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

Innovation Action – ICT-41-2020 - 5G PPP – 5G

Innovations for verticals with third party services

D2.3: Cloud-native services containerization

Delivery date: April 2023

Dissemination level: Public

Project Title:	**5G-EPICENTRE - 5G ExPerimentation Infrastructure hosting Cloud-nativE Netapps 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.

Document Information

Deliverable	D2.3: Cloud-native services containerization
Work Package	WP2: Cloud-native 5G NFV
Task(s)	T2.2: Cloud-native services containerization
Type	Other
Dissemination Level	Public
Due Date	M28, April 30, 2023
Submission Date	M28, April 30, 2023
Document Lead	Almudena Díaz Zayas (UMA)
Contributors	Jorge Márquez Ortega (UMA) Daniel del Teso (NEM) Daniele Ronzani (ATH) Alain Dubois (ADS) André Gomes (ONE) Pedro Tomás (ONE) Vitor Fonseca (ONE) Antonio Zanesco (YBQ) Antonis Protopsaltis (ORAMA) Hamzeh Khalili (CTTC) Rainer Wragge (OPTO) Ankur Gupta (HHI) Kirsten Krüger (HHI) Holger Gäbler (HHI)
Internal Review	Nicola di Prieto (ATH) Daniel del Teso (NEM)

Document history

Version	Date	Changes	Contributor(s)
V0.1	12/01/2023	Initial deliverable structure	Almudena Díaz Zayas (UMA)
V0.2	27/02/2023	50% of the deliverable content	Almudena Díaz Zayas (UMA) Daniel del Teso (NEM) Pablo Garrido (NEM) Daniele Ronzani (ATH) Alain Dubois (ADS)
V0.3	16/03/2023	90% of the deliverable content	Almudena Díaz Zayas (UMA) André Gomes (ONE) Pedro Tomás (ONE) Vitor Fonseca (ONE) Antonio Zanesco (YBQ) Antonis Protopsaltis (ORAMA) Hamzeh Khalili (CTTC) Rainer Wragge (OPTO) Ankur Gupta (HHI)
V1.0	13/04/2023	Internal Review Version	Almudena Díaz (UMA) Daniel de Teso (NEM) Pablo Garrido (NEM) Daniele Ronzani (ATH) Alain Dubois (ADS) André Gomes (ONE) Pedro Tomás (ONE) Vitor Fonseca (ONE) Antonio Zanesco (YBQ) Antonis Protopsaltis (ORAMA) Hamzeh Khalili (CTTC) Rainer Wragge (OPTO) Ankur Gupta (HHI)
V1.1	17/04/2023	1st version with suggested revisions	Nicola Di Prieto (ATH) Daniel de Teso (NEM)
V1.2	25/04/2023	First revisions after internal review	Almudena Díaz Zayas (UMA)
V1.5	26/04/2023	Quality review: copyediting; proofreading; formatting	Konstantinos C. Apostolakis (FORTH)
V2.0	27/04/2023	Final version for submission	Almudena Díaz Zayas (UMA)

Project Partners

Logo	Partner	Country	Short name
	AIRBUS DS SLC	France	**ADS**
	NOVA TELECOMMUNICATIONS SINGLE MEMBER S.A.	Greece	**NOVA**
	Altice Labs SA	Portugal	**ALB**
	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.	Germany	**HHI**
	Foundation for Research and Technology Hellas	Greece	**FORTH**
	Universidad de Malaga	Spain	**UMA**
	Centre Tecnològic de Telecomunicacions de Catalunya	Spain	**CTTC**
	Istella SpA	Italy	**IST**
	One Source Consultoria Informatica LDA	Portugal	**ONE**
	Iquadrat Informatica SL	Spain	**IQU**
	Nemergent Solutions S.L.	Spain	**NEM**
	EBOS Technologies Limited	Cyprus	**EBOS**
	Athonet SRL	Italy	**ATH**
	RedZinc Services Limited	Ireland	**RZ**
	OptoPrecision GmbH	Germany	**OPTO**
	Youbiquo SRL	Italy	**YBQ**
	ORamaVR SA	Switzerland	**ORAMA**

List of abbreviations

Abbreviation	Definition
5GC	5G Core
AMF	Access and Mobility Management Function
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AR	Augmented Reality
AUSF	Authentication Server Function
(C/P)AS	(Controlling/Participating) Application Server
ASF	Audio Signalisation Function
BK	BodyKit
BSF	Binding Support Function
CCC	Command and Control Centre
CDN	Content Delivery Network
CMS	Configuration Management Server
CNI	Container Network Interface
(I/P/S)CSCF	(Interrogating/Proxy/Serving) Call Session Control Function
CSI	Container Storage Interface
DB	Database
DBMS	Database Management System
DMF	Data Management Function
DN	Data Network
DNS	Domain Name System

ExaaS	Experiments as a Service
FQDN	Fully Qualified Domain Name
(F)HD	(Full) High Definition
HSS	Home Subscriber Server
GA	Grant Agreement
GMS	Group Management Server
IdMF	Identity Management Function
IMS	IP Multimedia Subsystem
IoT	Internet of Things
IP	Internet Protocol
K8s	Kubernetes
KMS	Key Management Server
KVM	Kernel-based Virtual Machine
KPI	Key Performance Indicator
KPIF	Key Performance Indicator Function
MANO	Management and Orchestration
MCPTT	Mission Critical Push-to-Talk
MCX	Mission Critical
MetalLB	Bare metal load-balancer for K8s
MQTT	Message Queueing (Telemetry Transport)
N/A	Not Applicable
(C/V)NF(C/M)	(Cloud-native/Virtual) Network Function (Component/Manager)
(C)NFV(I/O)	(Cloud-native) Network Functions Virtualization (Infrastructure/Orchestrator)
NFV	Network Functions Virtualization

NRF	Network Repository Function
NS	Network Service
NSSF	Network Slice Selection Function
OCI	Open Container Initiative
PDU	Protocol Data Unit
PCF	Policy Control Function
PTAggregate	Process for Data Aggregation
PTLive	Process for Live Data
PTS	Process To Save
PVC	Persistent Volume Claims
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
SA	Standalone
SBA	Service-Based Architecture
SCP	Service Communication Proxy
SD	Standard Definition
SDK	Software Development Kit
SDS	Software-Defined Storage
(U)SIM	(Universal) Subscriber Identity Module
SIP	Session Initiation Protocol
SMF	Session Management Function
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol

TLS	Transport Layer Security
UC	Use Case
UDP	User Datagram Protocol
UDR	Unified Data Repository
UE	User Equipment
(G)UI	(Graphical) User Interface
UDM	Unified Data Management
UL	Uplink
UPF	User Plane Function
(K)VM	(Kernel-based) Virtual Machine
VMF	Video Media Function
VPN	Virtual Private Network
VSF	Video Signalisation Function
WAN	Wide Area Network
WFS	Webfront Server
WP	Work Package

Executive summary

New broadband technologies require the constant evolution of the services that operate on them, to adapt and take advantage of each new technological leaps' benefits to their solutions. 5G technology is no exception. It is specifically conceptualized to work with virtualized solutions, enabling dynamic deployment models. These models are driving the proliferation of verticals, that must also embrace virtualization and dynamic deployment if they want to take advantage of all the benefits 5G has to offer. For this reason, 5G-EPICENTRE has opted for Cloud-Native solutions and the containerization of its services, to offer its users all the advantages that 5G technology can enable (while maintaining compatibility with other legacy solutions). Moreover, the containerization of services is optimal for providing robust and highly available Public Protection and Disaster Relief services.

Deliverable D2.3 is a self-contained document that collects the instructions and lessons learned during the deployment of the different use cases of the project in Kubernetes (K8s). The focus of this deliverable is to provide a clear understanding of the deployment of vertical solutions in containers on K8s.

This document includes an introduction to K8s, the requirements of the different use cases to be containerized, the K8s architecture of each one, and the instructions for their deployment. The document also provides a compilation of common issues and the lessons learned.

