5G experimentation environment for 3rd party media services

D3.3 System components, integration, configuration and testing - Initial

Document Summary Information

Grant Agreement No	101016714	**Acronym**	5GMediaHUB
Full Title	5G experimentation environment for 3rd party media services		
Start Date	01/01/2021	**Duration**	39 months
Project URL	www.5gmediahub.eu		
Deliverable No/Title	D3.3 System components, integration, configuration and testing - Initial		
Related Work Package	WP3	**Related Task**	T3.2 – System components, integration, configuration and testing
Contractual due date	31/5/2022	**Actual submission date**	31/5/2022
Type	Report	**Dissemination Level**	Public
Deliverable Editor	Kostis Tzanettis (AppArt)		
Contributors	George Margetis (FORTH), Stefania Stamou (FORTH), Martin Tolan (WIT), Kostis Tzanettis (APPART), Kostas Chalatsis (APPART), Michael Sfakianos (APPART), Andreas Kartakoullis (EBOS), Kostas Ramantas (IQU)		
Peer Reviewers	Andrea Castelli (BRA), TNOR		

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 101016714

Revision history (including peer reviewing & quality control)

Version	Issue Date	% Complete	Changes	Contributor(s)
v0.1	19/10/2021	0	Initial Deliverable Structure	Kostis Tzanettis (AppArt)
v0.2	01/04/2022	0	Updated Deliverable Structure	Kostis Tzanettis (AppArt)
v0.3	03/05/2022	10	Overall architecture of the experimentation facility (Section 2)	Kostis Tzanettis (AppArt)
v0.4	04/05/2022	20	Added the Experimentation Facility Evaluation Methodology (Section 4)	Stefania Stamou (FORTH), George Margetis (FORTH)
v0.5	06/05/2022	30	Added section for the NetApps Repo (Section 3.1.1)	Martin Tolan (WIT)
v0.6	06/05/2022	40	Added section for the User Management Module (Section 3.1.2)	Martin Tolan (WIT)
v0.7	9/05/2022	50	Added section for the Experimenters Portal Module (Section 3.1.3)	Andreas Kartakoullis (EBOS)
v0.8	9/05/2022	60	Added section for the Test Planning & Reservation Engine (Section 3.1.4)	Kostas Ramantas (IQU)
v0.9	10/05/2022	70	Added section for the Validation Testing & Verification Engine (Section 3.1.5)	Kostis Tzanettis (AppArt), Michael Sfakianos (AppArt), Kostas Chalatsis (AppArt)
v1.0	12/05/2022	80	Added section for the Continuous QoS/QoE Monitoring Engine (Section 3.1.6)	Martin Tolan (WIT)
v1.1	16/05/2022	90	Added the Summary, Introduction and Conclusion sections	Kostis Tzanettis (AppArt)
v1.2	18/05/2022	93	Final updates before peer review	Kostis Tzanettis (AppArt)
v1.3	25/05/2022	98	Peer Review & Quality Assurance	Andreas Kartakoullis (EBOS)

v1.4	27/05/2022	100	Final adjustments before submission	Kostis Tzanettis (AppArt)

Disclaimer

Copyright message

Table of Contents

List of Figures

List of Tables

Listings

Glossary of terms and abbreviations used

Abbreviation / Term	Description
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Delivery
CRUD	Create, Read, Update and Delete
ExP	Experimenters' Portal
FQDN	Fully-Qualified Domain Name
HTTP	Hypertext Transfer Protocol
QoSE	Quality of Service / Experience
MANO	Management and Orchestration
OSM	Open Source MANO
REST	Representational State Transfer
TPRE	Test Planning & Reservation Engine
UMM	User Management Module
V&VE	Validation Testing & Verification Engine

Executive Summary

Through this deliverable, the user can get an overview of the architecture of the Experimentation Facility, it's various components with their configuration and deployment strategy along with the integration between them. It is composed of six modules / engines, each one delivering a specific set of requirements as described within the project. These modules are:

- NetApps Repository;
- User Management Module;
- Experimenters Portal;
- Test Planning & Reservation Engine;
- Validation Testing & Verification Engine;
- Continuous QoS / QoE Monitoring Engine.

The experimentation facility will be hosted in a Kubernetes cluster consisting of one master node and three worker nodes. Each module has its Kubernetes namespace and service name and may use a persistence storage if required. Each one has its own configuration and deployment process which currently involves manual effort. The plan is to automate this process (deployment, configuration, testing) through a common repository and CI/CD pipelines until the next deliverable.

Based on the facility requirements specified in D1.3 "Functional requirements elicitation and analysis – Initial" [2], the platform will undergo a series of testing to verify it delivers what is described. The testing lifecycle includes five steps with the first one being setting the requirements (as delivered by D1.3), the second one the planning of the testing strategy, the third one preparing the scenarios for testing each component and the final two being the test execution and the reporting of the results to the end users. Tests will be both automated (in the form of unit and integration tests) and manual that include the overall system testing and the user acceptance tests.

1 Introduction

This document is the first of a series of two deliverables (D3.3 & D3.4) called "*System components, integration, configuration and testing*" related to the homonym task T3.2. The objective of this task is to perform the integration, configuration and evaluation of the Experimentation Facility components as described in task T1.4 which includes the three experimentation engines (i.e., validation & verification engine, continuous QoS/QoE monitoring engine, test planning & definition engine), the user management module, the NetApps repository and finally the experimenters' portal.

This first drop includes information regarding the activities of the task up to now. Its delivery coincides with the initiation of the first testing cycle thus information described here is what is going to be available during cycle one. The second drop - D3.4 - is due M29 (May 2023) and coincides with the task ending thus the final configuration and integration aspects are expected to be described in this second drop. It will be based on this first drop with updates and improvements of the overall platform setup and configuration.

Through this deliverable series, the reader can:

- Get an overview of the experimentation facility components;
- Get information regarding the integration and configuration of each one of the components;
- Get an overview of the facility evaluation methodology that is going to be applied to verify proper operation and requirement verification.

1.1 Mapping 5GMediaHUB Outputs

Purpose of this section is to map 5GMediaHUB's Grant Agreement commitments, both within the formal Deliverable and Task description, against the project's respective outputs and work performed.

Table 1: Adherence to 5GMediaHUB's GA Deliverable & Tasks Descriptions

GA Component Title	**GA Component Outline**	**Respective Document Chapter(s)**	**Justification**
TASKS			
T3.2 – System components, integration, configuration and testing	The objective of this task is to perform the integration, configuration and evaluation of the Experimentation Facility components, as defined in the system architecture of T1.4, including the three experimentation engines (i.e., test planning & definition, validation & verification, continuous QoS/QoE monitoring), the NetApps Repository, and the Experimenters Portal. In this respect, the partners (i.e., FORTH, WIT, EBOS, IQU led by APART) who will primarily develop the constituent components of the system will be involved in this task. Specifically, CI/CD pipelines will be	Section 2 & Section 3	Section 2 gives an overview of the experimentation facility integration diagram whereas Section 3 describes in detail the integration points of all modules.

	implemented leveraging the experimentation engines and NetApps repository, facilitating automated building of code, unit and functional testing, as well as continuous QoS and QoE performance tests.		
T3.2 – System components, integration, configuration and testing	Furthermore, the Experimenters Portal will be integrated with the experimentation Engines and NetApps Repository, offering an easy-to-use GUI to experimenters for uploading NetApps code, running tests and visualising results.	Section 3	Section 3 describes in detail the integration points of the experimenters' portal and all the other components
T3.2 – System components, integration, configuration and testing	These experimentation components will be deployed at the 5GMediaHUB "experimentation cloud" facilities, which include compute and storage infrastructure to be offered by CTTC, and their compliance with the requirements and specifications defined in T1.2 will be ensured.	Section 3 & Section 4	Section 3 describes in detail the hardware resources required per module while Section 4 describes the methodology used to test the overall facility to verify compliance with the requirements set in T1.2.
DELIVERABLE			
D3.3 System components, integration, configuration and testing - Initial Initial version of integrated Experimentation Tools components.			

1.2 Deliverable Overview and Report Structure

This deliverable has been structured in a top to bottom approach by giving to the reader the general picture of the components and sub-components of the facility before diving into more detail about the configuration and integration between the modules. It provides all the necessary details for the reader to understand the integration points, the configuration required for each component and also the evaluation process to verify compliance with the requirements set in T1.2. The sections of this deliverable are:

- **Section 1 "Introduction":** Introduces the reader into the document, provides a mapping between the description of T3.2 and the sections of this deliverable and finally gives an overview of the sections included;
- **Section 2 "*Experimentation Facility Overall Architecture*":** Provides the experimentation facility's integration design and each module's / engine's comprising sub-components along with their interconnection;
- **Section 3 "Experimentation Facility Module Integrations & Configuration":** Detail design of the module configuration and integration points;
- **Section 4 "Experimentation Facility Evaluation Methodology":** Describes the testing framework for verifying the functionality of the facility's components against technical specifications and requirements;

- **Section 5 "Conclusions":** Summarises the results achieved up to now in this task and within this deliverable and points out the next actions during the following months towards the updated second deliverable (D3.4) of this task.

