
D2.4.2

Version	2.5
Author	URJC
Dissemination	PU
Date	25/01/2016
Status	Final



D2.4.2: NUBOMEDIA Architecture v2

Project acronym:	NUBOMEDIA
Project title:	NUBOMEDIA: an elastic Platform as a Service (PaaS) cloud for interactive social multimedia
Project duration:	2014-02-01 to 2017-02-01
Project type:	STREP
Project reference:	610576
Project web page:	http://www.nubomedia.eu
Work package	WP2: User centric analysis of user scenarios and requirements
WP leader	VTTOOLS
Deliverable nature:	Report
Lead editor:	Luis López Fernández
Planned delivery date	31/01/2016
Actual delivery date	25/01/2016
Keywords	Architecture

The research leading to these results has been funded by the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 610576



FP7 ICT-2013.1.6. Connected and Social Media



This is a public deliverable that is provided to the community under a **Creative Commons Attribution-ShareAlike 4.0 International** License

<http://creativecommons.org/licenses/by-sa/4.0/>

You are free to:

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material
for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices:

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

For a full description of the license legal terms, please refer to:

<http://creativecommons.org/licenses/by-sa/4.0/legalcode>

Contributors:

Luis López Fernández (URJC)
Giuseppe Carella (TUB)
Lorenzo Tomasini (TUB)
Javier López Fernández (NAEVATEC)
Alin Calinciuc (USV)
Alice Cheambe (Fhg FOKUS)

Internal Reviewer(s):

Giuseppe Carella (TUB)

Version History

Version	Date	Authors	Comments
0.1	28/07/14	Luis López Fernández	First initial version
0.2	8/10/14	Luis López	Cloud architecture as functional architecture.
0.3	3/01/15	Luis Lopez	Mayor modifications to architecture in order to simplify it and to adapt to a more flat model where all the logic of distributed media pipeline management is left to applications.
1.0	19/01/15	Luis Lopez	Added CFD descriptions for TUB. Added new reference points descriptions.
1.1	30/06/2015	Alice Cheambe	Updated DSFE
1.2	13/07/2015	Giuseppe Carella	Restructuring the document considering the comments received by the reviewers after year 1 review
2.0	20/09/2015	Luis Lopez	Complete factorization of document for adapting to reviewer requirements.
2.1	2/01/2016	Giuseppe Carella	Contributions for NFVO and VNFM
2.2	2/01/2016	Alice Cheambe	Contributions for PaaS Manager and PaaS
2.3	14/01/2016	Giuseppe Carella	Added NVF basic introduction. Added EMM description.
2.4	14/01/2016	Alin Calinciuc	New version of the NUBOMEDIA IaaS.
2.5	18/01/2016	Alice Cheambe & Giuseppe Carella	PaaS manager main interactions. Reviewed EMM FMC architecture.

Table of contents

1	Executive summary.....	9
2	Terminology and format conventions	9
3	Introduction.....	9
4	NUBOMEDIA Architecture.....	10
4.1	Introduction.....	10
4.2	Architecture overview.....	11
4.3	NUBOMEDIA modules	13
4.3.1	<i>The PaaS Manager.....</i>	<i>13</i>
4.3.2	<i>NUBOMEDIA PaaS.....</i>	<i>16</i>
4.3.3	<i>Network Function Virtualization Orchestrator.....</i>	<i>18</i>
4.3.4	<i>Connectivity Manager</i>	<i>21</i>
4.3.5	<i>Virtual Network Function Manager.....</i>	<i>21</i>
4.3.6	<i>Virtual Infrastructure Manager.....</i>	<i>24</i>
4.3.7	<i>NUBOMEDIA Media Plane.....</i>	<i>25</i>
4.3.8	<i>NUBOMEDIA IaaS.....</i>	<i>36</i>
5	NUBOMEDIA Development APIs	39
5.1.1	<i>Media Capabilities APIs</i>	<i>40</i>
5.1.2	<i>Signaling APIs.....</i>	<i>40</i>
5.1.3	<i>Abstract Communication APIs.....</i>	<i>40</i>
	References.....	42

List of Figures:

<i>Figure 1. The NUBOMEDIA development model (right) is based on the popular tree tier development model of WWW applications (left), so that application developers create their programming logic consuming different service APIs: DD.BB. APIs in the case of WWW and media-oriented APIs in the case of NUBOMEDIA.</i>	10
<i>Figure 2. This figure shows the NUBOMEDIA system architecture following the Fundamental Modeling Concepts (FMC) block diagram visual notation. The NUBOMEDIA architecture is based on the ETSI NFV Management and Orchestration (MANO) recommendation but extends it with a PaaS layer. Thanks to it, when developers deploy their applications through a PaaS API, the PaaS Manager orchestrates all the required actions for the provisioning of the appropriate resources and services required for applications to run in a reliable and scalable way.</i>	12
<i>Figure 3. PaaS Manager internal architecture.</i>	14
<i>Figure 4. PaaS Manager Interactions</i>	15
<i>Figure 5. The NUBOMEDIA PaaS architecture.</i>	17
<i>Figure 6. Relationship between Docker containers, images, and registries.</i>	18
<i>Figure 7. ETSI NFV MANO Architecture.</i>	19
<i>Figure 8. NFVO internal architecture.</i>	20
<i>Figure 9. Following ETSI MANO architecture, the MS-VNFM and the CR-VNFM are managed by the NFVO through a control bus.</i>	22
<i>Figure 10. MS-VNFM Architecture extended the basic VNFM with the EMM.</i>	23
<i>Figure 11. Media Server (MS) component internal architecture. A MS comprises a number of Media Pipelines (MPs). MPs are graphs of interconnected Media Elements (MEs). Each ME implements a specific media capability: media transport, media archiving or media processing. MS pipelines and elements are controlled by the Media Server Manager, which communicated with the external world through the Media Control interface (iMC) and the Media Events interface (iME).</i>	25
<i>Figure 12. Media Element (ME) component internal architecture. A ME has a sink, where media may be received and a source, where media can be send. Sinks and Sources exchange media with other MEs through the Media Internal interface (iMI). ME specific capabilities are encapsulated behind an interface that it made available to the rest of MS components through the Media Control Internal interface (iMCI).</i>	26
<i>Figure 13. Endpoint MEs (left) are MEs with the capability of exchanging media with he external world through the iM interface. In an Endpoint, media received through its Sink is forwarded to an external entity (e.g. the network). Media received from the external entity is published into the Endpoint Source. Typically, Endpoint MEs publish events associated with the transport status (e.g. media-connection-established, etc.) Filter MEs (right) do not implement the iM interface and cannot exchange media with the external world. Hence, Filter MEs can only be used for processing the media received through their Sinks, which is then published to their Sources. Filter MEs typically publish events associated to the specific media processing being performed (e.g. face-detected in a FaceDetector filter).</i>	26
<i>Figure 14. Media Pipelines (PMs) are graphs or interconnected Media Elements (MEs) that implement a specific media processing for an application. The Media Pipeline Manager controls the lifecycle and behavior of its MEs by brokering control messages and events among the MEs and the Media Server components our of the MP.</i>	27
<i>Figure 15: The Media Server Manager (MSM) holds the appropriate logic for performing two functions. The first is to enable communications with applications through the iMC control interface and the iME event publishing interface. The second is to provide the appropriate logic for dispatching, managing and providing semantics to control messages and events, including the ones in charge of media capabilities lifecycle management (e.g. create, release, etc.).</i>	28
<i>Figure 16. Flow diagram showing the interactions among the Media Server Manager modules for creating a new media capability (i.e. Media Element or Media Pipeline). The specific media capability type is identified by the string MCid (Media_Capability_id), that needs to be registered among modules</i>	

capabilities into the Module Manager. Once the capability has been instantiated, it receives a unique id (mid or Media_id), which identifies it during its whole lifecycle.....30

Figure 17. Flow diagram showing the interactions among the Media Server Manager modules for invoking a method (identified as M) into a specific Media Element (identified as mid)31

Figure 18. This diagram shows the typical entities and interactions involved in a media application based on a Media Server. In the case of NUBOMEDIA, the Application Server corresponds with the NUBOMEDIA PaaS component and, more specifically, with the container where the specific application logic is executing.32

Figure 19. Flow diagram schematizing the different interactions taking place for establishing a WebRTC session between the Client Application and the Media Server.....34

Figure 20. Cloud Repository architecture35

Figure 21. Main interactions for recording an item into the Cloud Media Repository.35

Figure 22. Main interactions for playing an item from the Cloud Media Repository.....36

Figure 23 Functional architecture of NUBOMEDIA IaaS37

Figure 24. Architectural diagram showing the NUBOMEDIA API stack which comprises three types of APIs: Media Capability APIs (NUBOMEDIA Media and Repository APIs), Signaling APIs (JSON-RPC over WebSocket client and server APIs), and Abstract Communication APIs (Room and Tree client and server APIs). In this picture, the NUBOMEDIA Media Protocol corresponds to the iMC and iME interfaces described in Figure 11, while the NUBOMEDIA Signaling Protocols correspond to the iS interface described in Figure 2.39