Table of Contents

List of Figures

List of Tables

1 Introduction

5G not only provides improved network capabilities (higher data rates; lower latencies; high efficiency in the use of resources; lower energy consumption; greater security), but also represents a paradigm shift in order to adapt to the new requirements of the numerous use cases that were taken into account during the design of 5G technology. Specifically, 5G networks are supported by software components and virtualization solutions. This new approach turns 5G networks into agile networks, which can be programmatically assembled and configured for specific use cases. Moreover, the same network infrastructure can host multiple virtual networks with different performance and characteristics, aimed at different types of users. These networks are easily reconfigurable, without the need for modifications or additional investments of the physical components that make up the infrastructure. Such vision has given rise to the adoption and usage of Virtual Network Functions (VNFs), and their decomposition into micro-services.

In order to propitiate this paradigm change, the 5G-EPICENTRE project has adopted the use of container-based virtualization technologies, with Kubernetes (K8s) being the chosen option as Virtual Infrastructure (VI) and VNF Manager (see also D1.3 and D1.4). Container-based virtualization technologies have demonstrated their benefits over the use of Virtual Machines (VMs), as they are more lightweight in terms on consumed resources and can unlock superior performance. Moreover, relating to the specific vertical addressed in this project – Public Protection and Disaster Relief (PPDR) – requirements are demanded that can be also satisfied with the usage of K8s.

PPDR requires robust and secure service deployments. In this context, enabling robust and highly available services is a primary goal for the project. A valid approach to offer robust services is by splitting the different functionalities into micro-services, which both ease the management of the whole system, and offer redundancy and highly available services. Containers are used to run these micro-services.

However, the management and coordination of micro-services and their containerization is not a trivial task. K8s is an open-source platform for managing containerized workloads and services, which facilitates the management of a large and dynamic ecosystems. K8s also provides a framework to run distributed systems resiliently, taking care of the scaling and failover processes. Moreover, it is widely available, being supported by the most common Linux distributions (RHEL, Ubuntu, *etc.*) and major public clouds. This deliverable describes the process to reach a successful deployment of all the project Use Cases' (UCs) services in such a virtualized environment.

The deliverable introduces the concept of K8s, the requirements of each one of the uses cases regarding their containerization and architecture adopted for the deployment of the uses cases and the instructions for the deployment. Finally, as conclusions, the main issues and learned lessons are provided.

1.1 Mapping of project's outputs

The purpose of this section is to map 5G-EPICENTRE Grant Agreement (GA) commitments within the formal 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
T2.2: Cloud-native services containerization "This task will focus on the integration of edge computing in the 5GEPICENTRE	Section 2 – Use Cases containerization requirements. Section 3 – Kubernetes architecture	Sections 2, 3, 4, and 5 provide the requirements of the UCs to be containerized, the K8s architecture adopted by each one, the deployment instructions in each case, and

architecture. This will be achieved by encapsulating VNFs and NetApps in lightweight edge containers to achieve faster instantiation times, reduced latency, low resource utilization and better response times, especially considering the time-criticality of PPDR communications and applications. “	Section 4 – Instructions for the deployment	the lessons learned. Their content is valuable for supporting third-party PPDR verticals in the migration towards cloud-native deployments. This migration will enable the PPDR community to take advantage of the benefits provided by the usage cloud-native containerized solutions.
	Section 5 – Virtualized core networks	

1.2 Kubernetes concepts

K8s is a widely adopted solution for the orchestration of containers. K8s defines a **cluster** that consists of a set of worker machines called **nodes**, and a **control plane**. The nodes hosts **Pods**, which run containerized applications. At least one node must be defined. The control plane manages the nodes and Pods. Each Pod is a group of one or more **containers**, and are the smallest deployable computing unit that can be created and managed in K8s. Usually, each Pod corresponds to one containerized **micro-service**.

Each Pod get its own Internet Protocol (IP) address, and they are not permanent resources; Pods can be instantiated dynamically, according to different policies; replicate micro-services; re-instantiate micro-services that are down; and so forth. Since some Pods provide functionalities to other Pods, this yields the problem of keeping track of which IP address needs to be reached at any moment. K8s thus defines **services**.

A K8s service is an abstraction layer that exposes the application running on a set of Pods, defining a policy to access them. Different types of K8s services are defined:

- ClusterIP: A load balancer only reachable from within the cluster. Exposes the service on a cluster-internal IP.
- LoadBalancer: A load balancer that exposes the service externally, using a cloud provider's load balancer.
- NodePort: Exposes the service on each Node's IP at a static port. A NodePort service can be reached from outside the cluster, by requesting NodeIP:NodePort.

A different available option is by declaring a Headless service. In this case a Load Balancer is not declared, and it is only accessible within the cluster. A Domain Name System (DNS) request to this service will return all Pods’ IPs from that service.

Although K8s has its own definition for the concept "services", for the sake of consistency, this deliverable will continue to use the term to refer to the different functionalities that can be deployed, such as *vertical services*. To refer to K8s "services", they will be referred to directly (*e.g.*, loadBalancer, NodePort, *etc.*).

1.3 Kubernetes in 5G-EPICENTRE

The general architecture for adopting the use of K8s in the 5G-EPICENTRE project was described in D1.3. Within 5G-EPICENTRE, the goal is to smoothly evolve the existing Network Functions Virtualization Management and Orchestration (NFV-MANO) architectures in each testbed into a K8s-based orchestration system for automating VI and the deployment of Cloud-native Network Functions (CNF). The solution adopted by the project to fulfil this target is based on the usage of the KuberVirt VM management add-on. Kubervirt allows the usage of containers and VMs on the same cluster, or even the same node, using the same networks and same storage infrastructure. In addition to KubeVirt, an instance of libvirtd is used to manage the lifecycle of the VM process. Within this approach, it is possible to have pure K8s nodes to place containers into Pods, or K8s VMs Nodes, and to deploy K8s pods and K8s VMs pods, as shown in Figure 1.

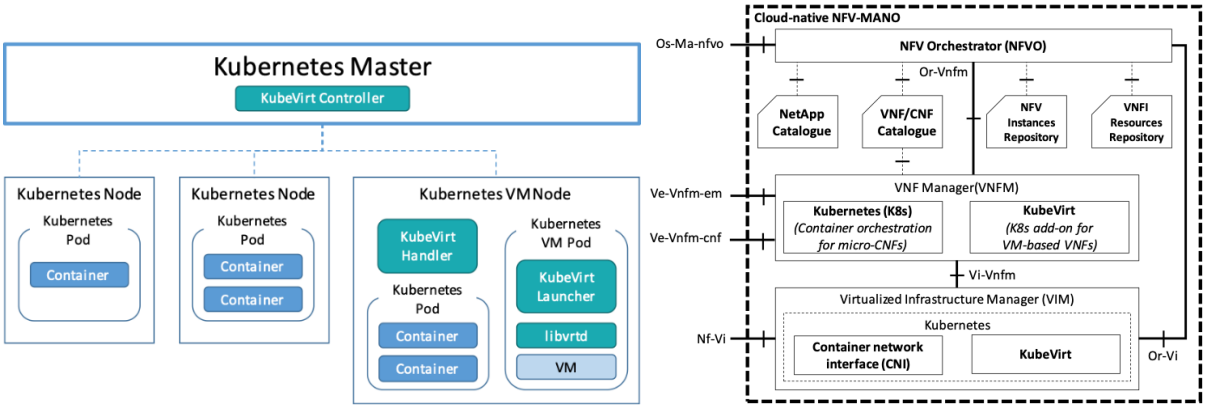

Figure 1. Deployment of Kubernetes in the context of the 5G-EPICENTRE project.

2 Use Cases containerization requirements.

This Section identifies the containerization requirements of the UCs covered in the project, in order to identify the hardware resources that should be provided by the platforms and also, the software add-ons needed for the deployment and management of the containerized services.