2 Experimentation Facility Overall Architecture

A high-level architecture of the experimentation facility was initially provided in Figure 19 of the first deliverable of the architecture, namely *D.1.7 "Architecture Design & Technical Specifications – Initial" [1]*. In this section, a deeper design is provided showing each module's / engine's comprising sub-components along with their interconnection as seen in Figure 1.

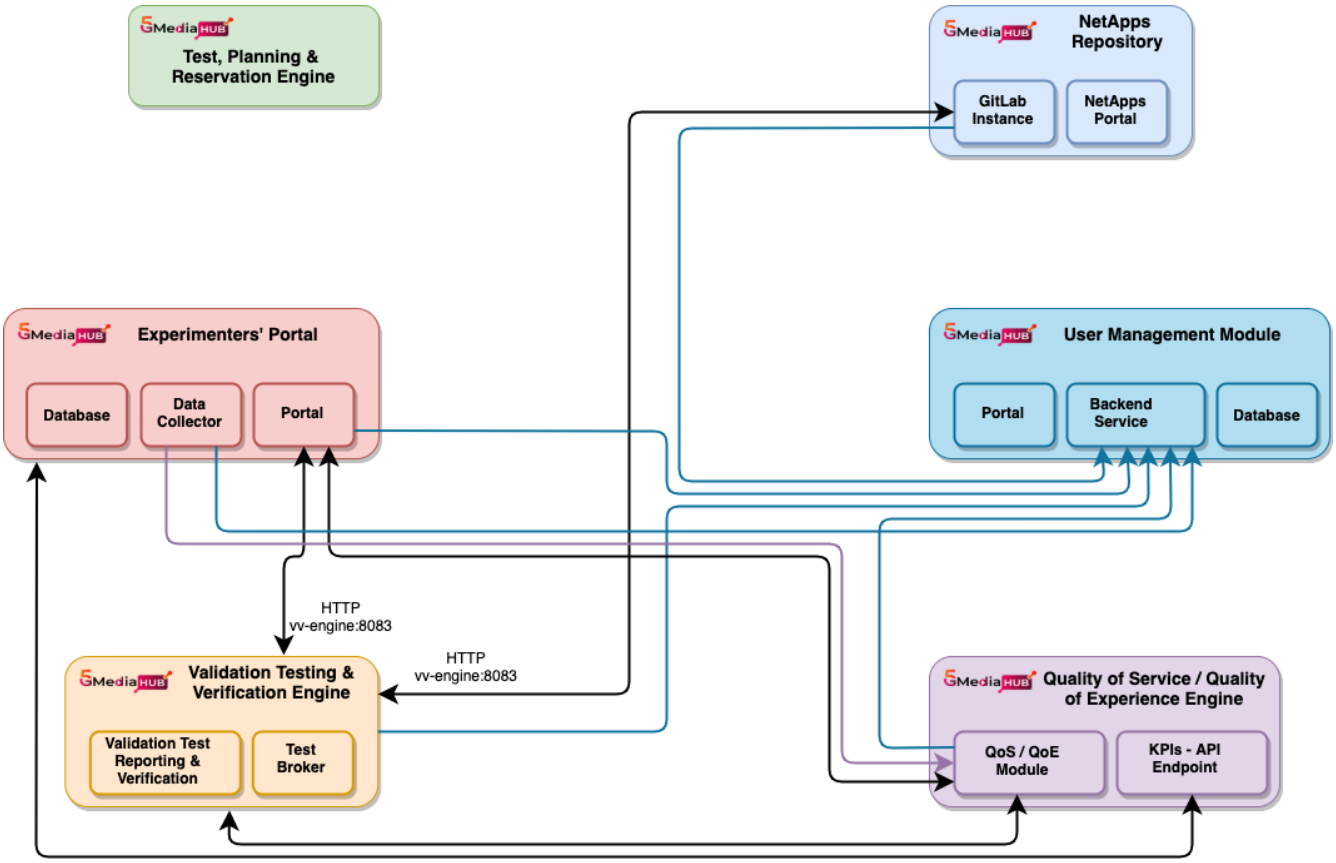

Figure 1: Experimentation Facility Overall Architecture

The modules described here are:

- **User Management Module (UMM):** module implemented for managing module access from one common module. Consists of (i) the portal (for user interaction), (ii) the backend service (for automating requests) and (iii) an internal database. All modules other than the "Test, Planning & Reservation Engine" integrate with the backend service of the UMM;
- **Experimenters Portal (ExP):** portal for experimenter users. Consists of 3 submodules, (i) the user portal for human interaction, (ii) the data collector for collecting the KPIs from the use cases and (iii) the internal database. The data collector integrates with the UMM, while the portal integrates with the V&VE, the QoSE Engine and the UMM;
- **Test, Planning & Reservation Engine (TPRE):** responsible for reserving the necessary resources for running an experiment;
- **Validation, Testing & Verification Engine (V&VE):** engine for validating NetApps according to the specifications set by the NetApp developer. Consists of (i) the *Validation Test Reporting & Verification* for checking out and triggering the NetApp pipeline and (ii) the *Test Broker* for executing the actual steps in the pipeline. This engine integrates with the ExP for triggering the pipeline, the UMM for authorization and the NetApps Repo for exporting the NetApps code/artefact and populating the validation result.

- **Continuous QoS/QoE Monitoring Engine (QoSE Engine):** engine for calculating the necessary KPIs. Raw data are collected by the data collector (sub-component of experimenter's portal) and the produced KPIs (QoS / QoE module) are exposed through REST APIs (KPIs-API Endpoint) to 3rd party applications. The QoSE portal is integrated with the UMM and the ExP;
- **NetApps Repository (NetApps Repo):** the module responsible for designing, storing and deploying the NetApps. Consists of (i) the GitLab instance and (ii) the NetApps portal through which one can manage and implement a NetApp. The NetApps repository integrates with the UMM and V&VE.

3 Experimentation Facility Module Integrations & Configuration

All aforementioned modules are going to be deployed on a Kubernetes cluster that formulates the experimentation facility. Though the description of the cluster itself is part of the upcoming deliverable *D3.5 "Interfacing and configuration of 5GMediaHUB Experimentation Tools and NetApps with 5G testbeds"*, for the purpose of this section, a high-level description of the cluster is provided in Figure 2.

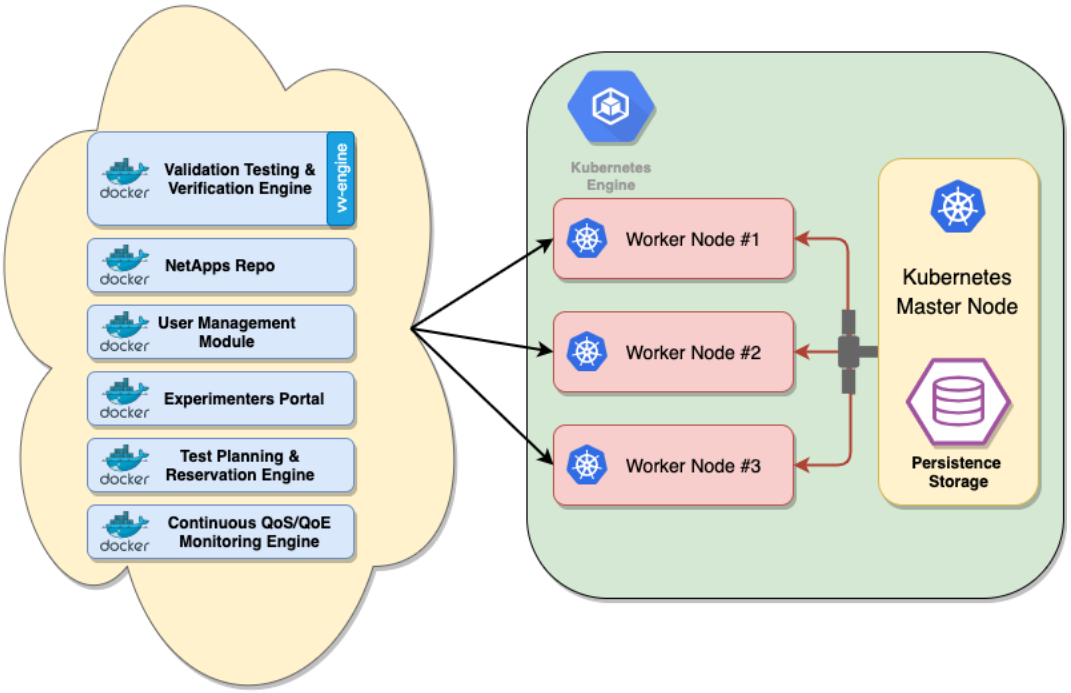

Figure 2: Deployment architecture of the experimentation facility

The Kubernetes cluster consists of one master node responsible for the cluster management and three worker nodes that will host all six modules. Each module has its service, it's namespace and, depending on whether persistency is required, a persistence storage is also available. Details of this configuration can be seen in Table 2.