Acronyms and abbreviations:

API	Application Programming Interface
AS	Application Server (refers to function instance)
ASS	Application Server Services
CFM	Cloud Functions Manager
CPU	Central Processing Unit
DAS	Distributed Application Server (refers to function class)
DMP	Distributed Media Pipeline
DMS	Distributed Media Server (refers to function class)
DoD	Denial of Service
DSFE	Distributed Signaling Front End (refers to function class)
IaaS	Infrastructure as a Service
JSON	JavaScript Object Notation
ME	Media Element
MP	Media Pipeline
MS	Media Server (refers to function instance)
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
NFVO	NFV Orchestrator
NS	Network Service
NSD	Network Service Descriptor
NSR	Network Service Record
PA	Platform Application
PaaS	Platform as a Service
PAL	Platform Application Logic
PAMC	Platform Application Media Control
PNF	Physical Network Function
RFC	Request For Comments
RTC	Real-Time Communications
SaaS	Software as a Service
SFE	Signaling Front End (refers to function instance)
UE	User Equipment
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNFM	VNF Manager
WS	WebSocket
WWW	World Wide Web
XMPP	eXtensive Message and Presence Protocol

1 Executive summary

This document contains a specification of the NUBOMEDIA architecture. This specification provides the basis on for driving the technological developments in WP3, WP4 and WP5.

2 Terminology and format conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

All architectural diagrams in this document have been created following the conventions of the Fundamental Modeling Concepts [FMC] framework.

Following FMC conventions, active components are depicted as named squared-corner boxes and passive components as named rounded-corner boxes. Components names comprise one or several upper-case terms. Interfaces are represented through named or un-named circles. For named interfaces, we accept the convention of them starting with a lower-case "i" character.

3 Introduction

The NUBOMEDIA Platform is an elastic cloud PaaS that has been created in the context of the NUBOMEDIA Project to comply with the following objectives:

- To provide elastic scalability for media delivery, so that physical and virtual computing and networking resources are allocated on demand to them accordingly to their QoS requirements.
- To provide a modular media plane suitable for hosting interacting media capabilities including media transport, media (de)coding, media filtering, media interoperability and media processing.
- To provide advanced autoscaling mechanisms able to orchestrate complex stateful media components.
- To provide a set of APIs suitable for complying with the requirements of a wide spectrum of developers by enabling, at the same time, abstraction and flexibility.

The objective of this document is to present an architecture for the NUBOMEDIA platform concentrating on understandability more than on exhaustivity or completeness. To this aim, the architecture represents NUBOMEDIA as a system (i.e. the dynamical system that emerges from the execution of the NUBOMEDIA software artifacts) and not NUBOMEDIA as a software (i.e. a number of source code artifacts). As a result, this system architecture is understood as a high level abstraction showing the different components comprising the platform and their interactions.

Due to its simplicity and expressiveness, the architectural diagrams presented in this document are based on the Fundamental Modeling Concepts [FMC] block diagram visual notation. This notation is convenient for representing the compositional structure of software systems through three types of graphical elements: agents, storages and channels. Agents are active systems provided of some kind of logic, which have access to adjacent passive systems. Example of agents are active software components or human users. Agents are presented through squared boxes. Storages are used by agents to store and recover data. Agents are represented through circled shapes. Channels, in

turn, represent communications among agents. Channels are represented through small circles qualifying an access type, represented through directed or undirected lines.

4 NUBOMEDIA Architecture

4.1 Introduction

The NUBOMEDIA Platform has been conceived for simplifying the way developers use to deal with RTC interactive media applications. Its aim is to provide simple mechanisms and APIs for creating such applications. For this, NUBOMEDIA tries to abstract all the low level details of service deployment, management, and exploitation allowing applications to transparently scale and adapt to the required load while preserving QoS guarantees.

For enhancing understandability and simplicity, the NUBOMEDIA Platform has been designed in such a way that, from the developer's perspective, its capabilities are accessed through APIs inspired by the popular WWW three tier model, as shown in Figure 1.

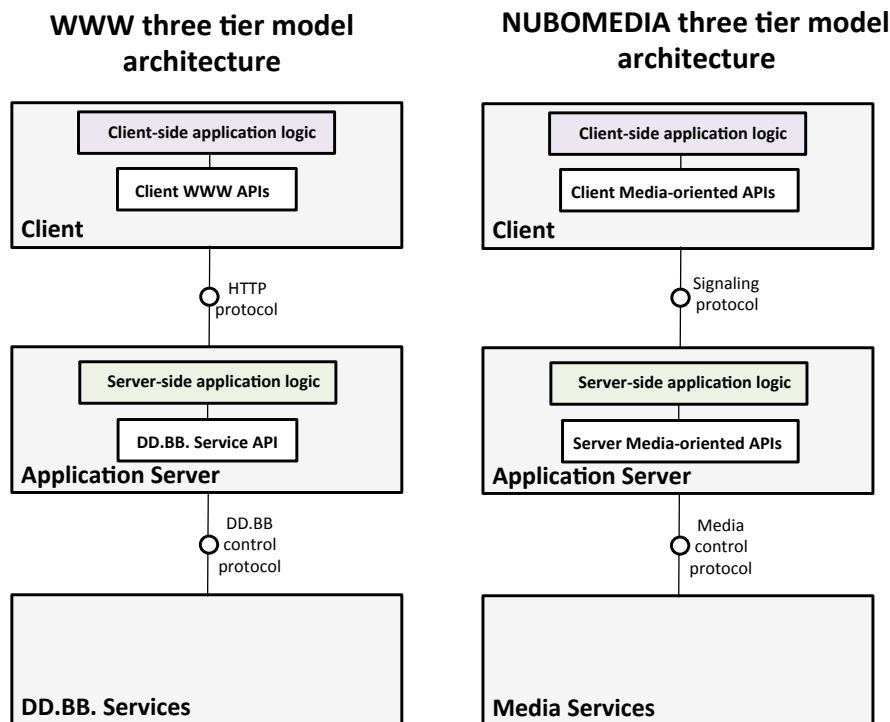


Figure 1. The NUBOMEDIA development model (right) is based on the popular tree tier development model of WWW applications (left), so that application developers create their programming logic consuming different service APIs: DD.BB. APIs in the case of WWW and media-oriented APIs in the case of NUBOMEDIA.

In the same way that WWW developers program their application logic consuming Data Base (DD.BB.) APIs, NUBOMEDIA makes it possible to create rich media services through media-oriented APIs. These APIs are based on the concept of Media Pipeline: a chain of Media Elements, where a Media Element is an abstraction holding a specific media capability and hiding its complexities behind a comprehensive interface. The parallelism is straightforward and understandable for developers:

- In the WWW development model, at the very top we have the client: a web browser. As browsers are thin client, they expose development APIs suitable for controlling the presentation and user interaction capabilities of the application. In NUBOMEDIA, the client is again understood as thin. It exposes media-

oriented APIs enabling media capture, media rendering and media transport toward the infrastructure.

- Below the client, we have the Application Server layer, which interacts at its northbound interface with the client through some kind of control protocol that, in the case of WWW applications is typically the HTTP protocol, and in the case of NUBOMEDIA is a RTC signaling protocol. This layer holds the business logic of the application, which translates the client requests into the appropriate orchestrated commands toward the service layer.
- At the very bottom of the architecture, the service layer holds data services, in the case of WWW applications, and RTC media services, in the case of NUBOMEDIA.

4.2 Architecture overview

The NUBOMEDIA Platform is a complex piece of software that requires a non-trivial architecture providing many different functions, capabilities and interfaces. For the sake of clarity and brevity, in this document we only identify its main parts and we do not dig deeply into its complex details.

As mentioned previously, a NUBOMEDIA application is composed by three parts: the client-side application logic (executed at the client browser), the server-side application logic (executed inside an Application Server), and the RTC media services (usually Network Functions hosted in service provider's datacenters). In order to orchestrate and expose such services via API, it was employed the same multi-layered architecture.

Starting from the bottom, the first important innovation is represented by the RTC media services cloudification. This is achieved introducing and extending NFV concepts in the NUBOMEDIA Architecture. The RTC media services layer scales accordingly to the application needs.

Continuing, the server-side application logic exposes advanced APIs for building a modular media plane consuming the media capabilities provided by the lower layers. This application can be deployed on the NUBOMEDIA PaaS using the provided NUBOMEDIA PaaS API. The innovation introduced by this approach is that the NUBOMEDIA Platform will instantiate a set of RTC media services which are dedicated to the application, and which the application can dynamically scale via the advanced APIs exposed by the media service layer.