2.1 UC1 - Multimedia Mission Critical Communication and Collaboration Platform

The Mission Critical (MCX) services system from Airbus (ADS) can be deployed on any container runtime engine, leveraging modern orchestration frameworks like K8s. The application is lightweight. Below are the recommendations relative to the system:

- 1vCPU and 1GB RAM.
- Bandwidth Receive (Rx): Around 500kpbs per video stream and per user, for Standard Definition (SD) stream at 15 frames-per-second (fps) / Around 5Mpbs per video stream and per user for Full High Definition (FHD) stream at 30fps.
- Bandwidth Transmit (Tx): Around 500kpbs per video stream for SD stream 15fps / Around 5Mpbs per video stream for FHD stream 30fps.

The application does not include a cartography server, and needs to be connected to the Internet to load maps background (nonetheless, a cache can be exploited).

ADS media micro-services can work with mixed protocols (Transmission Control Protocol [TCP] and User Datagram Protocol [UDP]), potentially increasing performances and Quality of Experience (QoE, not mandatory). In order to exploit this on K8s infrastructure, the feature gate MixedProtocolLBService must be enabled.

2.2 UC2: Multi-agency and multi-deployment mission critical communications and dynamic service scaling

Currently the MCX services deployment from Nemergent (NEM) has the following dependencies:

- **Network:** Container Network Interface (CNI, any should work).
- **Storage:** Container Storage Interface (CSI, any should work).
- **Seamless monitoring:** K8s Prometheus operator is required.
- **External access to services**, which can be configured to use LoadBalancer or NodePort (see Section 1.3). The system is by default configured to use LoadBalancer:
    - if LoadBalancer is to be used, the cluster should have an implementation or LoadBalancer (it has been tested using metalLB[1]).
    - If NodePort is to be used, it does not have any requirement.
- Backend and Enabler requires to be able to deploy it in hostnetwork mode.
- **Deploying tool:** Helm is required. It is not necessary to use Helm on the host machine. Helm can be used in a remote machine, pointing out to the host machine.

2.3 UC3: Ultra-reliable drone navigation and remote control

The realised applications can be deployed in a K8s environment that includes:

- K8s Master and 2 Worker Nodes (K8s cluster).
- Virtual Private Network (VPN) access: get external access to deploy the Docker image at the K8s cluster.
- Docker Community Edition: for container images.

[1] MetalLB, bare metal load-balancer for Kubernetes.

- Container Network Interface (Flannel[2]): communication between the containers.
- LoadBalancer (metalLB): for communication outside the cluster, with User and Operation control centre.

The Docker image requires 2.5 GB of memory. The Graphical User Interface (GUI) of the QGroundControl container application was tested under Ubuntu 20.04 with X11 Server (X11 forwarding).

For the applications, the following resources are required:

- 1 VM with 2 vCPU, 16GB RAM and 30GB hard disk space.
- Required bandwidth Uplink (UL, drone to live video server): Approximately 20Mpbs per video stream.

2.4 UC4: IoT for improving first responders' situational awareness and safety

The Mobitrust platform is the key technology behind UC4, which is centred on Internet of Things (IoT) for improving first responders' situational awareness and safety. Thought to be used under challenging conditions and with few available resources, the deployment of this Network Application under a cloud-native environment demands a short list of requirements, which can be summarized as:

- **Remote access:** ensure some form to interact with the K8s cluster from any location (*i.e.*, VPN access). The goal at this stage is to use *kubectl* (Kubernetes command-line tool) command, together with YAML[3] files describing the deployment of each micro-service (*i.e.*, *kubectl apply -n <namespace> -f <yaml_file>*).
- **LoadBalancer:** ensure that a LoadBalancer is installed on the K8s cluster to handle public (*i.e.*, outside the cluster) exposure of some services, and distribute the load among replicas of each microservice.
- **Ingress Controller:** ensure that an Ingress Controller is installed in the K8s cluster, to handle HTTP/HTTPS requests via a single proxy (*e.g.*, NGINX[4] Inc. Ingress Controller).
- **Persistent storage:** even though this is not mandatory, it maintains important data that may need to be available after events such as downtimes, reboots, *etc*. As such, persistent storage should be available through Persistent Volume Claims (PVCs).
- **CoreDNS change:** it is required that the cluster admin performs a minor change in CoreDNS[5], aiming to map the services in UC4 namespace to its own domain (mobitrust.org, which already has valid Secure Sockets Layer [SSL] certificates).

2.5 UC5: BlueEye Remote Video

BlueEye Remote video requires:

- Network: Docker image support.
- Storage: single volume required.
- 1 to 6 pod CPU.
- 2GB to 8 GB pod RAM.

2.6 UC6: Fast situational awareness and near real-time disaster mapping

OPTO deployment requires:

- Network: Any CNI should work.
- Storage: Any CSI should work.

[2] Flannel (https://github.com/flannel-io/flannel) is a simple, easy way to configure a layer 3 network fabric designed for K8s.
[3] YAML (https://yaml.org/) is a human-friendly data serialization language for all programming languages.
[4] NGINX (https://www.nginx.com/): Advanced Load Balancer, Web Server, & Reverse Proxy.
[5] CoreDNS (https://coredns.io/plugins/kubernetes/) is a plugin that implements the K8s DNS-Based Service Discovery Specification.

- Exposed IP-Ports to receive messages from outside.
- Requires to be able to publish on specified ip-ports (TCP/UDP).
- Delivering docker image tarball (tarball is a tar file).

2.7 UC7: AR and AI wearable electronics for PPDR

Biquo K8s deployment requires:

- Ability to expose K8s Services of LoadBalancer type, enabling both UDP and TCP packets forwarding.
- Persistent storage driver compatible with K8s PersistentVolumeClaim.
- A dedicated K8s namespace.

2.8 UC8: AR-assisted emergency surgical care

UC8, driven by ORAMA, is deployed as a VM rather than a container. Although Docker containers outperform VMs in the case of space and processing overhead, they are rather immature in graphics acceleration processes. In the context of UC8, which involves the deployment of an AR-application, the use of VMs is far more advantageous, since they have highly optimized graphics drivers and Kernel-based VM (KVM) pass-through support. Furthermore, Docker containers have limited graphics drivers support since only experimental versions (for all vendors) for Linux are currently available. Additionally, Unity3D support for Linux is still at an experimental stage as well, making the task of porting the Unity3D-based ORamaVR MAGES Software Development Kit (SDK) to Linux a difficult and error prone procedure. For all the aforementioned reasons, UC8 is deployed as a VM via KubeVirt. The general methodology elaborated in the project to support the deployment of VM is described in Section 3.4.2 of Deliverable D1.3.

3 Kubernetes architecture

This Section provides and overview of the architecture of the different UCs. A clear view of the architecture will enable a better understanding of the instructions provided in Section 4 for the deployment of the use cases.

3.1 UC1: Multimedia Mission Critical Communication and Collaboration Platform

Below is the description of the various services of the UC1 critical communication platform:

- **Identity Management Function (IdMF):** is responsible for authenticating a Mission Critical Push-To-Talk (MCPTT) client in the system. For testing purposes, the server can accept any new registration.
- **Situation Management Function (SMF):** is responsible for managing the situation information: status, locations, *etc.*
- **Audio Signalisation Function (ASF):** handles floor control for audio communications.
- **Audio Media Function (AMF):** handles real time packets for audio communications.
- **Video Signalisation Function (VSF):** handles control for video communications.
- **Video Media Function (VMF):** handles real time packets for video communications.
- **Data Management Function (DMF):** handles non real time data streams.
- **KPI Function (KPIF):** Records KPI related to communications.
- **Webfront Server (WFS):** Web User Interface (UI) for client.