Table 2: Module network & storage Kubernetes configuration

Module	Short Name	Namespace	Service Name	Persistence Storage
Validation & Verification Engine	*V&VE*	*vv-engine*	*vv-engine-service*	pv-mediahub
NetApps Repo	*NetApps Repo*	*netapps-repo-app-wit*	*netapps-repo-app-wit-service*	*pv1-mediahub*
User Management Module	*UMM*	*umm-wit*	*umm-wit-service*	*pv1-mediahub*
Experimenters Portal	*ExP*	*exp-portal*	*exp-portal-service*	*pv2-mediahub*
Test Planning & Reservation Engine	*TPRE*	-	-	-

Continuous QoS/QoE Monitoring Engine	*QoSE*	*qose-engine-wit*	*qose-engine-service*	*pv1-mediahub*

3.1 Modules Deployment & Integrations

This section focuses on giving the reader with an overview of all the necessary information regarding the configuration and hardware resources requirements of each module / engine. A detailed matrix of all the necessary integrations is also provided per module giving the reader an overview of the integration flows and the protocols. Finally, the deployment process is also described for each component.

3.1.1 NetApps Repository Module

The NetApps Repository Application is the primary portal that application developers will use to design, build and deploy their NetApps. The NetApps Repository can be mostly viewed as a standalone application but there are several functionalities that are intertwined with several of the other 5GMediaHub framework components. There are several features associated directly with this application, namely:

- Portal frontend application
- Portal backend service with API for remote invocation
- Backend repository (Gitlab instance)
- Service Catalogue Application (Provides OSM API Abstraction)

As NetApps are designed and their implementation matures the storage of those NetApps are via a specific instance of GitLab the provide both storage and triggering of the SOL-006 validation of the NetApps' scripts. Also provided is the ability to deploy and on-board the NetApp to the target 5G testbed vis the service catalogue application (which itself abstracts the underlying OSM APIs). Other framework components can interact with the NetApps repository directly via its exposed APIs which are mainly for triggering a NetApps on-boarding process as a pre-cursor to a test cycle execution. Also, similar to other 5GMediaHub components, the NetApps Repository application is integrated with the User Management Module as the UMM providing authentication and authorization for users and module access across all the set of 5GMediaHUB components.

3.1.1.1 Integration Endpoints

Table 3: NetApps Repo Integration Matrix

Source System / IP	Target System / IP	Protocol / Port	Functionality
NetApps Repo Application	*UMM*	*HTTP REST APIs / 80*	*The NetApps repository module calls the UMM for authentication/authorization*
NetApps Repo Application	*Gitlab Instance*	*HTTP REST APIs / 80/443*	*The NetApps repository module calls the GitLab APIs for source files management (project management, files storage related activities (CRUD) and pipeline execution.*
NetApps Repo Application	*Service Catalogue Application*	*HTTP REST APIs / 80/443*	*The NetApps repository module calls the Service Catalogue APIs as a wrapper around the set of exposed OSM northbound APIs as an abstraction layer.*

3.1.1.2 Configuration

The applications that constitute the NetApps Repository applications all take the form of containerized docker images that have been divided up based on a microservices architecture. Any services that require custom configurations are achieved by updating the Docker file associated with each of the images. The following sub-sections captures the Docker files for those components.

- **NetApps Portal Backend**

The NetApps portal backend application is written in Node JS using TypeScript and the docker file will first of all build the applications before creating the final version of the image with the newly built application on-boarded. This can be seen in Listing 1.

Listing 1: NetApps Portal Backend Docker file

```
FROM node:16-alpine AS build-deps
COPY package*.json .npmrc /build/
WORKDIR /build
RUN npm install
RUN rm -f .npmrc

# 2nd stage for compiling typescript
FROM node:16-alpine AS compile-env
RUN mkdir /compile
COPY --from=build-deps /build /compile
WORKDIR /compile
COPY . .
RUN npm run build

# 3rd stage for installing runtime dependencies
FROM node:16-alpine AS runtime-deps
COPY package.json .npmrc /build/
WORKDIR /build
RUN npm install --production
RUN rm -f .npmrc

# Final stage for running application
FROM node:16-alpine AS runtime-env
WORKDIR /app
# copy compiled artefacts
COPY --from=compile-env --chown=node:node /compile/build /app
# copy production dependencies
COPY --from=runtime-deps --chown=node:node /build /app
# Create logs directory
RUN mkdir /app/logs
RUN chown node:node /app/logs
# copy UMM config
COPY config.production.json /app/
# copy server config
COPY .env* /app/
USER node
EXPOSE 5080
ENV NODE_ENV=production
CMD [ "node", "app.js" ]
```

- **NetApps Portal Frontend**

The NetApps Portal frontend is also a Node JS written in TypeScript where the entire design canvas for the NetApps and user interface is implemented. The container design for the frontend portal application is captured in Listing 2.

Listing 2: NetApps Portal Frontend Docker file

```
FROM node:alpine AS builder
WORKDIR /app
COPY package*.json /app/
RUN npm ci
COPY ./ /app/
ARG NODE_OPTIONS=--openssl-legacy-provider
RUN npm run build

FROM node:alpine
WORKDIR /app
#!/bin/sh
RUN npm i -g serve
COPY --from=builder /app/build /app/
COPY .env .env.production /app/
EXPOSE 3006
ENTRYPOINT ["serve", "-s", "/app/", "-l", "3006"]
```

3.1.1.3 Deployment Architecture

A GitLab instance that is hosted by WIT is utilised as the code repository for the set of NetApps Repository application components as well as providing the facility to automate the CI pipeline for building the containers once updated application code has been checked in. The build containers are available for deployment within the container registry, also provided by the same GitLab instance and has already been integrated with the CTTC Kubernetes cluster. The docker images, once the CI Pipeline has successfully completed its task, are ready to be deployed to the cluster. The deployment files specific to the Kubernetes cluster are uploaded to the master node in the cluster and are executed directly on the master. The deployment file specifies the origin of the docker images as being the GitLab container registry and this allows for simple container management and tracking while also reducing the load on the master nodes within the cluster. The deployment on the cluster takes the following steps where some of the steps are only performed once while others are performed every time the image is re-deployed:

1. Create Namespace for applications to cluster
2. Add GitLab container registry pull credentials' secret to cluster/namespace
3. Create Service Accounts to cluster/namespace
4. Deploy application consisting of the following sub-items:
    a. Create the application deployment targeting the docker image container
    b. Create the Kubernetes service used to host this application
    c. Create the ingress used to provide the networking access for the application

Steps 1 – 3 are only executed the first time for a new application while all of the steps in 4 are executed for each new iteration of the application to be deployed.

- **Deployment Setup: Namespace**

The script file for the namespace is captured in Listing 4 and is executed by the following code in Listing 3

Listing 3: Kubernetes command to create NetApps Repository Namespace

```
kubectl apply -f netappsrepo-createnamespace.yaml
```

Listing 4: Script to create Namespace for the NetApps Repository

```
apiVersion: v1
kind: Namespace
metadata:
  name: netapps-repo-app-wit
```

The service account for the namespace is created to allow for the management of the applications within the namespace with the appropriate permissions applied. The script for the service account is captured in Listing 5 below.

Listing 5: Script to create the Service Account for the NetApps Repository

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: netapps-repo-app-wit-admin
  namespace: netapps-repo-app-wit
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["*"]

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: netapps-repo-app-wit-admin
  namespace: netapps-repo-app-wit

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: netapps-repo-app-wit-admin
  namespace: netapps-repo-app-wit
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: netapps-repo-app-wit-admin
subjects:
- kind: ServiceAccount
  name: netapps-repo-app-wit-admin
  namespace: netapps-repo-app-wit
```

- **Deployment of the Application**

A description of the deployment is required that describes the application, the origin of its image and any cluster setting required such as resources (CPU, Memory, etc.), hardware (Storage, Persistence, etc.) and networking (Protocols, ports, etc.) requirements. The deployment is broken up into three steps:

- Deployment of the image to the cluster/namespace
- Configuration of the service representing this application

- Ingress configuration defining the networking connectivity

The application deployment script is captured in Listing 6.

Listing 6: NetApps Repository Frontend Deployment Resource

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: netapps-repo-frontend-5gmedhub
  namespace: netapps-repo-app-wit
spec:
  replicas: 1
  selector:
    matchLabels:
      app: netapps-repo-frontend-5gmedhub
  template:
    metadata:
      labels:
        app: netapps-repo-frontend-5gmedhub
    spec:
      securityContext:
          fsGroup: 1000
          runAsUser: 1000
      imagePullSecrets:
      - name: wit5gmedhubglregcred
      serviceAccountName: netapps-repo-app-wit-admin
      containers:
      - name: netapps-repo-frontend-5gmedhub
        image:              gitlab.netapps-5gmediahub.eu:5050/5gmediahub-platform/netapps-
repository/net-apps-portal-frontend:latest
        ports:
        - name: httpport
          containerPort: 80
```

The definition of the service for the application is defined in Listing 7.