Nubomedia architecture overview

The NUBOMEDIA high-level architecture is depicted in Figure 2. This architecture complies with the ETSI NFV MANO specification [MANO]. Details about each of the NUBOMEDIA components are provided in sections below. At a very high-level perspective, and following top down approach, these are the core functions:

- NUBOMEDIA PaaS being the intermediate level between the infrastructural resources and the users of the platform. Particularly it includes the NUBOMEDIA PaaS Manager exposing the iD interface to the developers for deploying their applications. Due to this, this interface is also called the PaaS Manager API along this document. This level also holds the NUBOMEDIA PaaS hosting the applications and exposing their capabilities to the End-Users through the iS interface, which is also called the NUBOMEDIA Signaling interface or just Signaling interface, along this document.
- NUBOMEDIA Media Plane composed by the media plane capabilities (Media Servers and Cloud Repository) whose lifecycle is managed by the Network

Function Virtualization Orchestrator (NFVO) and Virtual Network Function Manager (VNFM) following the guidelines of the ETSI NFV specification for the virtualization of Network Functions. Among the media server capabilities that are exposed to the external world we can find media transports, which are represented through the iM interface.

- NUBOMEDIA IaaS composed by the infrastructure resources in terms of Compute Nodes (CN) providing virtual compute, storage and networking resources to the upper layers, and the Virtual Infrastructure Manager (VIM) providing the abstracted APIs for controlling their lifecycle.

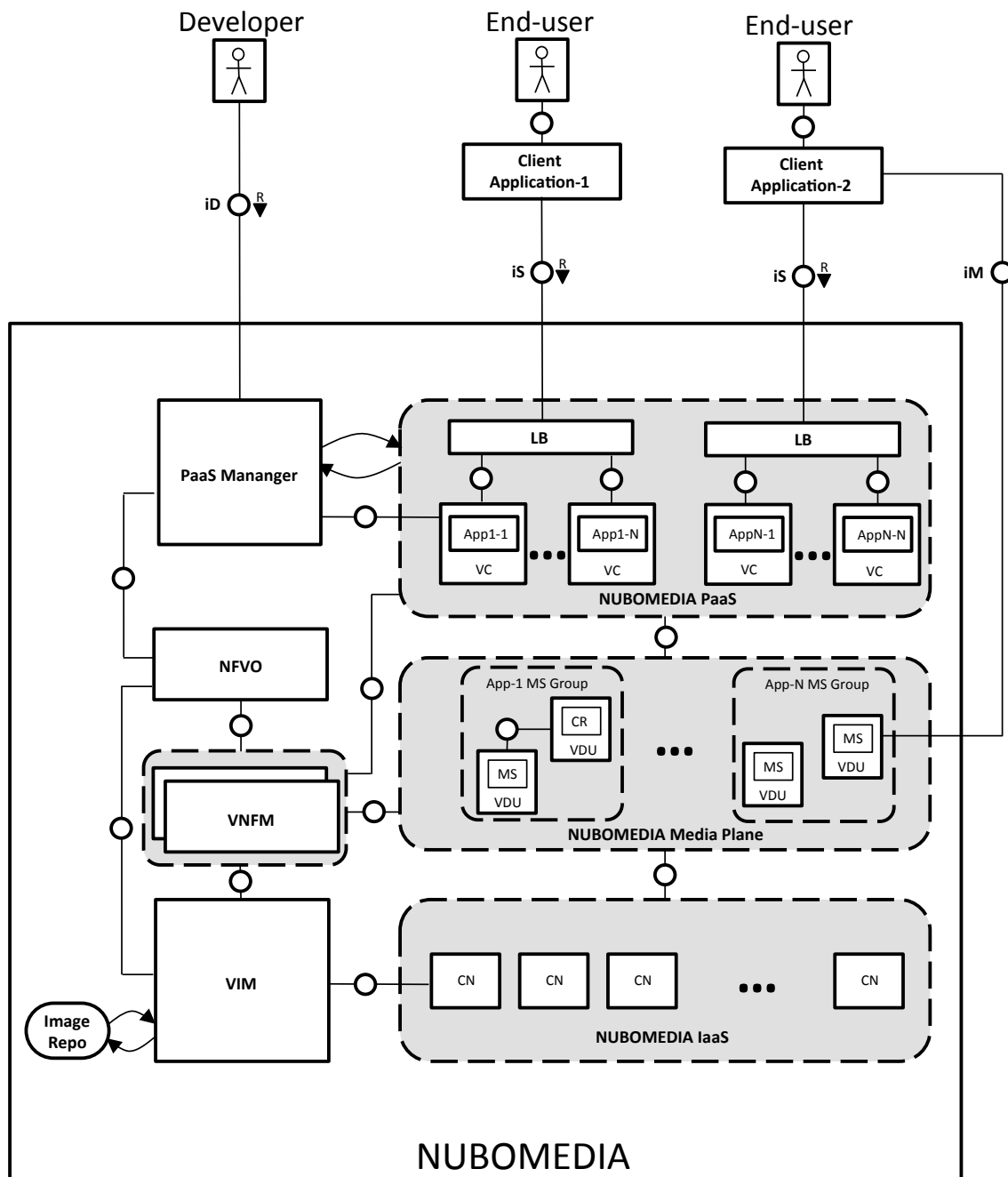


Figure 2. This figure shows the NUBOMEDIA system architecture following the Fundamental Modeling Concepts (FMC) block diagram visual notation. The NUBOMEDIA architecture is based on the ETSI NFV Management and Orchestration (MANO) recommendation but extends it with a PaaS layer. Thanks to it, when developers deploy their applications through a PaaS API, the PaaS Manager orchestrates all the required actions for the provisioning of the appropriate resources and services required for applications to run in a reliable and scalable way.

Application deployment overview

The iD interface is used by NUBOMEDIA application developers to deploy NUBOMEDIA applications. Those interface primitives are designed for capturing all the information that are necessary for the deployment. With that information, the PaaS Manager component is commanded to deploy the application. The application deployment requires the instantiation of an appropriate container with the appropriate application server where the application artifacts could be deployed.

In order to have a fully functional application, it is required that the media services are also available. For this reason, the PaaS Manager requests to the lower layer via, the Network Function Virtualization Orchestrator (NFVO), to create the appropriate Media Server (MS) instances. This process usually involves the instantiation of the virtual compute and networking resources via the VIM, and the execution of different lifecycle events via the VNFM(s).

Application consumption overview

The iS interface is used by end-user application to access application exposed services. Through this interface, signaling messages arrive to the NUBOMEDIA PaaS hosting all the deployed applications. A Load Balancer module performs message routing to the appropriate application instance, where the specific message semantics is executed. For this to happen, the PaaS Manager component controls, in runtime, the lifecycle of application containers so that they are instantiated and removed accordingly to end-user scaling needs.

As part of their logic and message semantics, NUBOMEDIA applications consume the media resources exposed by the NUBOMEDIA Media Plane. The NFVO creates one set of media services per application, so that media resources are isolated at application boundaries. Providing isolated sets of media services allows the creation of autoscaling mechanisms application-based. The idea is that the Media Plane control entities (NFVO/VNFM) provides an API to the applications for allowing the on demand scalability of the media service components. This API, described further in this document and in D3.2, provides capabilities of reserving media service components, and therefore allocate a certain number of media sessions to a specific instance.

4.3 NUBOMEDIA modules

4.3.1 The PaaS Manager

The PaaS Manager is the part of the system enabling developers to deploy and manage their server-side applications. The PaaS Manager is the component in charge of the full lifecycle of a NUBOMEDIA application, as defined before as a combination of the server-side application logic, hosted on the PaaS, and its required media functions, hosted on the NUBOMEDIA Media Plane, and NUBOMEDIA IaaS. Each time a developer requests the instantiation of an application, the PaaS Manager firstly requests the deployment of a set of Virtualized Network Functions (VNFs), defined as Network Service (refer to section 4.3.3 for more details), then instantiates the application on the PaaS providing all the details of the media functions that are dedicated to it. As shown in Figure 3, it comprises six main modules.