3.2 UC2: Multi-agency and multi-deployment mission critical communications and dynamic service scaling

Figure 2 illustrates the different micro-services integrated into the MCX application developed by NEM, and their exposed services: clusterIP, LoadBalancer/NodePort, Headless service. The description of the different micro-services integrated is provided below:

- **MCX Application Server (AS):** is responsible for providing control and management of communications (MCPTT-voice-, MCVideo-video- and MCData-data-). The MCX application server could be divided in two main roles in the system:
    - The **MCX Controlling Application Server (CAS):** handles the floor control for both private and group calls, and it forwards media flow as well. Besides the MCX Participating AS, the MCX CAS could also communicate with other Controlling, or non-Controlling Controlling AS(s).
    - The **MCX Participating Application Server (PAS):** handles the communication with the MCX clients, and plays the role of a relay point for floor control between the MCX clients and the MCX CAS. It also manages access and priority control of users to the communication in place and access control to MCX clients triggering communications (checking the capabilities of each). Finally, the MCX PAS connects with the anchor points in 5G, such as the Policy Control Function (PCF) through the N5 interface.
- **Identity Management Server (IdMS):** is responsible for authenticating an MCPTT client in the system. The server is provisioned with the client ID, MCPTT ID and password.
- **Configuration Management Server (CMS):** is responsible for managing the MCPTT/ MCX configurations, such as user profile, User Equipment (UE) configuration, functional aliases, and service configurations.
- **Group Management Server (GMS):** is responsible for managing the groups' information. The groups could be either be formed by clients/users, or other groups (group regroup).
- **Key Management Server (KMS):** is responsible for the distribution and storage of security keys, and information like encryption keys, for the communication of MCPTT calls (private and group), SDS (Software-Defined Storage) data protection, management server safe, and integrity-based communication (both signalling and media).

- **Session Initiation Protocol (SIP) / IP Multimedia Subsystem (IMS) Core:** provides the SIP core required as a SIP register and message forwarding framework. The SIP core is comprised of different elements: Proxy Call Session Control Function (P-CSCF), Interrogating Call Session Control Function (I-CSCF), Serving Call Session Control Function (S-CSCF) and Home Subscriber Server (HSS).
- **Backend & Enabler:** manages sessions between server and end users, and implements different interfaces and translations for non-MCPTT systems with MCPTT services.
- **Non-relational database:** used to store dynamic information, like registered users.
- **Database (DB):** SQL database used to store micro-service configuration information.
- **Enabler-WS:** Backend used by Dispatcher.
- **HTTP-Proxy:** middleware of external HTTP traffic, that inspects and redirects HTTP traffic to the corresponding micro-service.
- **MCPTT-Exporter:** module that exports monitoring information according the monitoring process instantiated (RabbitMQ or Prometheus).

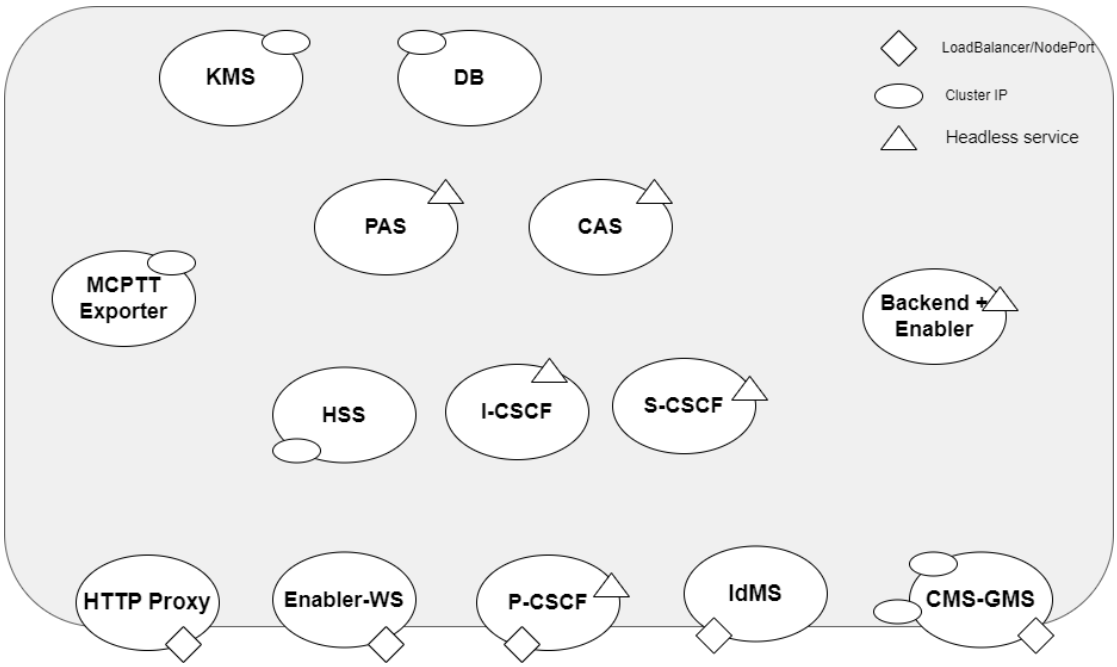

Figure 2: Kubernetes MCX architecture

3.3 UC3: Ultra-reliable drone navigation and remote control

The services provided for UC3 are:

- **Video splitter**, to route the drone video signal to the QGroundcontrol Clients and Operation Control Center.
- **QGroundcontrol App** (Docker Image), for video playback and drone control.
- **Message Broker** (RabbitMQ), for forwarding of measurement data.
- **Mavlink splitter**, for distributing drone information, *e.g.*, position to different clients.
- **Access management** for drone control, which is available exclusively to one person (feature is planned).

3.4 UC4: IoT for improving first responders' situational awareness and safety

Figure 3 presents the Mobitrust application's internal components, following a Network Application model approach.

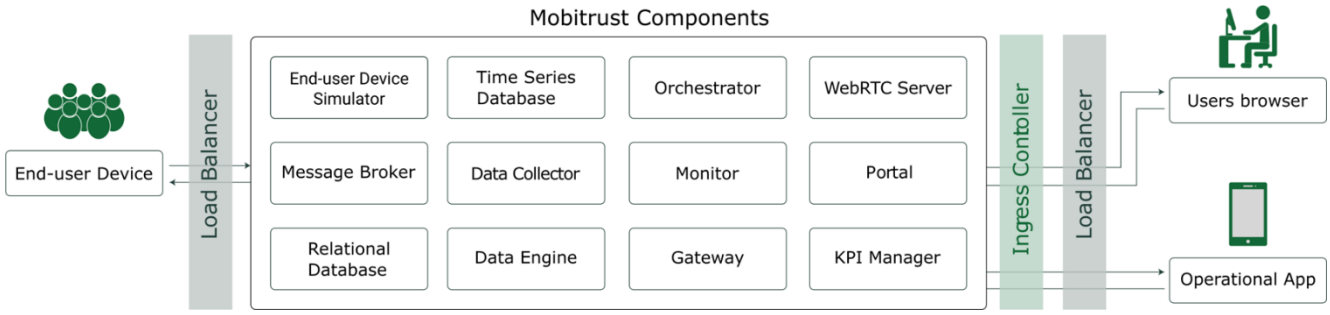

Figure 3: UC4 Components

The presented architecture is composed of twelve (12) internal components (micro-services). Additionally, the three main actors interacting with the deployment of this application are also illustrated. The following list presents a summative description for each one:

- **End-user Devices:** These correspond to the Mobitrust BodyKits (BKs), used by the first-responders in the field. They support 5G communications, sensors data collection, multimedia capture and data pre-processing.
- **End-user Device Simulator:** It is used for integration tests and debug purposes. It simulates the data streams usually established between a real end-user device (BK) and several Mobitrust components.
- **Time Series Database:** It corresponds to a database used to store the information collected from the multiple sensors in the Mobitrust BKs.
- **Orchestrator:** It is responsible for the management of control data. It deals with authentication and authorization of users, and is also responsible for the setup of end-user device components, including drivers and the establishment of data channels, both for sensors and multimedia devices (cameras and microphones).
- **WebRTC Server:** It is the component in charge of handling real-time audio and video transmissions from the field towards the Command and Control Centre (CCC).
- **Message Broker:** It corresponds to the communication backhaul of the system. It is responsible for all the communications among the different application components, following a publish/subscribe model.
- **Data Collector:** It is the component in charge for collecting and reporting metrics/sensor data. It obtains all the data from the message broker. This data is mainly generated by the sensors in the end-user devices.
- **Monitor:** It is responsible for monitoring and reporting on the state of the end-user devices.
- **Portal:** It provides a way to obtain situational awareness by visualising all the data collected by the platform.
- **Relational Database:** It corresponds to the component that stores the users' information, end-user devices details, WebRTC mount points and their associations, as well as the access control policies.
- **Data Engine:** It is a native data processing engine that can process streams and batch data from the Data Collector. Custom logic or user-defined functions can be plugged-in to process alerts with dynamic thresholds, and perform specific actions based on these alerts.
- **Gateway:** It is responsible for providing the services consumed by the CCC. It contains all backend operations that enable visualisation of data collected by the platform, as well as the processing of requests by human operators.
- **KPI-Manager:** It is responsible for managing the collection, processing and distribution of KPIs originated by the platform.
- **Users' browser:** It corresponds to the visualization of the CCC front-end application. This tool displays the geo-localisation of every field operator, plus specific per-operator data and streams, such as real-time video, real-time communications and sensors data (temperature, heartbeat, etc.).

