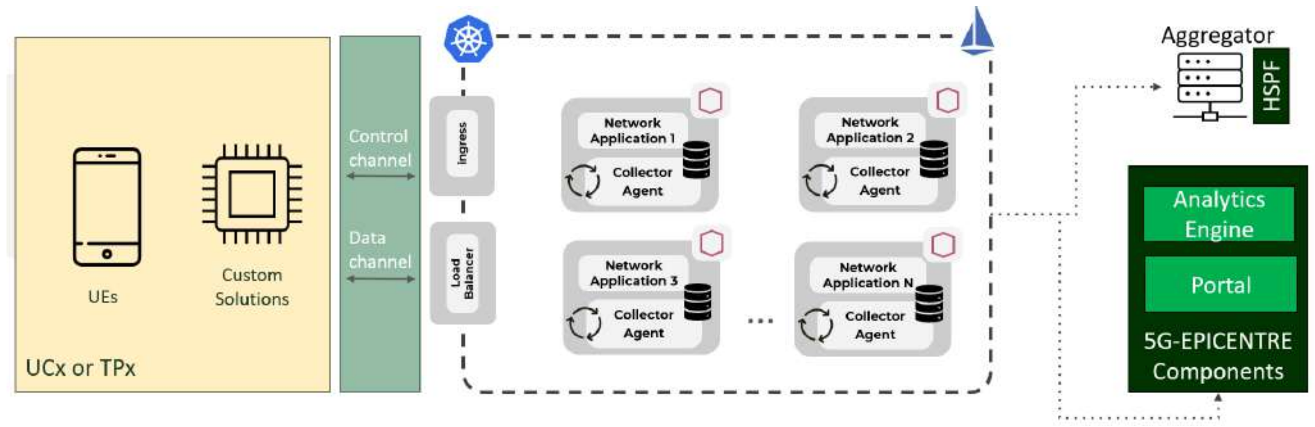


Figure 15: UCx, HSPF/Network Intrusion & Detection Network Application and 5G-EPICENTRE components.

Additional information and message examples will be described in D2.6 *"5G-EPICENTRE Analytics Engine"*.

4.5.3.1 Subscription API

This API is used to send a subscription request for the analytics services to the Analytics Network Application, and provide basic information for KPI calculation. The API documentation is summarised in Table 24. The response for this API contains the subscription id of the request. If the request is not successful, the response contains the corresponding error code and message.

Table 24: API for subscribing the analytics service based on the content of a KPI descriptor

Subscription	
Method	POST
Endpoint	/kpi_service_subscribe
Request Headers	None
Request Body	**descriptor** [Object] REQUIRED A JSON structure exchanged between the subscriber and the Analytics Network Application, to request KPI calculation and statistical services for the set of measurements specified in the JSON. It contains specific information about the *testbed_id*, the *experiment_id*, the *netapp_id* and the list of measurements to be collected during the experiment (examples and more details will be provided in D2.6). For each measurement to be used for KPI calculation, according to the standard methodology (see D1.6), the JSON specifies the expected number of iterations, the number of measurements per iteration, and (optionally) the "upgradable" and "optimal" thresholds.
Response	**200** The subscription was accepted. The response will contain the status "ok" and the subscription_id.
	400 The format of the descriptor is not valid.
	500 The server cannot process the request for an unknown reason.

4.5.3.2 Delete Subscription API

This API is used to delete the subscription to the Analytics services. The API documentation is summarized in Table 25. If the request is not successful, the response contains the corresponding error code and message.

Table 25: API for deleting the subscription to the analytics services identified by “subscription_id”

Delete Subscription	
Method	GET
Endpoint	/kpi_service_delete/:subscription_id
Request Headers	None
Request Body	None
Response	**200** The deletion was successful.
	400 The subscription_id was not found.
	500 The server cannot process the request for an unknown reason.

4.5.3.3 Get Subscription API

This API is used to retrieve the information associated with “subscription_id”. The API documentation is summarised in Table 26. The response will contain the descriptor of the subscription, a JSON structure containing the information about the measurements associated with the provided subscription_id. If the request is not successful, the response contains the corresponding error code and message.

Table 26: API for returning the descriptor associated with “subscription_id”

Get Subscription	
Method	GET
Endpoint	/kpi_service_get/:subscription_id
Request Headers	None
Request Body	None
Response	**200** The query was successful. The response will contain the information associated with “subscription_id”, structured in a JSON format.
	404 “subscription_id” not found.
	500 The server cannot process the request for an unknown reason.

5 Conclusions

This deliverable has elaborated on the final version of the 5G-EPICENTRE integrated experimentation facility, describing its component interactions in terms of platform usage scenarios. In addition, the deliverable has de-mystified the implementation details on how to achieve cross-testbed resource aggregation, orchestration and synchronization under a typical Karmada control plane architecture.

The document has elaborated on the core information flow streams (from top-to-bottom and bottom-up) which enables it to address key functional requirements (listed in D1.3 and D1.4), whilst enabling the technical partners to attain a common understanding and groundwork for how components come together to form the whole en-visioned platform.

During the project period M19-M30 (*i.e.*, since D4.4) significant progress has been achieved in terms of both individual developments and integration, whereby, because of the modular approach to the integration, the technical components of the project have easily been re-framed to fit external developments to the project (most notably, the new Network Application co-definition with other ICT-41 projects). 5G-EPICENTRE now fully con-templates 5G innovations for PPDR through the Network Applications development paradigm described in the latest 5G-PPP Network Application white papers (version 1, September 2022 and version 2, 2023 – to appear).

Specifically, the 5G-EPICENTRE experimentation facility offers complete support, and an implementation para-digm of the 'Hybrid' network application interaction approach, allowing verticals to exploit enablers for **ad-vanced service operations**; **usability of the 5G capabilities**; and **adjustment of networks** for PPDR needs. Draw-ing on the concrete reference architecture for an experimentation platform for interactions between the vertical and the testbed operators (D1.4), a successful demonstration of the envisioned platform was achieved in M30. Through this paradigm, different 5G standalone system configurations, each with integrated support for a K8s management environment, were federated over Karmada, achieving higher availability of services. To the best of our knowledge, and given that Karmada, as a solution has evolved in tandem to the 5G-EPICENTRE lifetime (Karmada v1.0.0 was released in December 2021 – M12 of the project), this is one of the first implementations of 5G testbed resource aggregation under a uniform K8s management system.

The goal of this report is to deliver on the final integrated vision of the 5G-EPICENTRE interconnected modules, and thus carry on with the technical integration work necessary to support third-party experimentation (in WP5), as well as undertake any component interface, or functionalities refinement, in accordance with those needs. Any such action shall be reported in D4.7.

References

[1] *Karmada: Open, Multi-Cloud, Multi-Cluster Kubernetes Orchestration*. Available at: https://github.com/karmada-io/karmada.

[2] Tabatabaeimehr, F., Khalili, H. Requena, M., Kahvazadeh, S., & Mangues-Bafalluy, J. (2023, July). Dynamic service placement in 6G multi-cloud scenarios. In *23rd International Conference on Transparent Optical Networks (ICTON 2023)*. Available at https://icton2023.upb.ro/icton-2023-proceedings/.

[3] Gokhale, S., Poosarla, R., Tikar, S., Gunjawate, S., Hajare, A., Deshpande, S., et al. (2021, September). Creating helm charts to ease deployment of enterprise application and its related services in kubernetes. In *2021 International Conference on Computing, Communication and Green Engineering (CCGE)* (pp. 1-5). IEEE. doi: https://doi.org/10.1109/CCGE50943.2021.9776450