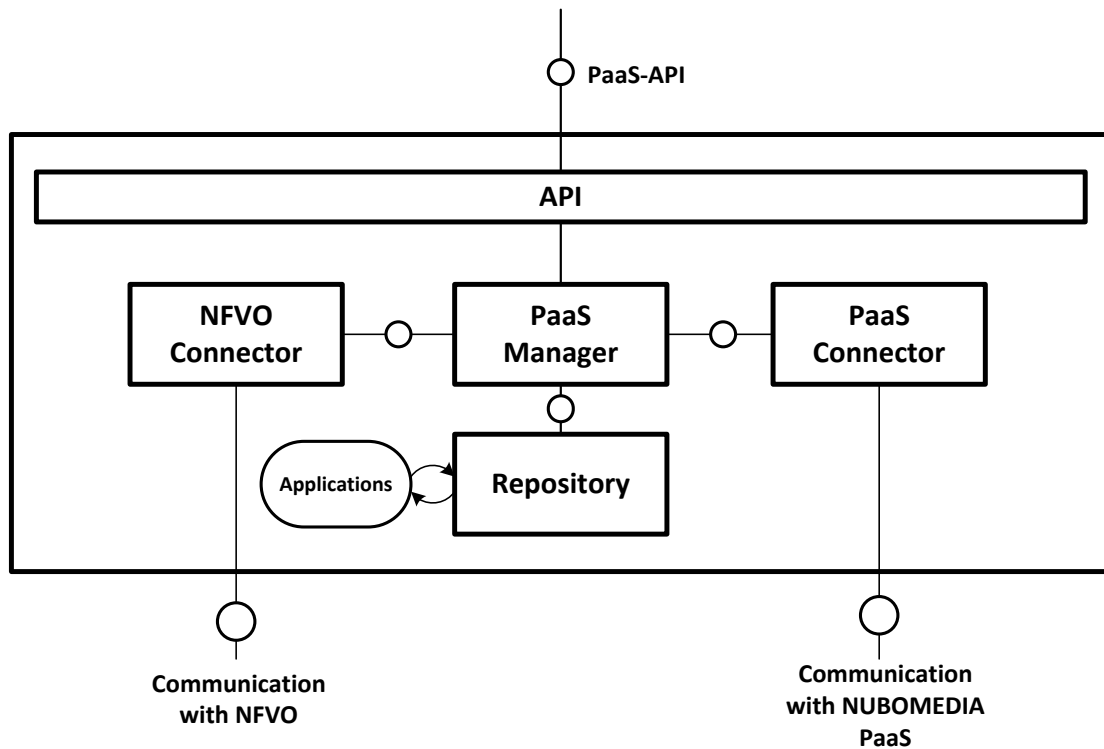


Figure 3. PaaS Manager internal architecture.

Those modules are briefly explained in the following paragraphs.

PaaS Manager API

This API abstract the underlying complexity involved in deploying applications directly on the PaaS. As show on Table 1, it is based on a simple REST interface offering CRUD primitives enabling creating, updating and removing applications.

Method	URL	Description
POST	/api/v1/nubomedia/paas/users	Create a new PaaS User
DELETE	/api/v1/nubomedia/paas/users/{name}	Deletes a PaaS User
POST	/api/v1/nubomedia/paas/app	Create (build and deploy) a new application
POST	/api/v1/nubomedia/paas/app/{id}	Retrieve status of a application with the given id.
GET	/api/v1/nubomedia/paas/app	Return the list of all deployed applications
DELETE	/api/v1/nubomedia/paas/app/{id}	Delete application with the given id
POST	/api/v1/nubomedia/paas/secret	Create a secret
DELETE	/api/v1/nubomedia/paas/{name}	Deletes a secret
POST	/api/v1/nubomedia/paas/oauth/authorize	Authenticate a PaaS User

Table 1. Summary of the REST API exposed by NUBOMEDIA at the iD interface. This interface is also called the PaaS Manager API along this document and makes possible for application developers to deploy their NUBOMEDIA-enabled applications into the NUBOMEDIA PaaS. For further information about this API, check NUBOMEDIA Project Deliverable D3.2.

The PaaS Manager internal module

The PaaS Manager internal module (let us qualify it as “internal”, for distinguishing from the container high-level module) provides the orchestration logic for application lifecycle management. It gives semantics to requests received by the PaaS API translating them into sequences of operations.

Repository

The repository interoperates with the persistent storage that store application meta-data and provides CRUD operations. The metadata of the deployed applications include:

- application identifier,
- application name,
- network service record identifier
- URL to the Git repository
- target ports exposed for the application
- number of replicas to be instantiated
- the secret key for accessing private repositories
- the deployment flavour for the media component
- additional configuration parameters which may be required by the media service components

NFVO Connector

This component consumes the services of the NFVO for requesting the instantiation of the network services required by the applications for providing media transport, processing and archiving.

PaaS Connector

The PaaS Connector connects the PaaS Manager to the NUBOMEDIA PaaS by abstracting the PaaS specific technology adopted, therefore providing an abstracted interface which can facilitate the integration of different PaaS systems.

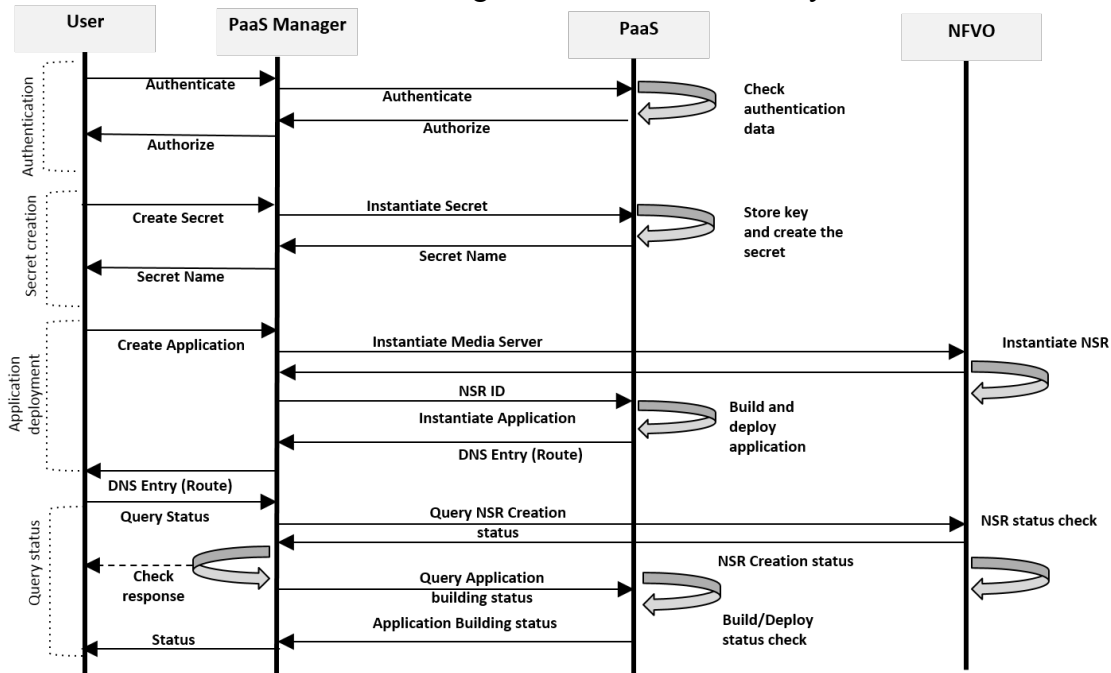


Figure 4. PaaS Manager Interactions

4.3.1.1 PaaS Manager main interactions

As depicted in Figure 4, the PaaS Manager provides four main functionalities to the user interacting with NUBOMEDIA.

Authentication: Before the user can interact with the PaaS, it needs an authorized token for using the PaaS API. For this purpose, users can use the PaaS Manager API to authenticate themselves on the PaaS. With the response a token is sent to the user which will be used in subsequent request in the duration of the session.

Secret creation: The Secret object provides a mechanism to hold sensitive information such as passwords, PaaS configuration files, configuration files and private source repository credentials. The User can create a secret data object and send a request via the PaaS Manager to the PaaS requesting storage of the secret data. The PaaS stores the secret data and generates a secret name to identify this data, which is sent back to the PaaS Manager, which forwards this to the user. An example where this is applicable is when developers use private repositories for their applications. They can then create a secret object with the credentials to access this private repository. This information is used in the application deployment phase.

Application deployment: To deploy an application, the user creates a JSON object containing the meta-data of the application as described in section. This JSON object can be sent via an HTTP Post Request to the PaaS Manager. Thereafter the PaaS manager orchestrates a series of requests to the NFVO component for deploying the Network Service containing the media service capabilities required, and a set of requests to the PaaS for deploying (using application meta-data and secret data object if required) the server-side application on a specific Application Server. With successful deployments, the user is provided a route pointing to a DNS entry from which the application can be reached, otherwise an error message.

Query deployment status: Depending on the application specific deployment configuration requirements, the deployment procedure on the PaaS and on the media plane could take a few seconds or longer. For long procedures, the user always has the option to query the deployment status via the PaaS Manager REST interface.

4.3.2 NUBOMEDIA PaaS

The NUBOMEDIA PaaS hosts the server-side application logic on top of on demand deployed Application Servers. For this, it needs to provide a set of services to the PaaS Manager, which mediates between developers and the PaaS. These services must make possible to deploy, scale and manage the application lifecycle. As shown in Figure 5, the NUBOMEDIA PaaS architecture comprise a number of building blocks, which are the following:

Core services. These include the following:

- **Authentication:** providing the authentication mechanisms enabling to determine the applicable security policies and permissions for a user.
- **Data Store:** this service provides data persistence and recovery capabilities for the different types of information required to deploy and manage an application.
- **Scheduler:** this service is responsible of the placement of application instances onto nodes within the cluster (see discussion below for understanding what's a pod and a node).