- **Operational App:** It corresponds to the mobile version of the CCC, and it is intended to be used by first-responders in the field. It is typically used by team leaders/commanders.

3.5 UC5: BlueEye Remote Video

RedZinc (RZ) containerized services use video stream with auto-scaling video routers in 5G Edge.

The integrated solution allows critical communications on the move, to send 'You See What I See' video to remote experts for support and oversight. In Urban Search and Rescue, *BlueEye Handsfree* enables the in-field searcher to send real-time point-of-view video to a remote commander. The remote commander can see the situation from the searcher's perspective and provide them appropriate support and expertise. The main components of this solution are the following (see Figure 4):

- **Video router:** it routes video streams between participants in a video call. In a K8s structure, the video router micro-service can be deployed and scaled as a K8s deployment, which allows it to be easily managed and scaled as needed.
- **TURN Server:** it helps relay video streams between participants who are unable to establish a direct peer-to-peer connection. In a k8s structure, it can be managed and scaled as a Kubernetes deployment.
- **Medical Hotdesk:** it is a shared workspace that is used by medical professionals to access patient information and collaborate on medical cases. It is integrated with the video call services to provide a seamless experience for medical professionals.
- **Image Repository:** it is a centralized location for storing and managing container images.
- **Orchestrator:** it is used to manage the deployment and scaling of the video router, TURN server, medical hotdesk, and other micro-services that make up the application. It is also configured to integrate with the TURN server to ensure that video streams are relayed efficiently and reliably.

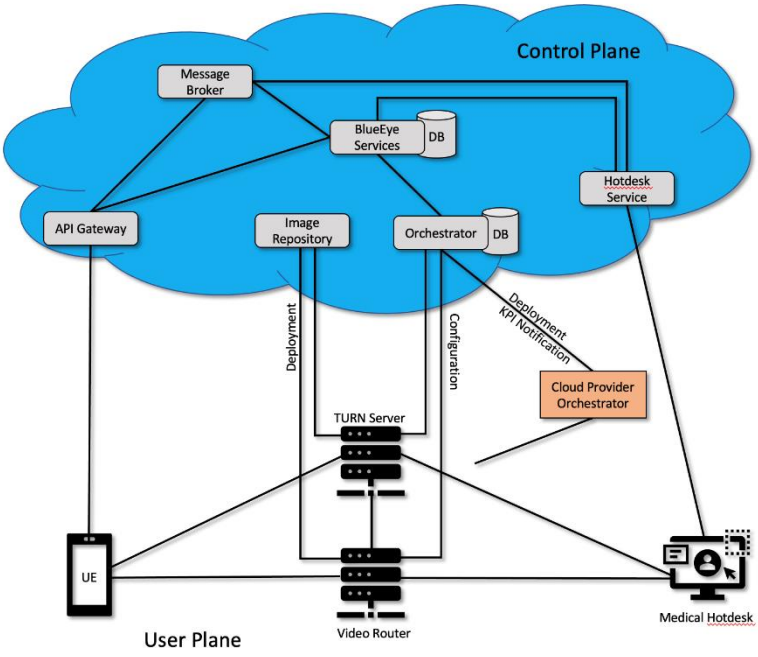

Figure 4: BlueEye Remote Video use case

3.6 UC6: Fast situational awareness and near real-time disaster mapping

OPTO containerized services use image streams to receive, relay and analyse images, and deliver the result objects as metadata.

- **Proxy:** receives image data and publishes it to the Detector (see below). Also publishes the image data on a specific network port.
- **Detector:** Receives the image data from the proxy and analyses it. Publishes metadata (*e.g.*, position and prediction) on a specific network port.
- **MQTT-Publisher:** Publishes measurement timestamps from proxy and detector to central MQTT instance in the host network.

3.7 UC7: AR and AI wearable electronics for PPDR

Biquo micro-services mainly support the QubeCenter application, which provides a robust infrastructure for seamless communication between various data producers and devices on the platform. Some of the services included in this configuration are responsible for generating real-time, high-priority data, that is consumed immediately. Meanwhile, other services are designed to store and archive data, maintaining the necessary interconnections for long-term use.

Biquo architectural components are show in Figure 5, and are described below:

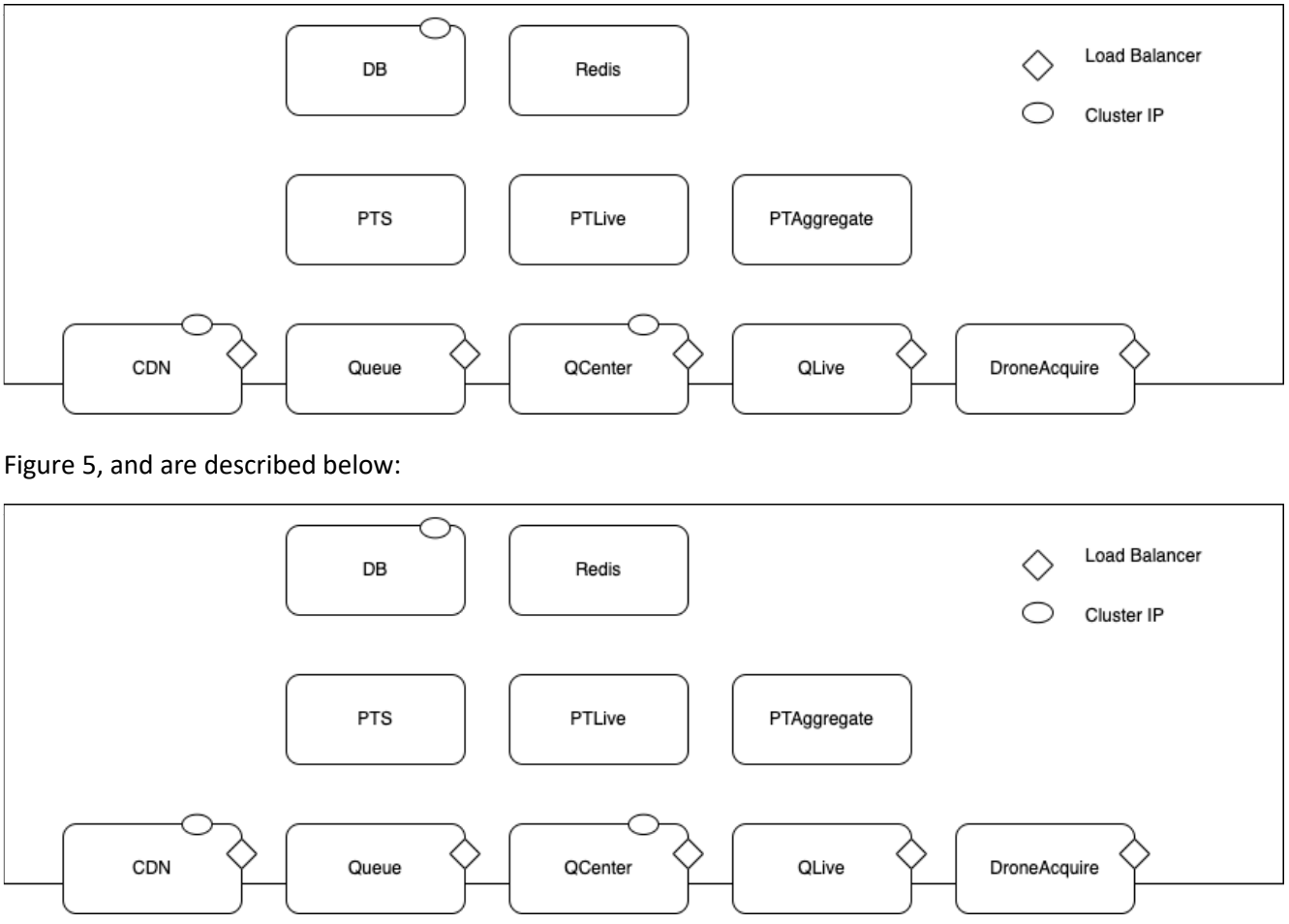

Figure 5: Biquo architecture

- **CDN (Content Delivery Network):** responsible for delivering data files, configured to reduce latency and reduce resources footprint.
- **PTS (Process to Save):** consumes messages from the Queue and saves them to the DB component; does not need a K8s Service, as it is not accessed neither from the Cluster, nor externally.

- **PTLive (Process for Live Data):** consumes messages from the Queue, in order to produce real-time data accessible from other components; does not need a K8s Service.
- **PTAggregate (Process for Data Aggregation):** consumes messages from the Queue and generates proximity maps of the actors (users, devices) in the system; does not need a K8s Service.
- **DB:** persistent relational Database Management System (DBMS); persistent data is here added by PTS components and retrieved by QCenter instance.
- **Redis:** in-memory data store, used for session management by other components; does not need a K8s Service.
- **Queue (RabbitMQ with AMQP protocol):** Message broker, responsible of all the communication among the involved components, via Advanced Message Queuing Protocol (AMQP) protocol.
- **QCenter (Queue Center):** web-based platform for managing and configuring services; it provides a central interface for managing users, devices and settings.
- **Qlive:** web-based application exposing real-time geo-located data consumed from the Queue.
- **DroneAcquire (Acquisition of Drone Video Streams):** acquires streaming video packets from a drone and serves live feeds, available for consumption by other actors (tested only in FHHI Testbed).

3.8 UC8: AR-assisted emergency surgical care

VMs are used instead of containers, a decision based on the information mentioned in Section 2.8. Although the 5G-EPICENTRE project revolves around supporting containerized solutions, it also maintains backwards compatibility with VM-based applications, when their containerisation is not feasible, or cannot yet produce satisfying results.

In ORAMA's UC8, a Windows VM is deployed, that stores:

- The main MAGES-based AR application.
- The TURN server, which circumvents WebRTC's "smart" behaviour, by forcing specific communication paths.

These executables, with a combination of Windows Batch files, comprises the first main part of the ORAMA application (AR_Application_LocalSystem).

For a detailed reference on how KubeVirt will be used to deploy UC8 VM, refer to Section 3.4.2 of Deliverable D1.3.

4 Instructions for the deployment

This Section provides the guidelines for the deployment of the eight UCs considered in the project.

4.1 UC1: Multimedia Mission Critical Communication and Collaboration Platform

Airbus directly provides K8s native deployment. The provided package ensures the automatic deployment of the application and its services, just applying the following command line:

```
$kubectl deploy -f .
```

4.2 UC2: Multi-agency and multi-deployment mission critical communication and dynamic service scaling

Nemergent MCX service can be deployed using Helm, a package manager for K8s. It helps to deploy applications reading the templates, deploying the services accordingly. If the cluster environment does not support Helm, it can be used from an external machine pointing out to the cluster. It is assumed that a cluster and a node is already deployed, if not reference to Section 4.2.1:

```
helm install --create-namespace --namespace nemergent mcptt chart/
--set imagePullSecrets=<your reguistry json in base64> --set
imageRegistry=<registry repository> --set loadBalancer.enabled=true --
set exporter.prometheusIntegration.enabled=true
```

imagePullSecrets: registry JSON in base 64, for example, after Docker login in the private registry, this JSON can be found in ./docker/config.json. It is just required when accessing private registry.

imageRegistry: A repository where containerized images are loaded and could be downloaded from. In NEM's case, "registry.nemergentsolutions. com/production".

Configuration parameters, can be re-configured using the same Helm command, replacing *install* with *upgrade*. If the *realm* is not set, the *realm* will be set with the *namespace* name. NodePort, or LoadBalancer can be used to expose services, but not at the same time; by default, LoadBalancer is enabled. If both are enabled, the services will use LoadBalancer.

4.2.1 Deployment cluster with microK8s

For local deployment microK8s[6] are used, an open-source system for automating deployment, scaling, and management of containerised applications. It provides the functionality of core K8s components, in a small footprint, scalable from a single node to a high-availability production cluster.

Using CentOS[7] snapd needs to be installed first:

```
> sudo rpm -ivh https://dl.fedoraproject.org/pub/epel/epel-release-latest-
```

[6] MicroK8s - Zero-ops Kubernetes for developers, edge and IoT.
[7] The CentOS Project

```
7.noarch.rpm
> sudo yum update
> sudo yum install snapd
> sudo systemctl enable --now snapd.socket
```

To enable classic snap support, enter the following command:

```
> sudo ln -s /var/lib/snapd/snap /snap
```

Install microk8s:

```
> sudo snap install microk8s --classic
> sudo usermod -a -G microk8s $USER
> sudo chown -f -R $USER ~/.kube
```

The system will also need to be restarted for the group update to take place.

To be as lightweight as possible, microk8s only installs the basics of a usable K8s installation. Therefore, the following add-ons should be enabled:

```
> microk8s enable dns:XXX.XXX.XXX.XXX ingress storage rbac
metallb:LOWER_IP-UPPER_IP
```

A brief explanation of each of the enabled feature:

- **dns:** deploys CoreDNS.
- **ingress:** a simple ingress controller for external access.
- **metalLB:** deploys the MetalLB Loadbalancer.
- **Storage:** creates a default storage class, which allocates storage from a host directory. By default, storage add-on uses local filesystem storage to the node where it was added. A more complex storage system might be needed in a complex deployment.
- **rbac:** Enable Role Based Access Control for authorization.

4.2.2 Helm installation

Download the desired version of Helm from: https://github.com/helm/helm/releases.

The installation process is as follows:

```
> wget https://get.helm.sh/helm-v3.8.0-linux-amd64.tar.gz
tar -zxvf helm-v3.8.0-linux-amd64.tar.gz
> mv linux-amd64/helm /usr/local/bin/helm
```

4.3 UC3: Ultra-reliable drone navigation and remote control

The installation is done via YAML files under K8s, or as Container Image under Docker CE. HHI provides a native K8s deployment. The YAML files can be implemented simply by executing:

```
> kubectl create -f <yaml.file>
```

They are uploaded in the gitlab.5gepicenter.eu repository.

The IP address and port of the Docker container containing the QGroundcontrol application must be adjusted in the YAML file for the video splitter deployment. To install the Docker container, the Docker image must be pulled from gitlab.5gepicenter.eu repository. Then, it needs to be run with the shell script 'start-qgc-container.sh', which is also uploaded in the gitlab.5gepicenter.eu repository.

4.4 UC4: IoT for improving first responders' situational awareness and safety

This sub-section aims to present the minimal list of phases needed to deploy UC4 under a cloud native environment (*i.e.*, K8s cluster or similar).