Listing 7: NetApps Repository Frontend Service Resource

```
apiVersion: v1
kind: Service
metadata:
  name: netapps-repo-frontend-5gmedhub-service
  namespace: netapps-repo-app-wit
  labels:
    app: netapps-repo-frontend-5gmedhub
spec:
  selector:
    app: netapps-repo-frontend-5gmedhub
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 80
```

The ingress design for the service is details in Listing 8.

Listing 8: NetApps Repository Frontend Ingress Resource

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: netapps-repo-frontend-5gmedhub
  namespace: netapps-repo-app-wit
  #annotations:
  #    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
          #- host: services.5gmediahub.eu
   - http:
      paths:
      - pathType: Prefix
        path: "/netappsrepofe"
        backend:
          service:
            name: netapps-repo-frontend-5gmedhub-service
            port:
              number: 80
```

The resource scripts for the remaining NetApps Repository application components are available in Annex I – *Kubernetes Resource Scripts for the 5GMediaHub Components*, under Listing 30, Listing 31 and Listing 32.

3.1.1.4 Hardware Resources

The NetApps Repository application, as a whole is a cloud-based application that has specific requirements around storage, internet access and web service deployment environment. As the containers are deployed as web services within the hosting Kubernetes cluster the specific additional requirements are captured in Table 4.

Table 4: NetApps Repository Deployment Hardware Requirements Overview

(Sub)Module	RAM	Storage space	Network requirements
NetApps Frontend	250MB	N/A	Internet access required over FQDN
NetApps Backend	250MB	N/A	Internet access required over FQDN
NetApps GitLab Repository Instance	500MB	10GB	Private Repository with internal network access within the NetApps Repository namespace

3.1.2 User Management Module

The User Management Module (UMM) provide the set of authentication and authorization backend functionality required for all users intending on utilizing the 5GMediaHub experimenters' platform. The UMM also provides the level of authorization individual users may have depending on the actual framework application that the user is interacting with. This helps to control, with a fine-grained level of detail, the integrity of the applications and the functionality accessible to users within those applications. As all of the platform components require access to the UMM, the UMM must be deployed in a manner that is accessible to those components without incurring a significant overhead for communications. To mitigate any delays the UMM is to be co-located within the

deployment infrastructure along with the other 5GMediaHub applications, components & services, within its own namespace. It shall be publicly available over the internet allowing for administrator access for configuration of users and applications without any esoteric knowledge of the infrastructure its deployed upon. The main components that constitute the UMM are the following:

- UMM frontend portal application
- UMM backend service with APIs for programmatic access
- UMM database

The frontend application allows for the administrator to add new users and organisations to the portal and assign the appropriate rights to each via its portal front end user interface. The backend service exposes a Rest API that allows for other application & services to validate a users' credentials and also to query the access rights for users with respect to the client application executing the request. The UMM database hosts all of the information such as user data and applications configurations with respect to authorisation levels and accesses. The database is pivotal to the integrity of the UMM as a whole and as such its contents require persistence across the entire deployment infrastructure.

3.1.2.1 Integration Endpoints

The User Management Module is a self-contained component that has no dependencies with any external components. The other applications and services within the 5GMediaHub experimenters' platform will be utilizing the UMM but the details surrounding this integration are documented within each section for those applications.

3.1.2.2 Configuration

All of the components that comprise the overall UMM application, are designed as docker containers that can be easily deployed to the deployment infrastructure. The design of the components, are contained within their particular Docker file including their specific application that is included as part of the container image itself. The following sections describe each of the images for the overall UMM application.

- **UMM Portal Frontend**

The application for the UMM Frontend is written using Typescript in NodeJS. The application is built by the Gitlab pipeline hosting the code and the docker image is built including the application artefacts. The Docker file can be seen in Listing 9.

Listing 9: UMM Portal Frontend Docker file

```
FROM node:alpine AS builder
WORKDIR /app
COPY package*.json .npmrc /app/
RUN npm ci
COPY ./ /app/
ARG NODE_OPTIONS=--openssl-legacy-provider
RUN rm -f .npmrc
RUN npm run build
FROM node:alpine
#!/bin/sh
RUN npm i -g serve
COPY --from=builder /app/build /app/
EXPOSE 3000
ENTRYPOINT ["serve", "-s", "/app/", "-l", "3000"]
```

- **UMM Backend Service**

Again, the UMM backend service is written in Typescript with NodeJS and the Docker file can be seen in Listing 10.

Listing 10: UMM Backend Service Docker file

```
FROM node:16-alpine AS build-deps
COPY package*.json .npmrc /build/
WORKDIR /build
RUN npm install
RUN rm -f .npmrc

# 2nd stage for compiling typescript
FROM node:16-alpine AS compile-env
RUN mkdir /compile
COPY --from=build-deps /build /compile
WORKDIR /compile
COPY . .
RUN npm run build

# 3rd stage for installing runtime dependencies
FROM node:16-alpine AS runtime-deps
COPY package.json .npmrc /build/
WORKDIR /build
RUN npm install --production
RUN rm -f .npmrc

# Final stage for running application
FROM node:16-alpine AS runtime-env
WORKDIR /app
# copy compiled artefacts
COPY --from=compile-env --chown=node:node /compile/build /app
# copy production dependencies
COPY --from=runtime-deps --chown=node:node /build /app
# Create logs directory
RUN mkdir /app/logs
RUN chown node:node /app/logs
# copy UMM config
COPY config.json .
# copy server config
COPY .env* /app/
USER node
EXPOSE 6000
CMD [ "node", "index.js" ]
```

3.1.2.3 Deployment Architecture

The deployment process for the UMM Application is similar to that of the NetApps Repository Application as documented in Section 3.1.1.3. The GitLab instance hosts all of the UMM applications' source code, configuration files & scripts and the applications' artefacts are built using the pipelines once the code has been updated in the repository. The resultant containers can then be deployed to the deployment infrastructure within the target Kubernetes cluster. The UMM has its own specific namespace within the cluster and this can be seen in Listing 11.

Listing 11: Namespace resource for the UMM Application Frontend

```
apiVersion: v1
kind: Namespace
metadata:
  name: umm-wit
```

The resource for creating the service account for the namespace is captured in Listing 12 while the deployment of the entire frontend application, frontend service and ingress are captured in Listing 13, Listing 14 and Listing 15.

Listing 12: Kubernetes resource to create the Service Account for the UMM Application Frontend

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: umm-wit-admin
  namespace: umm-wit
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["*"]

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: umm-wit-admin
  namespace: umm-wit

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: umm-wit-admin
  namespace: umm-wit
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: umm-wit-admin
subjects:
- kind: ServiceAccount
  name: umm-wit-admin
  namespace: umm-wit
```

Listing 13: UMM Portal Frontend Deployment Resource

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: umm-frontend-5gmedhub
  namespace: umm-wit
spec:
  replicas: 1
  selector:
    matchLabels:
      app: umm-frontend-5gmedhub
  template:
    metadata:
      labels:
        app: umm-frontend-5gmedhub
    spec:
      securityContext:
          fsGroup: 1000
          runAsUser: 1000
      imagePullSecrets:
      - name: wit5gmedhubglregcred
      serviceAccountName: umm-wit-admin
      containers:
      - name: umm-frontend-5gmedhub
        image:      gitlab.netapps-5gmediahub.eu:5050/5gmediahub-platform/user-management-
module/umm-portal:latest
        ports:
        - name: httpport
          containerPort: 80
```

Listing 14: UMM Portal Frontend Service Resource

```
apiVersion: v1
kind: Service
metadata:
  name: umm-frontend-5gmedhub-service
  namespace: umm-wit
  labels:
    app: umm-frontend-5gmedhub
spec:
  selector:
    app: umm-frontend-5gmedhub
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 80
```

Listing 15: UMM Portal Frontend Ingress Resource

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: umm-frontend-5gmedhub
  namespace: umm-wit
  #annotations:
  #    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
          #- host: services.5gmediahub.eu
   - http:
      paths:
      - pathType: Prefix
        path: "/ummportal"
        backend:
          service:
            name: umm-frontend-5gmedhub-service
            port:
              number: 80
```

The Kubernetes resources required for the deployment of the set of UMM backend are available in Annex I – *Kubernetes Resource Scripts for the 5GMediaHub Components*, under Listing 33, Listing 34 and Listing 35.

The UMM Portal database is required to persist the data across Kubernetes pod and node boundaries. To facilitate this the database is based on a PostgresDB database where its data is mapped to a persistent volume claim that is facilitated by the configuration of the Kubernetes cluster. The password for the database is added to the cluster as a secret and prevents leakage by not having to place it into the deployment resources scripts for the database. The resource script for the database service and stateful set are captured in Listing 16 and Listing 17.