- **Management/Replication:** this service ensures that a specific number of application instance replicas are running at all times. This service manages application scalability so that, whenever there are too many pods running some are killed, and whenever too few pods are in execution some are launched. This service also replaces pods terminated pods (e.g. failure, disruptive node maintenance, etc.)

APIs. The core services of the NUBOMEDIA PaaS are provided as micro-services. Micro-services is a software architecture approach in which complex applications are split into small independent processes which communicate among each other using language agnostic APIs. As shown above, our core components are micro-services, which interact among each other and with the external world through common REST APIs.

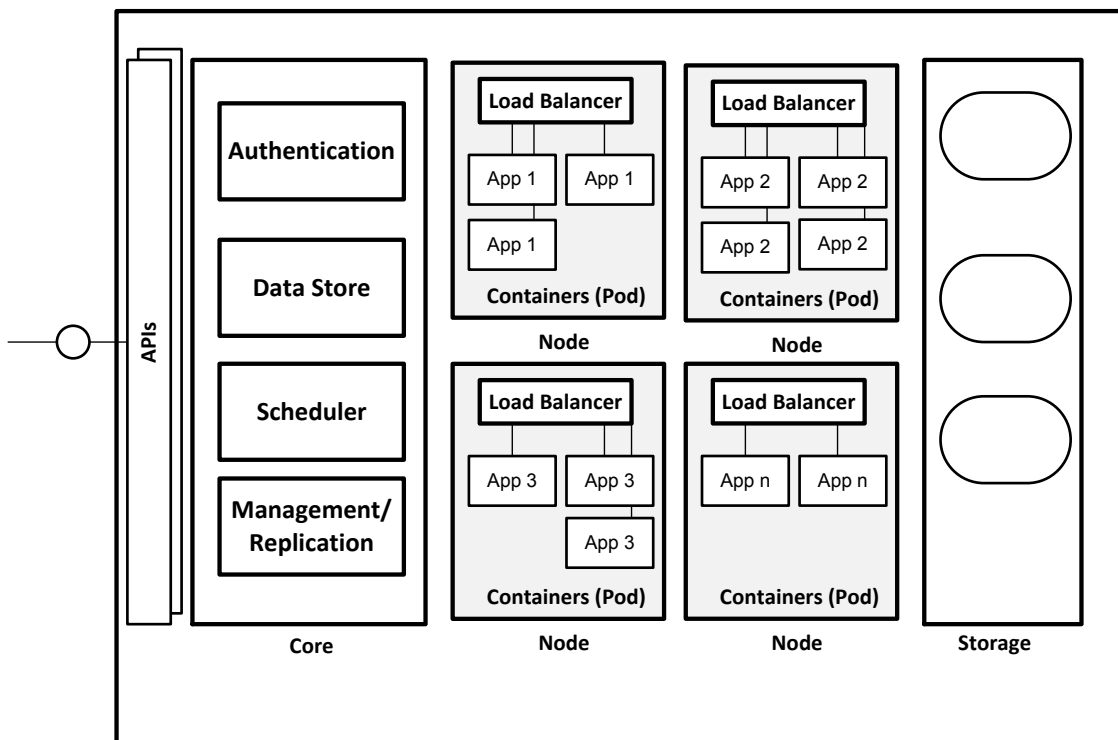


Figure 5. The NUBOMEDIA PaaS architecture

Containers, pods and nodes. Containers, pods and nodes are concepts imported from Kubernetes (<http://www.kubernetes.io>). Hence, the interested reader should refer to Kubernetes documentation for further information. For the objectives of this document, it is enough to say that containers are the basic unit of NUBOMEDIA PaaS applications. A container corresponds with the Linux notion of containers as a lightweight mechanism for isolating running processes. The NUBOMEDIA PaaS uses Docker containers (<http://www.docker.com>). Nodes, in turn, are worker machines, physical or virtual, where pods are run. Following Kubernetes terminology, a *pod* is an application logical host in a containerized environment. In other words, pods are collocated groups of containers that make up the application. Hence, nodes are capable of running containers and of managing groups of containers. The interesting feature about nodes is that they expose a simple network proxy and a load balancer suitable for forwarding traffic basing in round-robin mechanisms across sets of containers (i.e. pods). Hence, given that pods can be replicated at any time, and given that the same pods can be deployed across multiple nodes, the PaaS provides the appropriate horizontal scaling and redundancy for any service packaged into a container image.

Registry. The registry is a service for storing and retrieving Docker images. It contains a collection of Docker image repositories. In the case of NUBOMEDIA, all applications are deployed using Docker containers and the PaaS provides an internal registry for managing custom Docker images.

The relationship between Docker containers, images, and registries is depicted in Figure 6. All applications are deployed using Docker containers. A Docker container uses Docker image which is a binary that includes all of the requirements for running a single Docker container, as well as metadata describing the application's needs (e.g. require Java Runtime Environment, etc.) and capabilities of the application. Different Images for different versions of the application can be defined and stored in the Registry and used later during deployment on different containers.

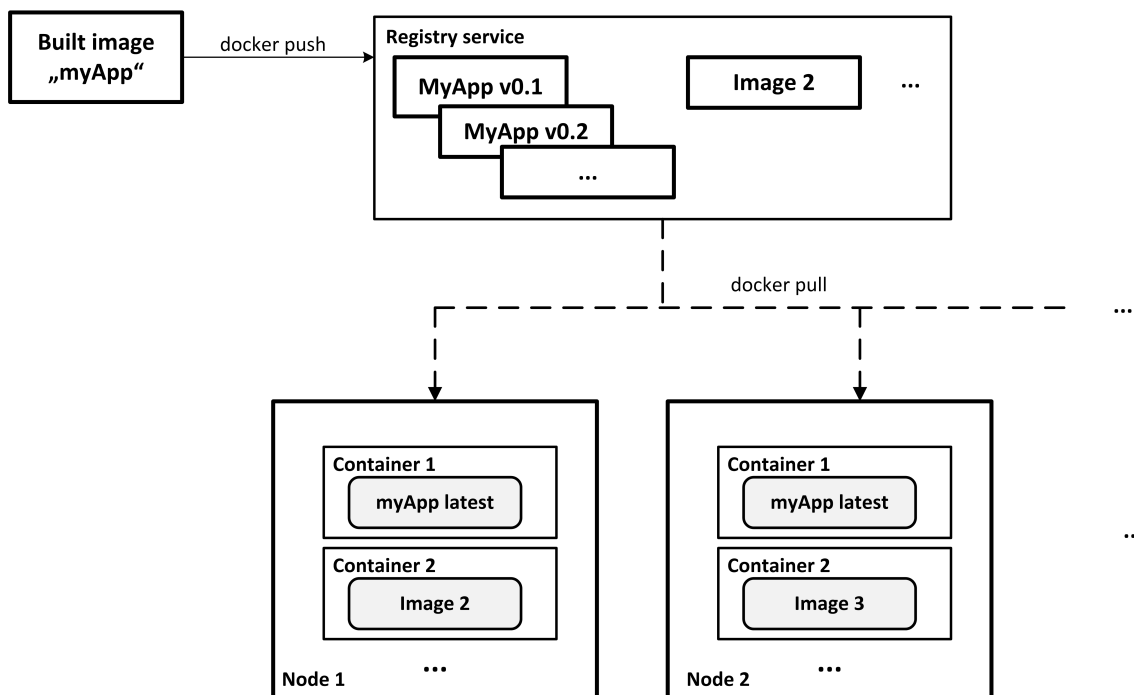


Figure 6. Relationship between Docker containers, images, and registries

4.3.3 Network Function Virtualization Orchestrator

In the ETSI white paper [ETSI_WP] the definition of NFV is that it “*aims to transform the way that network operators architect networks by evolving standard IT virtualization technology to consolidate many network equipment types onto industry standard high volume servers, switches and storage, which could be located in Datacenters, Network Nodes and in the end user premises. It involves the implementation of network functions in software that can run on a range of industry standard server hardware, and that can be moved to, or instantiated in, various locations in the network as required, without the need for installation of new equipment.*”

NFV is designed to consolidate and deliver the networking components needed to support a fully virtualized infrastructure, including virtual servers, storage and even other networks. It utilizes standard IT virtualization technologies that run on high-volume service, switch and storage hardware to virtualize network functions. It is applicable to any data plane processing or control plane function in both wired and wireless network infrastructures.

Network Function Virtualization considers the implementation of NFs as entities only in software that run over the Network Function Virtualization Infrastructure. Considering that Media Servers are intrinsically Network Functions, NUBOMEDIA followed the ETSI NFV specification for the management and orchestration of the media service components. The Network Function Virtualization Orchestrator (NFVO) is the component managing the lifecycle of a Network Service (NS) composed by multiple MS and a Cloud Repository.