4.4.1 Pull images from the container registry

The first step consists of establishing the connection to the container registry to be possible to pull the container images. That may be achieved using the following command:

```
kubectl -n <namespace> create secret docker-registry <secret_name> --docker-server=<registry> --docker-username=<username> --docker-password=<access_token> --docker-email=<email>
```

For example:

```
kubectl -n UC4 create secret docker-registry mt-5g-epicentre-secret --docker-server=registry.5gepicentre.eu --docker-username=deployments-user –docker-password=… –docker-email=user@5gepicentre.eu
```

4.4.2 Generate a *tls* secret for the Ingress Controller

All the communications between the cluster and external networks should be encrypted. Hence, Transport Layer Security (TLS) secrets are needed (encoded certificate and respective private key). Loading an existing certificate and key to the cluster namespace can be achieved with the following command:

```
kubectl create secret tls <secret_name> --key <key.pem> --cert <crt.pem> -n <namespace>
```

For example:

```
kubectl create secret tls star.UC4.org-secret --key star.UC4.org-key.pem --cert star.UC4.org-crt.pem -n UC4
```

4.4.3 Map an external domain to CoreDNS

The cluster hosting the deployment of this application needs to map the services in UC4 namespace to the Network Application's own domain (mobitrust.org). As such, a CoreDNS change needs to be performed, which may be achieved by:

1. Discovering the cluster Fully Qualified Domain Name (FQDN) and UC4 namespace, as well as the domain to be used (mobitrust.org).
2. Opening the ConfigMap for CoreDNS file:

```
kubectl edit configmap coredns -n kube-system
```

3. Add the following rewrite rule (in bold):

```
apiVersion: v1
data:
  Corefile: |
    .:53 {
        errors
        health {
           lameduck 5s
        }
        ready
        rewrite stop {
           name regex (.*)\.mobitrust\.org\.$ {1}.5g-epicentre.svc.cluster.local
           answer name (.*)\.5g-epicentre\.svc\.cluster\.local\.$ {1}.mobitrust.org
        }
...
```

In the previous file, “cluster.local” was the cluster domain, “mobitrust.org” was the Network Application domain and “5g-epicentre” was the namespace. It must be highlighted, that this step needs to be performed by the cluster admin.

4.4.4 Grant permissions

This consists on creating a role binding, which is needed for the WebRTC deployment (the application “media server”, as described in Section 3.4 to obtain the LoadBalancer IP of its service. The creation of role bindings in a K8s cluster may be accomplished by:

```
kubectl create rolebinding <name> --clusterrole=view --serviceaccount=<namespace>:default
--namespace=<namespace>
```

For example:

```
kubectl create rolebinding default-viewer --clusterrole=view --serviceaccount=5g-epicen-
tre:default --namespace=5g-epicentre
```

4.4.5 Deployment

After the completion of the previous steps, the internal components of UC4 may be deployed. Currently, the deployment is usually achieved with the use of a custom bash script. It contains a list of *kubectl* commands that instantiate the different micro-services of the application in the correct order.

To deploy a micro-service, the following command is used together with a YAML (deployment description):

```
kubectl apply -f <filename> -n <namespace>
```

While for deleting its deployment:

```
kubectl remove -f <filename> -n <namespace>
```

Considering that some micro-services have dependencies among them (*i.e.*, database access), startup probes have been implemented to ensure the dependencies are fully running and answering, before those micro-services can be deployed. An example of a **startupProbe** in the a YAML file can be:

```
startupProbe:
            tcpSocket:
              host:
              port: <port>
            failureThreshold: <maximum_of_attempts>
            periodSeconds: <elapsed_time_per_attempt>
```

Within the custom bash script, another *kubectl* command is used to wait on the availability of dependencies before proceeding, which may be accomplished through:

```
kubectl wait --for=condition=available --timeout=600s deployment/<service_name> -n
<namespace>
```

4.5 UC5: BlueEye Remote Video

UC5 containers are not meant to be manually deployed, this means that the containerized micro-services for the video call service are not meant to be deployed manually by operators or developers. Instead, the deployment process is automated and managed by an orchestrator service, as it is common practice in a K8s structure, where containerized applications are deployed and managed using automated tools and processes.

The UC5 Orchestrator Service, with access to the cloud's *kubeconfig* file, generates the pod and service definitions, and loads them directly to K8s: The deployment process is automated using an orchestrator service. The orchestrator service has access to the *kubeconfig* file, which is a configuration file that provides the necessary credentials and connection details to interact with the K8s cluster. Using this access, the orchestrator service generates the pod and service definitions for the containerized micro-services and loads them directly into K8s. This allows the micro-services to be deployed and managed automatically, without manual intervention.

The UC5 Orchestrator receives "Hello" messages from deployed services as soon as they are up and provides them with any required extra configurations: The orchestrator service monitors the status of the deployed microservices. The microservices send "Hello" messages to the orchestrator service as soon as they are up and running. The orchestrator service then provides any extra configurations, or updates that are required for the microservices to function properly. It helps monitor the health of the deployed microservices, and provides any necessary support (or updates) to ensure that the application is running smoothly.

4.6 UC6: Fast situational awareness and near real-time disaster mapping

OPTO is delivering Docker image tarballs, which can be imported into local, or remote Docker registries. To start a container from these images, OPTO delivers a compose YAML file, which starts the services with all needed parameters.

4.7 UC7: AR and AI wearable electronics for PPDR

Biquo deployment flow relies on **kubectl** and **git;** access to 5G-EPICENTRE git repository and Docker container registry has to be granted, providing appropriate tokens.

Once the cluster is setup, a Biquo instance can be initiated with the following steps:

1. Clone the dedicated kuberenets YAML files git repository:

```
git clone https://gitlab.5gepicentre.eu/youbiquo/biquo
```

2. Apply the configuration:

```
kubectl apply -f <directory>
```

4.8 UC8: AR-assisted emergency surgical care

4.8.1 Requirements

The basic requirements for deploying the UC8 VM are the following:

- LibVirt.
- Linux (or WSL2 on Windows 11 only).
- KVM (already present on WSL2, can be checked with the cpu-checker ubuntu package).
  For WSL2, libvirtd and virtlogd should be started as root:

```
§ sudo su
§ chmod 666 /dev/kvm && libvirtd & virtlogd &
```

4.8.2 Virtual machine configuration

The base image of UC8 requires a disk image of about 50GB for Windows 10, the AR-application and additional drivers. For Windows, the latest ISO is used to setup the VM. For the configuration, the following are needed during the VM setup from virt-manager:

- When prompted for OS ISO, choose the Windows ISO and type in “Windows 10” in the field below (click ‘show unsupported operating systems’ if the option does not initially appear).
- Storage volume format should be ‘raw’, and size should be at least 50GB.
- RAM is dependent on the computer/node, half amount of the available RAM is a good choice (*e.g.*, if 16GB of RAM are available, 8GB of RAM for the VM should be sufficient).
- CPUs should be half the amount of the available cores as well.
- In CPU options, make sure that ‘Copy host configuration’ is enabled (important for perf on WSL2).
- Before finalizing the process, choose 'customize configuration before install':
  - Select the SATA Disk created (the 50GB one), and for bus type, choose ‘virtio’.
  - For the NIC (create it if it doesn’t exist), choose ‘virtio’ for the device model.
  - Add a CDROM virtual hardware disk device, and select the virtio drivers ISO.
  - In the Boot Options, make sure that the installer iso is first, and the 50GB disk is second.

4.8.3 Virtual machine deployment

The UC8 edge server component will be deployed via libvirt on a LXD[8] container, as a VM running Windows 10. The edge node is equipped with an NVIDIA GeForce RTX 2080 Ti GPU card, suitable for the needs of a high-fidelity AR application. The CPU of the VM is an Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz, with 16GB of RAM. The VM deployed needs exclusive use of the GPU with GPU pass-through, since vGPU licensing is not available in the testbed. The scenario will take advantage of the capabilities and number of servers that CTTC facilities can provide. To provide GPU access, the KubeVirt kubernetes module is used, which can provide hardware devices to

[8]LXD (https://linuxcontainers.org/lxd/introduction/): Linux Container

libvirt. To support GPU pass-through, VFIO is used to disable GPU utilization from within the host operating system, and hand it off to libvirt. It is important to note that this is done in the host system; meaning that no other application can make use of the GPU during the VM's runtime.