Listing 16: UMM Portal Database Kubernetes Stateful Set Resource

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: umm-db-5gmedhub
  namespace: umm-wit
spec:
  serviceName: umm-db-5gmedhub-service
  selector:
    matchLabels:
      app: umm-db-5gmedhub
  replicas: 1
  template:
    metadata:
      labels:
        app: umm-db-5gmedhub
        selector: umm-db-5gmedhub
    spec:
      volumes:
        - name: postgresdb-pv-storage
          persistentVolumeClaim:
            claimName: postgresdb-persistentvolumeclaim
      containers:
      - name: umm-db-5gmedhub
        image: postgres
        ports:
        - containerPort: 5432
          name: "postgresdb-port"
        volumeMounts:
          - mountPath: "/var/lib/postgresql/data"
            subPath: pgdata
            name: postgresdb-pv-storage
        env:
          - name: POSTGRES_USER
            value: 5gmediahub
          - name: POSTGRES_PASSWORD
            valueFrom:
              secretKeyRef:
                name: db-credentials
                key: db-password
          - name: POSTGRES_DB
            value: 5gmediahub
          - name: PGDATA
            value: /var/lib/postgresql/data/pgdata
```

Listing 17: UMM Portal Database Kubernetes Service Resource

```
apiVersion: v1
kind: Service
metadata:
  name: umm-db-5gmedhub-service
  namespace: umm-wit
  labels:
    app: umm-db-5gmedhub
spec:
  selector:
    app: umm-db-5gmedhub
  ports:
  - name: http
    protocol: TCP
    port: 5432
    targetPort: 5432
```

3.1.2.4 Hardware Resources

The UMM Portal application is a cloud-based application that has specific requirements around storage, internet access and web service deployment environment. As the containers are deployed as web services within the hosting Kubernetes cluster the specific additional requirements are captured in Table 4.

Table 5: UMM Portal Application Deployment Hardware Requirements Overview

(Sub)Module	RAM	Storage space	Network requirements
UMM Portal Frontend	250MB	N/A	Internet access required over FQDN
UMM Portal Backend RestAPI	250MB	N/A	Internet access required over FQDN
UMM Portal Database	500MB	20GB	Private Repository with internal network access within the UMM Portal namespace

3.1.3 Experimenters Portal Module

A key component of the 5GMediaHUB Experimentation Facility is the web-based Portal, which will act as the main entry point to the Facility modules and experimentation tools for various actors and stakeholders. Additionally, the Portal also implements some key functions such as to define the experiment workflow, which allows the setup of experiments for the evaluation of media applications in terms of certain performance KPIs, and guides the users through the process of composing a new experiment (e.g., experiment scheduling, resource reservation, etc.). Furthermore, the Portal hosts the Data Collector module, which records and stores KPIs from experiments that are then visualized in the form of charts and diagrams, helping experimenters assess the performance of their applications.

3.1.3.1 Integration Endpoints

Table 6: Portal Integration Matrix

Source System	Target System	Protocol / Port	Functionality
Portal backend	*UMM*	*HTTP REST / 88*	Grants a secure and easy access to the different kind of actors of the 5GMediaHUB Experimentation Facility.
Portal backend	*NetApps Repo*	*HTTP REST / 88*	*Provides the users with the ability to design, build and on-board NetApps to the CDSO.*
*	*Data Collector*	*HTTPs / 443*	*The data collector will provide a REST API than can be consumed by the use cases in order to ingest data into the experimenters' portal database.*
Portal backend	*V&V Engine*	*HTTP REST / 88*	*Validate and verify NetApps functional and non-functional specifications, i.e., KPIs and thresholds*
Portal backend	*QoSE Engine*	*HTTP REST / 88*	*KPIs and other important metrics collection to be used for visualization*
Portal backend	*Planning & Experimentation engine*	*HTTP REST / 88*	*Experiment instantiation and resources allocation (e.g., UE, 5G slice)*

3.1.3.2 Deployment Architecture

The Portal is currently Dockerized at the EBOS server and all the needed configuration for the integration with the CTTC Kubernetes cluster will be completed during June 2022. Deployment files will be provided to specify the initial point of the docker images located at EBOS server in order to enable the container management at the CTTC Kubernetes cluster. The Portal deployment strategy will be in the area of Blue/Green deployments that is a form of progressive delivery where a new version of the application is deployed while the old version still exists. The two versions will exist together for a brief period until all the users are routed to the new version and thereafter the old version will be discarded.

3.1.3.3 Hardware Resources

Table 7, specifies the computational resources required within the Kubernetes clusters segregated per submodule.

Table 7: Portal and Data Collector Hardware Resources

Submodule	RAM	Storage space	Network requirements
Data Collector	4GB	500MB	Accessible to the use case components
Portal	4GB	50GB	Internet access

3.1.4 Test, Planning & Reservation Engine Module

This is a standalone module of the Experimentation facility and as such there are no integrations with the other modules. As such, all the necessary information for its configuration and implementation is described in *D2.5 "Test Planning & Definition Engine Implementation - Initial" [3].*

3.1.5 Validation, Testing & Verification Engine Module

The Validation Testing & Verification Engine is the primary tool for validating the NetApps utilized within the project. It. Through the CI/CD pipelines pushed to the NetApps repository along with the source code and artefacts, it provides the necessary tools for validating them through a list of KPIs and relevant thresholds. Once the NetApp is onboarded by the NetApps Repo, the V&V engine is triggered to execute the pipeline stages for validating the NetApp. The outcome is also published in the registry for future view from the use case application developers.

3.1.5.1 Integration Endpoints

The Validation, Testing and Verification Engine (V&VE) is integrated with a number of 5GMediaHUB modules, primarily for triggering purposes. The triggering of the verification process is designed to take place from two external systems, besides the internal triggering by the engine itself; the NetApps Repository and the Experimenters Portal can trigger the initiation of the verification process, through the relevant HTTP Web Services provided by the V&V Engine.

Furthermore, as with all other 5GMediaHUB modules, the V&VE is integrated with the User Management Module, as the latter serves the purpose of providing module access, by authentication and authorization, across all 5GMediaHUB components.

While the V&VE is deployed in the Kubernetes clusters, as all of 5GMediaHUB's elements, it also performs deployments itself: The engine leverages the Kubernetes clusters in order to deploy docker containers dynamically, when performing a validation pipeline. In this context, the engine is considered as integrated with the Kubernetes clusters as well.

Table 8 showcases the integration points of the Validation, Testing and Verification engine, along with the respective protocols and ports.

Table 8: V&V Engine Integration Matrix

Source System	Target System	Protocol / Port	Functionality
ExP	V&V Engine	HTTPS / 443	Triggering of the Validation & Verification Engine from the Experimentation portal. This is done through plain HTTPs requests.
NetApps Repo	V&V Engine	HTTPS / 443	Triggering of the Validation & Verification Engine from the NetApps repository. This is done through webhooks (HTTPs requests)
V&V Engine	UMM	HTTPS / 443	Use of UMM's Single Sign on features from the Validation & Verification engine via a REST API.
V&V Engine	Kubernetes clusters	HTTPS / 6443	Deployment of Validation & Verification Engine docker images to the Kubernetes clusters in order to run the Validation & Verification procedures.

3.1.5.2 Configuration

The main configuration of the Validation and Verification engine is stored in a YAML file, which contains a series of needed instructions executed when the module is initialized. This file can be split into five parts based on what each part configures.

- **Kubernetes Cloud**

This configuration allows the Validation and Verification engine to deploy the modules provided by the NetApps into the Kubernetes Cluster. In this section the following main parameters are defined:

- URL of the Kubernetes API;
- Kubernetes namespace where the modules are deployed;
- engine's agents / workers performing the deployments;
- resources needed by the engine's agents.

The above configuration is demonstrated in Listing 18.

Listing 18: V&V Engine Kubernetes Cloud configuration

```
clouds:
 - kubernetes:
    name: "5g-media-hub-k8s"
    serverUrl: "https://${KUBERNETES_ADDR}:6443"
    skipTlsVerify: true
    credentialsId: "5g-media-hub-k8s-credentials"
    namespace: "vv-engine"
    jenkinsUrl: "https://${KUBERNETES_ADDR}/vv-engine/"
    containerCapStr: 42
    maxRequestsPerHostStr: 64
    retentionTimeout: 5
    connectTimeout: 10
    readTimeout: 20
    templates:
     - name: "k8s-agent"
       namespace: "vv-engine"
       activeDeadlineSeconds: 120
       activeDeadlineSecondsStr: 120
       yamlMergeStrategy: "override"
       containers:
        - name: "jnlp"
          image: "jenkins/inbound-agent:latest"
          alwaysPullImage: true
          workingDir: "/home/jenkins"
          ttyEnabled: true
          resourceRequestCpu: "500m"
          resourceLimitCpu: "1000m"
          resourceRequestMemory: "500Mi"
          resourceLimitMemory: "1Gi"
          args: 9999999
          command: "sleep"
          livenessProbe:
          name: "jnlp"
```