The NFV information model is generally divided in two sets of entities: Descriptors and Records. Information inside Descriptor elements are rather static information mostly used for initiating the on-boarding process of the Virtualized Network Functions (VNFs). Information inside Record elements are instead, dynamic information that are the result of the on-board process and will vary during the time. The cause of this variation can be a lifecycle operation completion or even a result of an unexpected event (i.e. a fault)

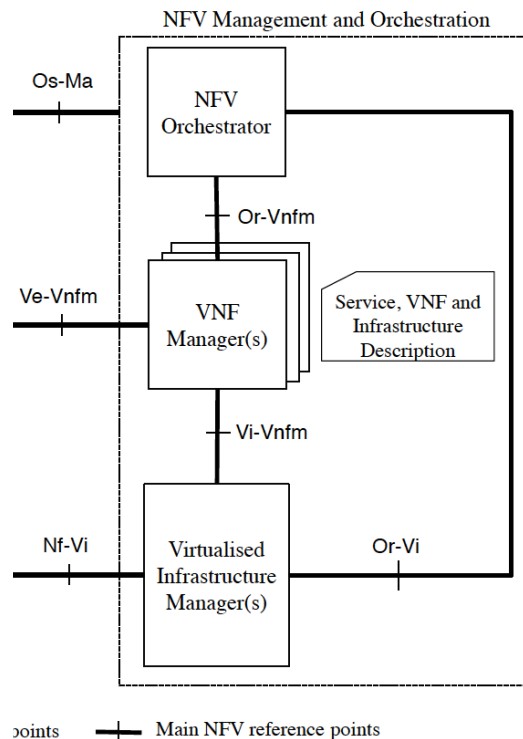


Figure 7. ETSI NFV MANO Architecture

The main entity is the NS that describes the relationship between VNFs and possibly Physical Network Functions (PNFs) that it contains and the links needed to connect VNFs that are instantiated on the Network Function Virtualization Infrastructure (NFVI), which is basically what is defined with NUBOMEDIA IaaS in this document. A VNF is a Network Function implemented as a software component, a functional block with a clear interface and behavior. Hence, it is possible to deploy it in a virtualized infrastructure. The VNFD (Virtual Network Function Descriptor) describes a VNF in terms of its deployment and operational behavior requirements. A VNF can be instantiated either on a single or multiple VNF Components (VNFC). The VNF Manager (VNFM) makes use of the VNFD while instantiating and managing the lifecycle of the VNF. The information provided in the VNFD are also used by the NFVO to manage and orchestrate Network Services and virtualized resources on the NFVI. The VNFD contains connectivity, interface and KPIs requirements that can be used by NFV-MANO functional blocks to establish appropriate Virtual Links within the

NFVI between its VNFC instances, or between a VNF instance and the endpoint interface to the other VNF.

This information model is internally used by the NFVO, by the VNFM and by the Virtualized Infrastructure Manager (VIM). The Figure 7, taken from the ETSI NFV MANO [MANO] specification, shows the ETSI NFV architecture for the MANO domain and specifies how the main three components are connected. As it can be noticed, NUBOMEDIA followed the ETSI NFV MANO architecture and extended it for supporting additional capabilities.

While NFVO is mainly responsible of the management of the lifecycle of the Network Services, the VNFM is responsible for the lifecycle management of a single VNF (e.g. instantiation, update, query, scaling, termination, etc.). From the NFV specification either NFVO or VNFM shall instantiate virtual resource through the VIM. As it can be observed in Figure 7, both components have an interface to the VIM. The selection of either the first or the second approach is left completely to the VNF developer, however the allocation of resources is done by the NFVO which is the only component having a complete view of available resources on the NFVI.

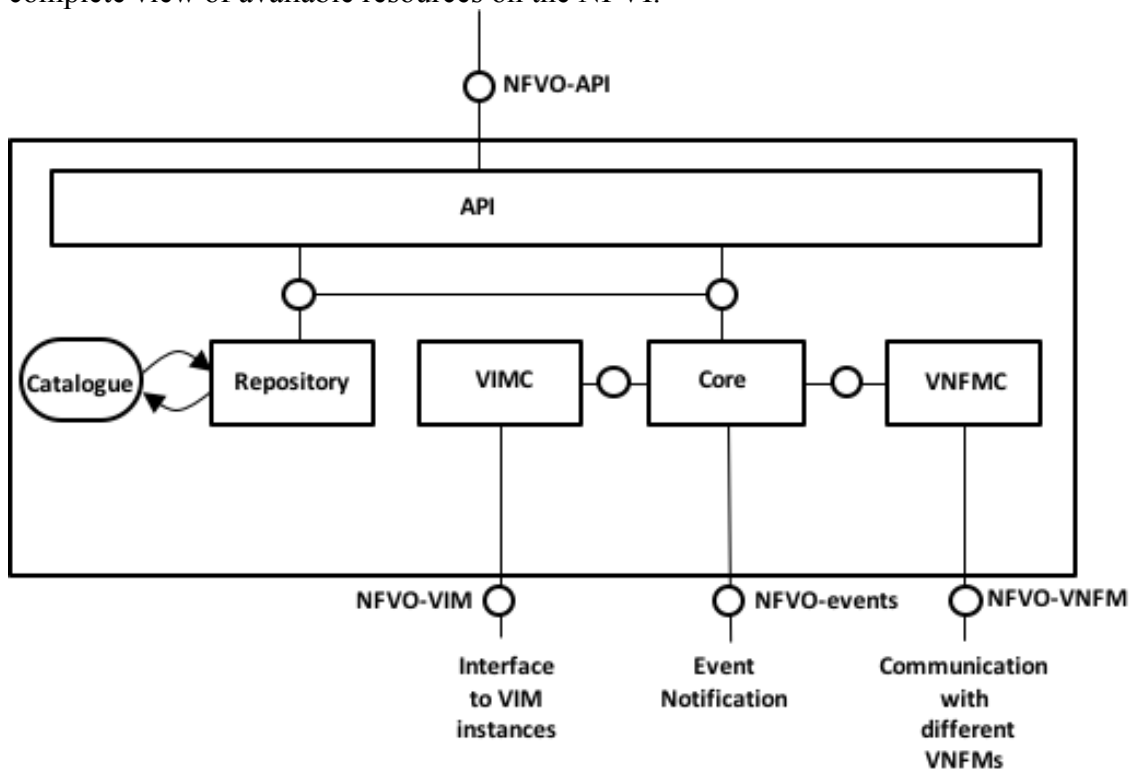


Figure 8. NFVO internal architecture.

The VNFM provides an API to the NFVO for managing the lifecycle of the VNF. It can be specific, in case the VNF it manages requires specific mechanisms for being managed, or generic, in case the VNF all the lifecycle events can be described in the descriptor and do not require additional capabilities. In NUBOMEDIA both approaches were used. In particular, the media service components are managed by the Elastic Media Manager (EMM), which is the name associated with the Media Server VNFM (MS-VNFM), as it extends the standard functionalities of the VNFM providing also specific autoscaling mechanisms and media session placement.

As already mentioned, the NFVO manages the lifecycle of a Network Service. It exposes an interface to the PaaS Manager providing functionalities for managing the Network Service. This interface enables the following operations: 1) instantiation of the

virtual computing resources (via the VIM interface) required by the Network Service, 2) provisioning of guaranteed networking resources and 3) management of the lifecycle of the different VNFs via the VNFM. As shown in Figure 8, the NFVO comprises the following modules:

- API, which exposes the NFVO-API consumed by the PaaS Manager. Following ETSI terminology, this API makes possible to instantiate Network Service Records (NSRs) from Network Service Descriptors (NSDs). An NSD is a deployment template that describes a Network Service in terms of its deployment and operational behavior requirements. NSD are represented as JSON files and are stored into a catalog. Inside the NSD we can find the VNF Descriptors (VNFD) containing the Virtual Deployment Unit (VDU) Descriptors describing aspects such as instance flavor, deployable components, scaling limits, etc. An instantiation of an NSD is known as an NSRs, while an instantiation of a VNFD is a VNF Record (VNFR).
- Repository, which exposes to the NFVO the Catalogue: a repository where the different resources available to the NFVO (such as the NSDs) are stored.
- Virtual Infrastructure Manager Connector (VIMC): it is a module used by the Core for interoperating with the VIM.
- Core, represents the central module of the NFVO. It coordinates all the lifecycle events of the Network Service Records instantiation. It has several fundamental functionalities: from the management of the catalogue to the actual instantiation of a Network Service Record (NSR) starting from a Network Service Descriptor (NSD).
- VNFM Connector (VNPMC), which provides an internal interface for dispatching messages to the different VNFMs which are registered.

4.3.4 Connectivity Manager

The Connectivity Manager (CM) provides mechanisms for controlling QoS parameters, like bandwidth, between media service components running on the IaaS (further description is given in D3.3.1). Basically, the CM provides capabilities for enforcing specific level of QoS between VNF Components.