5 Virtualized core networks

5.1 Athonet 5GC

Athonet (ATH) provides a fully Standalone (SA) software-based 5G mobile core (5GC) network for voice and data services, running in public and private clouds, or in a commercial off-the-shelf hardware ('in a box'). The software is completely virtualized, and implements all the needed core network components exposing 3GPP-standard interfaces. This solution allows the 5GC to connect to any standardized 5G radio/user equipment and Data Networks (DNs), in order to provide a complete end-to-end service.

Although the provided ATH 5GC is not designed to be deployed on a K8s cluster, it runs as a VM over a commercial-off-the-shelf hardware, managed by a VMware hypervisor instance. Every Network Function (NF) is directly packaged as an independent OCI (Open Container Initiative) container, running on a container runtime in the VM. This design makes the 5GC modularized, and its micro-service architecture enables deployments that can split control-plane and user-plane, as well as allowing interoperability with different vendors' core components.

In particular, referring to Figure 6, the currently provided NFs and their exposed standardized interfaces are:

- Access and Mobility Management Function (AMF), which exposes the N1 and N2 interfaces.
- Session Management Function (SMF), which exposes the N4 interface.
- User Plane Function (UPF), which exposes the N3, N4 and N6 interfaces.
- Unified Data Management (UDM).
- Authentication Server Function (AUSF).
- Unified Data Repository (UDR).
- Policy Control Function (PCF), which exposes the N5 interface.
- Network Repository Function (NRF).
- Network Slice Selection Function (NSSF).

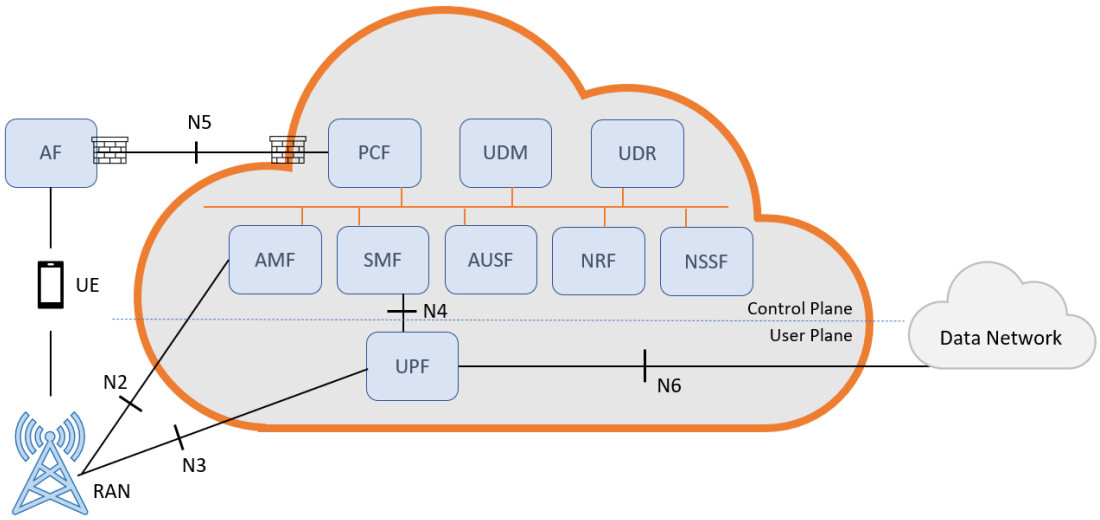

Figure 6: ATH 5G core network and its interfacing.

The 5GC is equipped with an integrated monitoring system (Prometheus[9] and Grafana[10]), adopted to collect and visualize 3GPP standard-defined 5GC KPIs. This information can be *shipped* to any other tool able to catch Prometheus output data.

[9] Prometheus (https://prometheus.io/): Monitoring System & Time Series database.
[10] Grafana (https://grafana.com/); The Open Observability Platform.

Virtualization of the 5GC facilitates its deployment in heterogeneous infrastructures, such as hyperscalers, private or public cloud frameworks, private premises, *etc*. Furthermore, the ATH 5GC solution offers open and standard 3GPP interfaces, allowing interaction with any other standard-compliant external component (*e.g.*, Radio Access Network [RAN], orchestrators, UEs, analytic/monitoring systems, Application Functions).

5.2 Open5GS

The containerized Open5GS deployment has the following dependencies:

- K8s environment as a container orchestrator.
- K8s cluster and Helm Chart version 3.
- Calico CNI - Container Network Interface plugin.
- Open5GS Docker image.

The K8s 5G SA core functions are implemented through the deployment of Open5GS. This incorporates several network functions to facilitate the deployment of the 5G core functions and integrated with Amarisoft RAN (gNB) in a K8s format. The deployment Open5GS SA Core contains set of functions such as: NRF, Service Communication Proxy (SCP), AMF, SMF, UPF, AUSF, UDM, UDR, PCF, NSSF, and Binding Support Function (BSF).

The core of 5G SA employs Service-Based Architecture (SBA), wherein control plane functions are configured to register with the NRF. The NRF in turn assists in the discovery of other core functions. The AMF takes care of connection and mobility management and gNBs (5G basestations) are connected to the AMF. The UDM, AUSF, and UDR are responsible for generating Subscriber Identity Module (SIM) authentication vectors and holding subscriber profiles. Session management is handled by the SMF. The NSSF provides network slice selection, while PCF is used for enforcing subscriber policies and charging. Lastly, the SCP facilitates indirect communication. The user plane of the 5G SA core comprising just one function. The UPF is responsible for transporting user data packets between the gNB and the external Wide Area Network (WAN), and it is also connected to the SMF.

Similar to the generic deployment, the containerized Open5GC can support 3GPP Release 16, Universal SIM (USIM) cards (using Milenage), and multiple Protocol Data Unit (PDU) sessions.

6 Deployment issues

Most of the issues faced during the deployment of the different UCs in the different platforms have been related to the networking. In general, the solutions were not fully containerised before this project, or had only been tested in single node K8s environments. When deployed in 5G-EPICENTRE testbeds, strange behaviour was detected in some of the pods. Internal processes led to debug the problems found, to enable the solutions to work in this kind of environment. Here, the most relevant fixes applied during the different deployments are provided.

6.1 Networking issues

In UC4, some of the services require to be accessible via FQDN. In order to achieve that, a CoreDNS addition should be provided. During the tests, access to those services has been possible by specifying direct IP address in dedicated ConfigMap, but this requires a subsequent deploy of a single file after IP assignment, provided by the Load Balancer.

In UC3, a direct connection of the application server, running the QGroundcontrol App, to the 5GC is necessary.

6.2 Storages issues

NEM's MCX solution requires storage volumes, where the information of the different tenants, groups, users, *etc*. should be stored. This information must be stored even if the service is re-instantiated or restarted, otherwise deployment's provisioning must be done again. This is done using persistent volumes, and pods specifically dedicated to this task. The new multi-mode context, to which the solution has been exposed, has led to problems with these volumes, which have been fixed.

Even though it is not mandatory, to achieve its full capacities, ONE's Mobitrust solution also requires persistent storage. This ensures that relevant information that is not loaded at startup (*i.e.*, changes to users, devices, policies, *etc*.) is kept, even during power failures, reboots and other events, where volatile storage is cleared. The provisioning of storage at the K8s cluster is performed using PVCs. The dynamic provisioning of PVCs in the two testbeds where UC4 is deployed has faced some challenges, due to different storage backends, and their integration with K8s, which have been overcome thanks to the joint efforts of the involved partner teams.

6.3 Security issues

Pods' excessive weight has also been detected, after service's long-runs testing. There is an ongoing work to address these issues in order to improve the deployment time KPI. The weight of each pod is being lightened when possible, and the solution presented in D2.1 and D2.7 is being improved.

6.4 Other issues

The cluster hosting the deployment of the UC4 Network Application needs to map all related services to the Meritrust's own domain (mobitrust.org). Since clusters are multi-tenant, and have multiple applications, changing the domain of the cluster is not a viable solution. Upon trying to find a simple solution that does not bypass K8s CoreDNS (and still uses it for internal communication), the easiest fix was a change to CoreDNS configuration, that maps all FQDNs of services in UC4 namespace to another domain (mobitrust.org), and vice-versa.