```
        volumes:
         - emptyDirVolume:
            memory: false
            mountPath: "/tmp"
        idleMinutes: 1
        activeDeadlineSeconds: 120
        slaveConnectTimeout: 1000
        envVars:
         - envVar:
            key: "KUBERNETES_ADDR"
            value: "https://${KUBERNETES_ADDR}:6443"
```

- **Engine's Users.**

In this section an initial set of the engine users is defined with their permissions. An administrator of the engine is specified, the configuration of which is demonstrated in Listing 19.

Listing 19: V&V Engine Users

```
securityRealm:
   local:
    allowsSignup: false
    users:
     - id: ${JENKINS_ADMIN_ID}
       password: ${JENKINS_ADMIN_PASSWORD}
```

- **3rd Party Credentials.**

The Validation & Verification Engine will integrate with modules such as the NetApps Repo and the Kubernetes infrastructure as it is described above. For these integrations the corresponding credentials are needed. In this section, these credentials are defined, as it is depicted in Listing 20.

Listing 20: V&V Engine 3rd party credentials

```
credentials:
 system:
  domainCredentials:
  - credentials:
     - usernamePassword:
         id: "gitlab"
         password: ${GITLAB}
        scope: GLOBAL
        username: "vv-engine@5gmediahub.eu"
     - kubeconfig:
        description: "k8sConfig"
        id: "k8sConfig"
        kubeconfigSource:
           fileOnMaster:
              kubeconfigFile: "/var/jenkins_home/kubeconfig.yaml"
        scope: GLOBAL
```

In order for the engine to be able to perform deployments on the cluster, the Kubernetes configuration file should be provided. The file, named **kubeconfig.yaml,** is demonstrated in Listing 21.

Listing 21: V&V Engine Kubernetes configuration file

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: ***
    server: https://21.10.200.2:6443
  name: cluster.local
contexts:
- context:
    cluster: cluster.local
    user: kubernetes-admin
  name: kubernetes-admin@cluster.local
current-context: kubernetes-admin@cluster.local
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: ***
    client-key-data: ***
```

- **Libraries**

In this section the location of the global libraries is defined. These libraries provide the necessary functionalities for the engine to execute the validation pipelines of the NetApps. These libraries are stored in the NetApps repo, on a dedicated project and they are written in groovy language. The configuration is achieved by using the configuration shown on Listing 22.

Listing 22: V&V Engine Libraries

```
globalLibraries:
   libraries:
   - defaultVersion: "master"
     name: "vv-engine-library"
     retriever:
       modernSCM:
         scm:
           git:
             credentialsId: "gitlab"
             id: "47afd804-a4a1-415b-820e-07b83b49dcfe"
             remote: "${NETAPP_REPO_URL}/vv-engine-libraries.git"
             traits:
             - "gitBranchDiscovery"
```

- **Job Creation**

In this section of the configuration the main jobs which execute the NetApps pipelines, are created. These jobs are provided to the engine via a groovy script, as it is shown in Listing 23.

Listing 23: V&V Engine jobs

```
jobs:
  - file: /var/jenkins_home/seeds.groovy
```

The Validation & Verification Engine is containerized, so as to be deployed in the Kubernetes cluster. This is achieved by using a Docker file, which contains all the necessary information for the creation of the docker image. Furthermore, in the aforementioned file the needed commands for the installation of the Apache JMeter component, the Robot framework and the necessary plugins are defined. The configuration of the Docker file is demonstrated in Listing 24.

Listing 24: V&V Engine Docker file

```
FROM jenkins/jenkins:2.332.3-jdk11
USER root
ARG TZ="Europe/Amsterdam"
ENV TZ ${TZ}
ENV JAVA_OPTS -Djenkins.install.runSetupWizard=false
ENV JENKINS_OPTS --httpPort=8083 --prefix=/vv-engine
ARG DOCKERVERSION=20.10.1
ARG JMETER_VERSION="5.4"
ENV JMETER_HOME /opt/apache-jmeter-${JMETER_VERSION}
ENV       JMETER_BIN       ${JMETER_HOME}/bin
ENV       JMETER_DOWNLOAD_URL                              https://archive.apache.org/dist/jmeter/binaries/apache-jmeter-
${JMETER_VERSION}.tgz
ENV CASC_JENKINS_CONFIG /var/jenkins_home/casc.yaml
ENV GITLAB ***
ENV KUBERNETES_ADDR 21.10.200.2
ENV NETAPP_REPO_URL https://gitlab.netapps-5gmediahub.eu
RUN curl -fsSLO https://download.docker.com/linux/static/stable/x86_64/docker-${DOCKERVERSION}.tgz \
  && tar xzvf docker-${DOCKERVERSION}.tgz --strip 1 \
             -C /usr/local/bin docker/docker \
  && rm docker-${DOCKERVERSION}.tgz

COPY plugins.txt /usr/share/jenkins/ref/plugins.txt
COPY casc.yaml /usr/share/jenkins/ref/casc.yaml
COPY seeds.groovy /usr/share/jenkins/ref/seeds.groovy
COPY kubeconfig.yaml /usr/share/jenkins/ref/kubeconfig.yaml

RUN cp /usr/share/jenkins/ref/casc.yaml /var/jenkins_home/casc.yaml
RUN cp /usr/share/jenkins/ref/seeds.groovy /var/jenkins_home/seeds.groovy
RUN cp /usr/share/jenkins/ref/kubeconfig.yaml /var/jenkins_home/kubeconfig.yaml
RUN /usr/local/bin/install-plugins.sh  < /usr/share/jenkins/ref/plugins.txt
RUN    apt update \
            && apt upgrade -y \
            && apt-get install ca-certificates -y \
            && update-ca-certificates \
            && mkdir -p /tmp/dependencies  \
            && apt-get install -y libcurl3-nss \
            && curl -L --silent ${JMETER_DOWNLOAD_URL} >  /tmp/dependencies/apache-jmeter-${JMETER_VERSION}.tgz  \
            && mkdir -p /opt  \
            && tar -xzf /tmp/dependencies/apache-jmeter-${JMETER_VERSION}.tgz -C /opt  \
            && rm -rf /tmp/dependencies
RUN apt-get update && apt-get install -y python3.6 python3-distutils python3-pip python3-apt
RUN pip install robotframework
RUN pip install requests
# Set global PATH such that "jmeter" command is found
ENV PATH $PATH:$JMETER_BIN
USER jenkins
```

3.1.5.3 Deployment Architecture

A Docker image of the Validation & Verification Engine is uploaded on the master node of the cluster and stored on the registry of the docker engine of the cluster. The deployment on the Kubernetes cluster is performed via YAML files. These files contain all the necessary configuration regarding the creation of the necessary pod, user, service, ingress in a dedicated namespace on the cluster. The deployment procedure of the engine is described below.

- **Deployment Setup: Namespace**

At first, for the needs of the engine, the following Kubernetes configuration should be performed once. All the necessary components of the engine are deployed under a dedicated namespace on the Kubernetes cluster. This namespace is called **vv-engine** and it can be created by using the command in Listing 25.

Listing 25: V&V Engine Kubernetes namespace creation

```
kubectl create namespace vv-engine
```

- **Deployment Setup: Service Account**

A service account is created for the namespace mentioned above having admin rights, in order for this account to handle all the necessary tasks regarding the components of the engine. A binding of this service account with a cluster role is performed. The name of the service account is **vv-engine-admin**, and the relevant Kubernetes configuration is shown in Listing 26.

Listing 26: V&V Engine Kubernetes service account creation

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: vv-engine-admin
  namespace: vv-engine
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["*"]

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: vv-engine-admin
  namespace: vv-engine

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: vv-engine-admin
  namespace: vv-engine
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: vv-engine-admin
subjects:
```

```
- kind: ServiceAccount
  name: vv-engine-admin
  namespace: vv-engine
```

- **Deployment Setup: Persistent Storage**

The Validation & Verification engine needs a persistent storage so as to store the state of the executed pipelines, along with their corresponding results. This is achieved by using a persistent volume claim, named **pv-mediahub**, which is provided by the infrastructure.

- **Deployment of the Engine**

When the above steps are finished then the below procedure can take place each time an updated version of the engine is needed to be deployed.

1. A docker image of the engine, which is preconfigured as mentioned on the subsection 3.1.5.2 is uploaded on the master node of the cluster and stored on the image registry of the docker engine. The name of the image is **vv-engine**.
2. A deployment of the engine is executed, by applying the necessary configuration stored on a YAML file. In this file, as it can be seen in Listing 27, the service account which handles the engine is defined, as well as the deployment using the image of the engine with the relevant configuration, such as the needed resources and the persistent volume claim. The name of the deployment is **vv-engine**. In the YAML file mentioned above, the instructions of a dedicated Kubernetes Service are included. This Service allows the application to be exposed as a network service.

Listing 27: V&V Engine Kubernetes create Deployment and Service

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vv-engine
  namespace: vv-engine
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vv-engine
  template:
    metadata:
      labels:
        app: vv-engine
    spec:
      securityContext:
        fsGroup: 1000
        runAsUser: 1000
      serviceAccountName: vv-engine-admin
      containers:
        - name: vv-engine
          image: vv-engine:0.0.1
          resources:
            limits:
              memory: "2Gi"
              cpu: "500m"
            requests:
              memory: "500Mi"
```

```
          cpu: "100m"
        ports:
         - name: httpport
           containerPort: 8083
         - name: jnlpport
           containerPort: 50000
        env:
         - name: JENKINS_ADMIN_PASSWORD
           value: "***"
         - name: JENKINS_ADMIN_ID
           value: "***"
        livenessProbe:
         httpGet:
           path: "/vv-engine/login"
           port: 8083
         initialDelaySeconds: 180
         periodSeconds: 10
         timeoutSeconds: 5
         failureThreshold: 5
        readinessProbe:
         httpGet:
           path: "/vv-engine/login"
           port: 8083
         initialDelaySeconds: 180
         periodSeconds: 10
         timeoutSeconds: 5
         failureThreshold: 3
        volumeMounts:
         - name: jenkins-data
           mountPath: /var/jenkins_home
    volumes:
     - name: jenkins-data
       persistentVolumeClaim:
          claimName: pv-mediahub

---
apiVersion: v1
kind: Service
metadata:
 name: vv-engine-service
 namespace: vv-engine
spec:
 selector:
  app: vv-engine
 ports:
  - port: 80
    targetPort: 8083
```