For this, the CM makes use of CM Agents components running directly on the NFVI that are able to control network traffic based on requirements of the applications. The CM Agent uses Software Defined Networks (SDN) techniques and, in particular, it creates queues and assigns priorities to flows. The CM Agent exposes a REST API providing the capabilities of selecting a specific QoS level (in terms of desired bandwidth) among different VDUs. The CM abstracts even more the low level details providing the NFVO an API for controlling the QoS level for a certain VNF.

4.3.5 Virtual Network Function Manager

The VNFM, as already mentioned before, provides lifecycle management for a VNF. The VNFM can be specific (i.e. adapted to a specific type of VNF) or generic. In NUBOMEDIA we use the two. The MS-VNFM is specific and its functionalities are adapted to managing Media Server functions. On the other hand, the Cloud Repository one (CR-VNFM) is based on a generic architecture. Both expose the same type of interface to the NFVO, as specified by ETSI MANO. In both cases also, the main function of the VNFM is to deploy and manage a specific VNFR (Virtual Network Function Record), as specified in the corresponding VNFD.

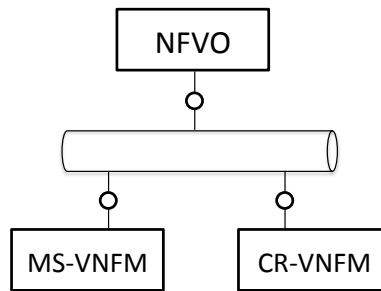


Figure 9. Following ETSI MANO architecture, the MS-VNFM and the CR-VNFM are managed by the NFVO through a control bus

The core of the MS-VNFM, shown in Figure 10, contains the following components:

- **Media Server Manager:** The Media Server Manager is in charge of processing all the operations of the Lifecycle Management (i.e. VNFR instantiation, VNFR termination, start, modify, configure, etc.) that are triggered by the NFVO at the predicted point in time. So, first it starts with the VNFR instantiation requested by the NFVO. Once all the components of the specific VNFR are instantiated, the NFVO will call the start procedure of the VNFM to prepare available Media Servers, start the elasticity, and finally let the VNFR go into the ACTIVE state reflecting that all Components are up and running ready for the foreseen activities. Once the termination of a specific VNFR is requested by the NFVO to the VNFM, the Media Server Manager releases all resources and terminates the management of the corresponding VNFR by notifying the NFVO.
- **Media Server Management:** The Media Server Management is responsible for managing the Media Servers that offers resources for Applications and Media Pipelines. Media Servers are extensions of VNFCInstances (Virtual Network Function Component Instances - or simply Virtual Machines) where new Applications will be assigned to. Once new Applications are registered, the capacity of corresponding Media Servers will be reduced by the amount of capacity the Applications have requested. After unregistering Applications, the capacity of Media Servers will be released respectively.
- **Application Management:** The Application Management is in charge of managing Applications by providing the capabilities for registering and unregistering Applications. For this purpose, it requests the Media Server Management for allocating and releasing corresponding resources for newly created or removed Applications and Media Pipelines.

Furthermore, the MS-VNFM provides an API to applications deployed in the NUBOMEDIA PaaS for retrieving dynamically the available Media Servers. This API is consumed by the NUBOMEDIA Media API implementation to determine in which specific Media Server instance a newly created Media Pipeline is placed. Allocating new Media Pipelines is transparent to the developer, who just needs to create the Media Pipelines without worrying about where they are located. For this, the VNFM has been extended with the autoscaling component, providing semantics to that placement interface and guaranteeing the availability of Media Servers through an horizontal autoscaling mechanism based on two operations: scaling-out (i.e. to add resources when necessary) and scaling-in (i.e. to remove resources when they are no longer required). Both, the scaling-in and -out, are fired by simple autoscaling policies based on QoS metric thresholds, so that, for example, further resources are added when the average CPU load of Media Server instances is over an upper limit and resources are collected when it's under a lower bound.

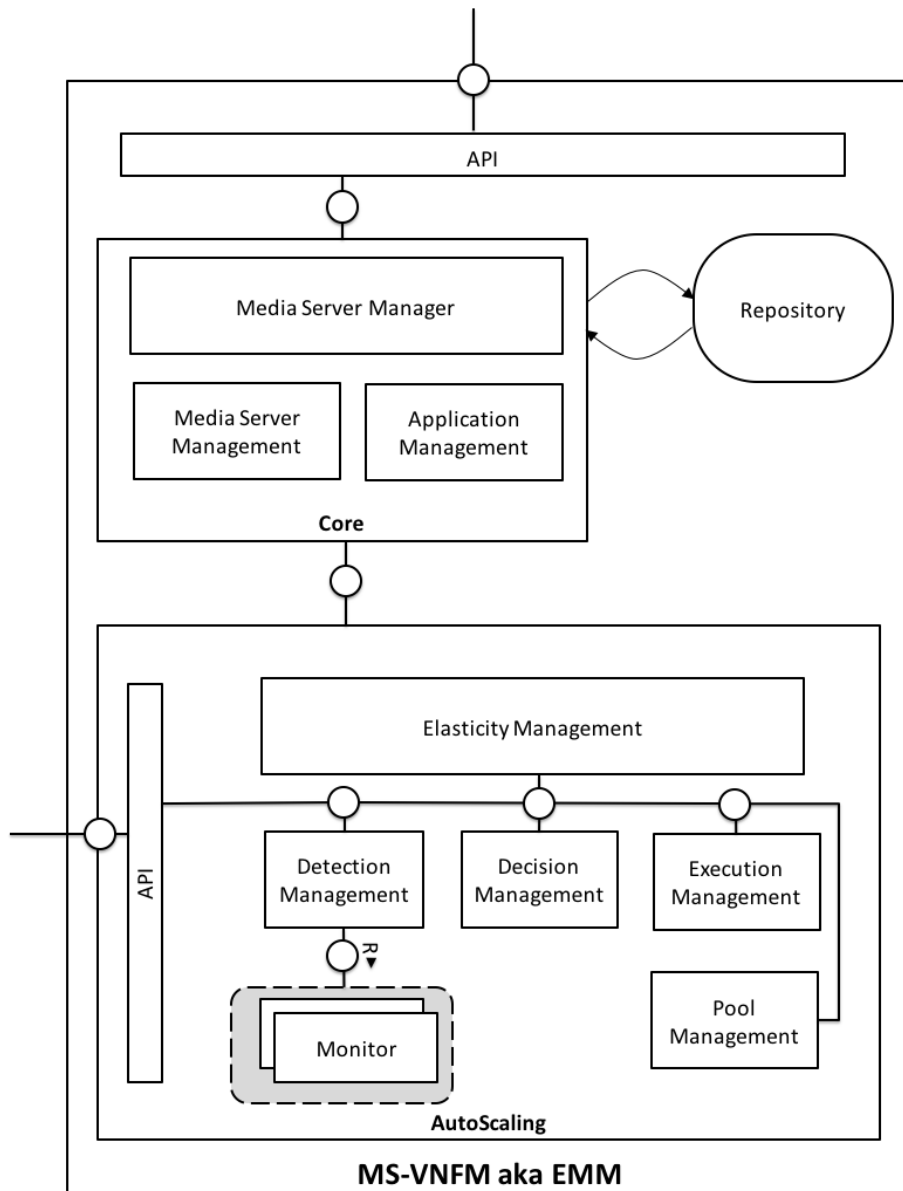


Figure 10. MS-VNFM Architecture extended the basic VNFM with the EMM

As specified in the ETSI NFVO MANO, autoscaling is spread over several components in order to detect the need of scaling (Detection Management), to make decisions (Decision Management) and finally to execute the actions (Execution Management) that were decided before. Additionally, the autoscaling system is extended with a Pool mechanism (controlled by the Pool Management) to decrease significantly the time of allocating new Media Servers. This pool contains Media Servers that allows to provide already prepared and launched Media Servers on demand without waiting the time needed for deploying a completely new Media Server. At runtime, the Alarm Detection is triggered by Elasticity Management whereas the Elasticity Management is either notified by the NFVO once the initial deployment is finished or directly through the VNFM when the VNFR goes into ACTIVE. The CR-VNFM, in turn, is simpler and manages the lifecycle of Cloud Repository elements. For this, it just receives information about the virtual resources instantiated by the NFVO and installs the repository components.

4.3.6 Virtual Infrastructure Manager

The VIM provides an interface for controlling the NUBOMEDIA IaaS. As in the ETSI NFV specification, the VIM follows the OpenStack approach and, through its APIs, offers the ability to start new computing resources by using already pre-configured images containing the appropriate artifacts for every of the required functions. The computing nodes are instantiated on the NUBOMEDIA IaaS on top of one or more Compute Nodes. Computing nodes are distributed across physical machines depending on the required flavors and resources.