In order for the application to expose HTTP and HTTPS routes from outside the cluster to the service of the engine within the cluster an Ingress configuration takes place as seen in Listing 28. This configuration allows the Ingress controller, provided by the infrastructure, to route the requests towards a specific path, to the Ingress resource of the engine.

Listing 28: V&V Engine Kubernetes create Ingress resource

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: vv-engine
  namespace: vv-engine
spec:
  ingressClassName: nginx
  rules:
   - host: services.5gmediahub.eu
     http:
      paths:
      - pathType: Prefix
        path: "/vv-engine"
        backend:
          service:
            name: vv-engine-service
            port:
              number: 80
```

3.1.5.4 Hardware Resources

Table 9, specifies the computational resources required within the Kubernetes clusters in order for the Validation & Verification engine to be fully functional.

Table 9: V&V Engine Kubernetes Compute Resources

RAM	Storage space	Network requirements
4GB	20GB	Internet Access

3.1.6 Continuous QoS/QoE Monitoring Engine Module

The Continuous Quality of Service & Quality of Experience Engine (QoS/E Engine) is comprised of several sub-components each of which fulfils a specific functionality as required by the overall engine. The sub-components are:

- NetApps Monitoring Manager
- Vertical Application Monitoring Manager
- QoE Prediction Manager
- QoS/E database instance

All of the data for the QoS/E Engine is supplied by the data collector module of the experimenters' portal and is ingested into the QoS/E for processing.

3.1.6.1 Integration Endpoints

Table 10: QoSE Integration Matrix

Source System / IP	Target System / IP	Protocol / Port	Functionality
QoS/E Engine	*Experimenters Portal -> Data Collector Module*	*HTTP REST APIs / 80/443*	*Issues requests to the data collector to retrieve all of the raw input data for processing.*
QoS/E Engine	*UMM*	*HTTP REST APIs / 80/443*	*The QoS/E Engine module calls the UMM for authentication/authorization*

3.1.6.2 Configuration

The configuration of each sub-component will be within the design of its individual Docker file. As the QoS/E Engine is still in the early stages of design the applications for each of the sub-components have yet to be designed and developed hence the Docker files for each component hasn't been developed as of the time of writing.

3.1.6.3 Deployment Architecture

Similar to the deployment process employed for both the NetApps Repository application and the User Management Module, the QoS/E Engine has all of its sources hosted on a specific GitLab instance and the applications' artefacts are built using the continuous integration pipelines once the code has been updated in the GitLab repository. The output from the CI build pipelines is the docker containers that are stored in the container registry, also hosted by the GitLab instance. The overall design is that the individual components will execute within a microservices architecture where all the individual modules will be represented by an individual container executing within the Kubernetes cluster deployment infrastructure. The namespace has already been configured and the resource can be seen in Listing 29.

Listing 29: Namespace resource for the QoS/E Engine

```
apiVersion: v1
kind: Namespace
metadata:
  name: qose-engine-wit
```

The remaining resources required for the deployment of the QoS/E Engine components will be implemented once the applications' designs have been finalised and they will form part of the overall application development process.

3.1.6.4 Hardware Resources

The QoS/E Engine is a completely cloud based deployment where the majority of hardware requirements is based around persistent storage (for the database data), inter cluster network access (for raw data ingress from the data collector) and internet access (for the reporting and control APIs). The specific hardware resources per sub-component is captured in Table 11.

Table 11: QoS/E Engine Deployment Hardware Requirements Overview

(Sub)Module	RAM	Storage space	Network requirements

NetApps Monitoring Manager	500MB	N/A	QoS/E Engine Namespace access
Vertical Monitoring Manager	500MB	N/A	QoS/E Engine Namespace access
QoE Prediction and Control Manager	1GB	N/A	QoS/E Engine Namespace access
QoS/E Database Instance	500MB	10GB	Private Repository with internal network access within the QoS/E Engine namespace and access to the cluster Persistent Storage
QoS/E Engine API Gateway	500MB	N/A	Internet access required over FQDN

4 Experimentation Facility Evaluation Methodology

The purpose of the current section is to describe the testing framework for verifying the functionality of the components against technical specifications and requirements. The testing results will provide feedback to the technical Work Packages (WPs) in the form of reports on defects and required adjustments. When deviations in components' behaviour are identified, requirements that have been defined under WP1 will be revisited during the second round of requirements definition. This process will contribute to the updated version of the requirements deliverable (i.e., D1.4 "Functional requirements elicitation and analysis-Final", due in M24).

In the sub-sections that follow, the definition of the testing framework is provided and what tests developers should carry out are described. The ultimate aim of testing is the early fault detection, before the platform grows in complexity, so as to resolve any issues that may arise more efficiently.

4.1 5GMediaHUB testing lifecycle

The testing lifecycle that will be applied is shown in Figure 3, and follows an iterative phased process.

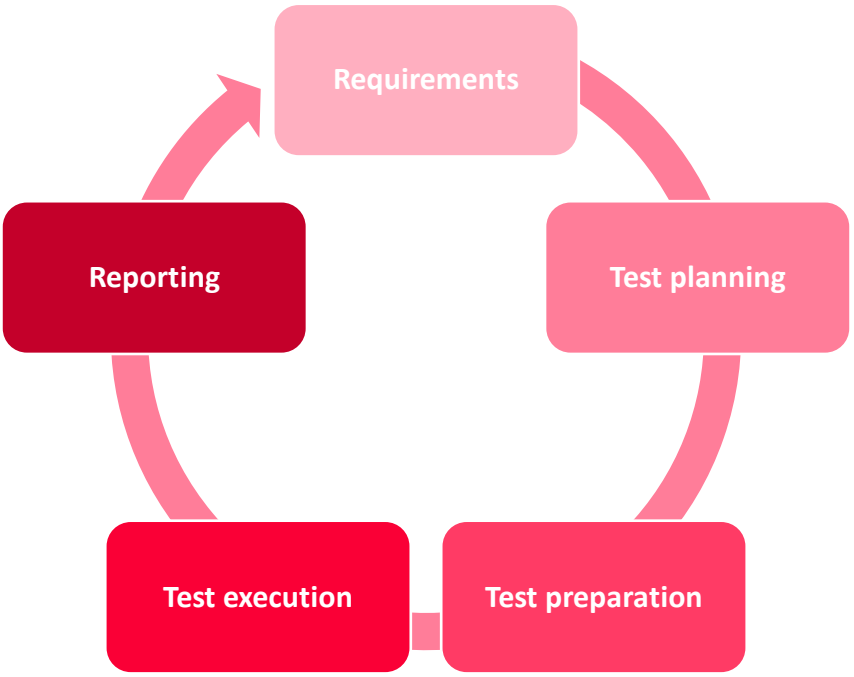

Figure 3: 5GMediaHUB testing lifecycle

The first stage of the testing lifecycle involves **planning**. During this stage, the testing team assembles and decides on the testing strategy that must be followed. The purpose of this phase is to revisit the identified requirements to determine which components and what features in these components will be tested. At the end of this phase a plan should be devised, providing a detailed description of unit and integration testing so as to be understandable and easy to apply. This plan will be further elaborated and updated to include the following information: people who will participate in the testing, the tools needed for the testing and the proper test environment.

The second phase of the lifecycle is **tests preparation**. Here, the partners will specify test scenarios and test cases for their components, which will provide an overview of what will be tested, how the components should behave in real life and what are the steps to be followed to validate that the component delivers on its specifications without major defects. During this phase, the identification and collection of test data needed to run the tests will be explored.

The final two phases of the lifecycle include the **execution** of the test cases and the **reporting** of the results to the other WPs. In this regard, the test results will be compared with the expected ones to determine whether a test case has passed or failed the criteria set by the owning partner. Bugs will be logged into the GitLab bug/issue tracking tool to be fixed in the task where the component is developed. Reporting will consist of an analysis of the test cases and discovered defects and will be forwarded to the appropriate partner or WP to restart the cycle.

4.2 Automated and manual testing

To ensure that the developed software operates as expected and is of utmost quality, performing tests is of essence. The purpose of testing is to isolate and identify defects before the software is available on the market and, as a result, improve its quality and ensure high performance. Testing can be done in two ways: automatically or manually.

Due to the alignment of the 5GMediaHUB integration plan with CI/CD and development and operations (DevOps) methodologies, introducing automation in testing is vital to avoid bottlenecks and ensure integration targets are met without major issues. Apart from the time saved when conducting tests in an automated manner, automated tests are particularly useful to ensure that the software updates do not retroactively introduce problems in the existing code (regression testing), therefore ensuring a much faster delivery cycle compared to manual testing. Hence, 5GMediaHUB will align its testing lifecycle approach to have as many automated tests as possible, and consequently to minimise the human factor and reduce the overall test effort and time consumed.

The final chosen strategy for testing within 5GMediaHUB is based on automated tests within GitLab (unit and integration tests), while system and acceptance tests will involve human operators.

4.2.1 Automated tests

Two types of automated tests can be performed within 5GMediaHUB:

1) **Unit tests:** They are carried out to verify the implementation of a function or class inside a software module.
2) **Integration tests:** They are performed to scrutinise the interface integrity.

In 5GMediaHUB automated tests will run completely in the GitLab continuous integration / continuous delivery (CI/CD) environment. As mentioned in D1.5 “DevOps implementation and testing Methodology and benchmarking of results – Initial”, “the GitLab CI pipeline is part of the 5GMediaHUB testing framework and includes all required unit tests and integration tests”. GitLab allows a software module to be installed automatically and a Docker container to be built. Also, with GitLab unit and integration tests can be performed by means of CI/CD pipelines. To enable automated testing integration into a pipeline, a test stage should be defined in the YAML CI/CD configuration located in the root of the respective repository.

4.2.1.1 Unit tests

Unit tests help developers test if individual components are working as they are expected to. As part of a DevOps pipeline, unit tests are executed frequently for the purposes of quality assurance. For this reason, unit tests should be executed automatically. Unit tests are executed via a program, usually in the form of a test script written in the same programming language as the unit to be tested. For instance, in a typical object-oriented programming scenario, each class should be coupled to a test class intended to test the public methods of the respective class.

Unit tests should be carried out when there are changes to code. As shown in Figure 3 they should be written prior to the implementation phase to make sure that any potential issues do not arise during the production phase.

4.2.1.2 Integration tests

The purpose of integration tests is to ensure the communication between components. In integration tests whether two units or modules interface properly is verified. Integration tests are executed after unit tests. If a component is composed of two or more sub-components, then whether these sub-components are compatible with each other should also be checked before proceeding to the integration tests of other independent components.

The definition of integration tests should take place during the test preparation phase of the 5GMediaHUB testing lifecycle, since how components will communicate with each other will have been determined. To this end, a prespecified set of rules described in the test case scenarios will be used for the cross-examination of application programming interfaces (APIs) exposed by components. Integration tests can thus be viewed as a way to automatically verify basic operations of 5GMediaHUB sub-systems, i.e., autonomous interconnected groups of components within the 5GMediaHUB architecture.

4.2.2 Manual tests

Manual tests can also be carried out. Although automated tests are preferable because they take less time, one main advantage of manual tests is that they cost less than automated tests. Manual tests are mainly implemented in the cases of system and acceptance testing.

By **system testing** we refer to the process of verifying that the system as a whole operates properly and that it meets customers' expectations. Integrators are responsible for this type of testing. Due to the fact that system tests will be executed manually, they should be planned during the test planning stage, where the need to prepare system demonstrators and test data is identified. Test cases will be drafted during test preparation in which a protocol will be defined for the human tester to assert performance against functional and non-functional requirements.

Acceptance tests are carried out by end-users to check if the developed system meets the goals as defined in the business requirements. The adoption of the software by target stakeholders is determined by the level of success of acceptance tests. During the development phase, 5GMediaHUB partners involved in the development of a particular component will as act as end-users themselves and run use case trials.

5 Conclusions

This deliverable was the initial attempt to document all the integration points and the configurations needed for each one of the six experimentation facility modules. It has described the work done so far while integrating and on boarding the components onto the Kubernetes cluster provided within the CTTC infrastructure which hosts the experimentation facility. At this point of the project, the Validation & Verification Engine has proceeded with an initial deployment on the cluster while the other modules are currently deployed on each partners' staging environment for testing and integration purposes. As we move forward with the cycle 1 trials, these will be onboarded the Kubernetes infrastructure and their integration details will be updated in the next drop of this deliverable (D3.4).

In the coming months, while moving on with testing cycle 1, the experimentation facility modules will be fully onboarded the Kubernetes cluster and the evaluation plan described in section 4 will be carried out as part of T3.5. At this point, the trials can start using the experimentation facility and within this task, the relative partners will resolve potential issues and evaluate improvements that can be applied for the cycle 2 trials.

6 References

[1] 5GMediaHUB Deliverable D1.7 – Architecture Design & Technical Specifications – Initial. [Online]. Available: https://www.5gmediahub.eu/wp-content/uploads/2021/09/5GMediaHUB-D-1.7-Architecture-design-and-technical-specifications_20210830_Final.pdf

[2] 5GMediaHUB Deliverable D1.3 - Functional requirements elicitation and analysis – Initial. [Online]. Available: https://www.5gmediahub.eu/wp-content/uploads/2021/08/5GMediaHUB-D1.3-Functional-requirements-elicitation-and-analysis-_20210731_Final.pdf

[3] 5GMediaHUB Deliverable D2.5 – Test Planning & Definition Engine development – Initial. [Online]. Available: https://www.5gmediahub.eu/wp-content/uploads/2022/04/5GMediaHUB-D2.5_submitted.pdf

Annex I – Kubernetes Resource Scripts for the 5GMediaHub Components

Listing 30: NetApps Repository Backend Deployment Resource

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: netapps-repo-backend-5gmedhub
  namespace: netapps-repo-app-wit
spec:
  replicas: 1
  selector:
    matchLabels:
      app: netapps-repo-backend-5gmedhub
  template:
    metadata:
      labels:
        app: netapps-repo-backend-5gmedhub
    spec:
      securityContext:
          fsGroup: 1000
          runAsUser: 1000
      imagePullSecrets:
      - name: wit5gmedhubglregcred
      serviceAccountName: netapps-repo-app-wit-admin
      containers:
      - name: netapps-repo-backend-5gmedhub
        image:              gitlab.netapps-5gmediahub.eu:5050/5gmediahub-platform/netapps-
repository/net-apps-portal-backend:latest
        ports:
        - name: httpport
          containerPort: 80
```

Listing 31: NetApps Repository Backend Service Resource

```
apiVersion: v1
kind: Service
metadata:
  name: netapps-repo-backend-5gmedhub-service
  namespace: netapps-repo-app-wit
  labels:
    app: netapps-repo-backend-5gmedhub
spec:
  selector:
    app: netapps-repo-backend-5gmedhub
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 80
```

Listing 32: NetApps Repository Backend Ingress Resource

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: netapps-repo-backend-5gmedhub
  namespace: netapps-repo-app-wit
  #annotations:
  #    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
          #- host: services.5gmediahub.eu
   - http:
      paths:
      - pathType: Prefix
        path: "/netappsrepobe"
        backend:
          service:
            name: netapps-repo-backend-5gmedhub-service
            port:
              number: 80
```

Listing 33: UMM Portal Backend Service Deployment Resource

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: umm-backend-5gmedhub
  namespace: umm-wit
spec:
  replicas: 1
  selector:
    matchLabels:
      app: umm-backend-5gmedhub
  template:
    metadata:
      labels:
        app: umm-backend-5gmedhub
    spec:
      securityContext:
          fsGroup: 1000
          runAsUser: 1000
      imagePullSecrets:
      - name: wit5gmedhubglregcred
      serviceAccountName: umm-wit-admin
      containers:
      - name: umm-backend-5gmedhub
        image:      gitlab.netapps-5gmediahub.eu:5050/5gmediahub-platform/user-management-
module/umm-backend:latest
        ports:
        - name: httpport
          containerPort: 80
```

Listing 34: UMM Portal Backend Service - Service Resource

```
apiVersion: v1
kind: Service
metadata:
  name: umm-backend-5gmedhub-service
  namespace: umm-wit
  labels:
    app: umm-backend-5gmedhub
spec:
  selector:
    app: umm-backend-5gmedhub
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 80
```

Listing 35: UMM Portal Backend Service Ingress Resource

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: umm-backend-5gmedhub
  namespace: umm-wit
  #annotations:
  #    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
          #- host: services.5gmediahub.eu
   - http:
      paths:
      - pathType: Prefix
        path: "/ummportalapi"
        backend:
          service:
            name: umm-backend-5gmedhub-service
            port:
              number: 80
```