4.3.7 NUBOMEDIA Media Plane

The NUBOMEDIA Media Plane provides the media capabilities to NUBOMEDIA applications. These capabilities include media transport, media archiving and media processing. The scalability of the NUBOMEDIA Media Plane is controlled by the NFVO component, which adapts it to the load offered by applications.

As shown in Figure 2, the NUBOMEDIA Media Plane basic component is the Media Server (MS), so that it can be seen as a collection of MS instances whose lifecycle is controlled by the NFVO and MS-VNFM and whose relationships and interdependencies depend on the application logic. MS instances are grouped in a per-application basis so that each application is assigned, in exclusivity, a MS group. This means that each application has an independent and isolated media plane.

4.3.7.1 Media Server architecture

As shown in Figure 11, Media Servers hold a number of media capabilities, which may include media transport, media transcoding, media processing, media mixing and media archiving. These media capabilities are encapsulated into abstractions called Media Elements (ME), so that Media Elements hide the complexities of a specific media capability behind a comprehensive interface. MEs have different flavors (i.e. types) in correspondence with their different specialties (e.g. *RtpEndpoint*, *RecorderEndpoint*, *FaceDetectorFilter*, etc.)

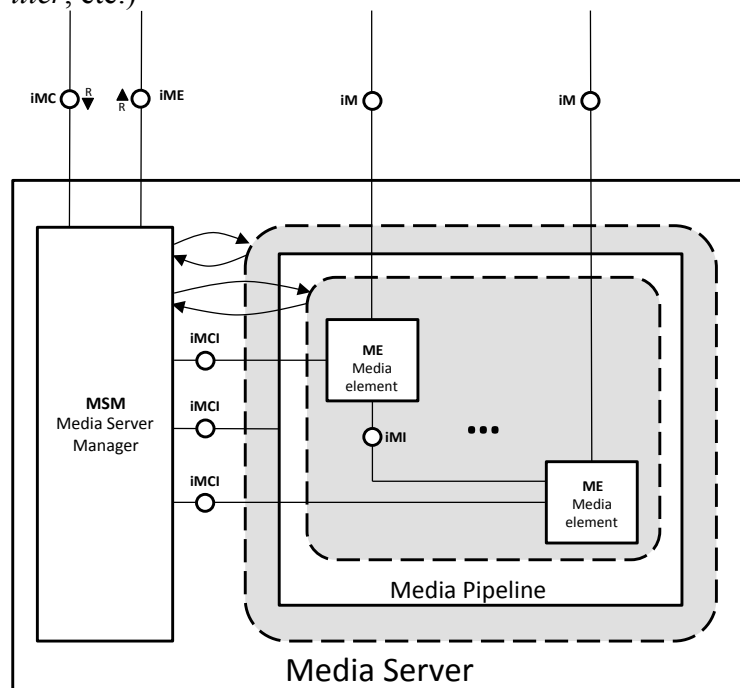


Figure 11. Media Server (MS) component internal architecture. A MS comprises a number of Media Pipelines (MPs). MPs are graphs of interconnected Media Elements (MEs). Each ME implements a specific media capability: media transport, media archiving or media processing. MS pipelines and elements are controlled by the Media Server Manager, which communicated with the external world through the Media Control interface (iMC) and the Media Events interface (iME).

The Media Element component

As shown on Figure 12, a Media Element can be seen as a black box taking media from a sink, applying a specific media capability (e.g. processing, transporting, etc.), and issuing media through a source. As a result, Media Element instances can exchange media among each other through the Media Internal interface (iMI) following this scheme:

- A Media Element can provide, through its source, media to many sink Media Elements.
- A Media Element can receive, through its sink, media from only one source Media Element.

Media Element instances can also exchange media with the external world through the Media interface (iM). Hence, the iM interface represents the capability of the Media Server to send and receive media. Depending on the Media Element type, the iM interface may be based on different transport protocols or mechanisms. For example, in a Media Element specialized in RTP transport, the iM interface shall be based on RTP. In a Media Element specialized in recording, the iM interface may consist on a file system access.

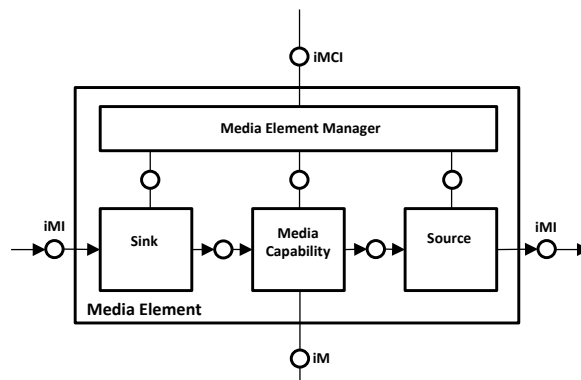


Figure 12. Media Element (ME) component internal architecture. A ME has a sink, where media may be received and a source, where media can be send. Sinks and Sources exchange media with other MEs through the Media Internal interface (iMI). ME specific capabilities are encapsulated behind an interface that it made available to the rest of MS components through the Media Control Internal interface (iMCI)

Remark, however, that not all Media Elements need to provide a iM interface implementation. Media Elements having the ability of exchanging media with the external world (and hence, providing an iM implementation) are called **Endpoints**. Media Elements which do not provide an iM implementation are called **Filters**, because their only possible capabilities are to process and transform the media received on its sink. Their architectural differences are shown on Figure 13.

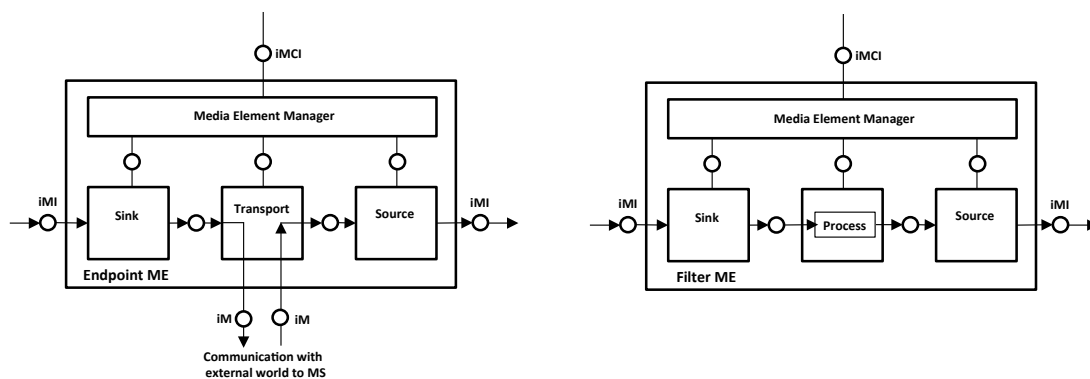


Figure 13. Endpoint MEs (left) are MEs with the capability of exchanging media with the external world through the iM interface. In an Endpoint, media received through its Sink is forwarded to an external entity (e.g. the network). Media received from the external entity is published into the Endpoint Source. Typically, Endpoint MEs publish events associated with the transport status (e.g. media-connection-established, etc.) Filter MEs (right) do not implement the iM interface and cannot exchange media with the external world. Hence, Filter MEs can only be used for processing the media received through their Sinks, which is then published to their Sources. Filter MEs typically publish events associated to the specific media processing being performed (e.g. face-detected in a *FaceDetector* filter)

The Media Element behavior is controlled by the Media Element Manager (MEM), which communicates with other MS components through the Media Control Internal interface (iMCI). The iMCI provides

- A mechanism for sending commands to the Media Element.
- A mechanism for publishing media events and state information from the Media Element.

There is special type of Media Element called a Media Hubs. A Media Hub extends Media Elements capabilities with the ability of receiving a variable number of media streams at its input. In other words, Media Hubs are Media Elements with a variable number of sinks. Media Hubs are typically used for managing groups of streams in a consistent and coherent way.

The Media Pipeline component

Media Element capabilities can be complemented through a pipelining mechanism meaning that media issued by a specific Media Element can be fed into a subsequent Media Element whose output, in turn, could be fed with the next Media Element and so on and so forth. A graph of interconnected Media Elements is called a Media Pipeline (MP). Hence, a Media Pipeline is a set of Media Elements plus a specification of their interconnecting topology. Remark that Media Pipeline topologies do not need to be “chains” (sequences of interconnected Media Elements) but can take arbitrary graph topologies as long as the Media Element interconnectivity rules specified above are satisfied.

Hence, in a Media Pipeline one can combine in a very flexible way Media Element capabilities enabling applications to leverage different types of communication topologies among end-users, which may include any combination of simplex, full-duplex, one-to-one, one-to-many and many-to-many.

Media Pipeline capabilities are managed by the Media Pipeline Manager module, which controls the lifecycle of its composing Media Elements, forwards them commands from the external world and publishes Media Element events to subscribers. The Media Pipeline Manager uses the iMCI interface for performing these actions.

A Media Server instance can hold zero or more Media Pipelines.

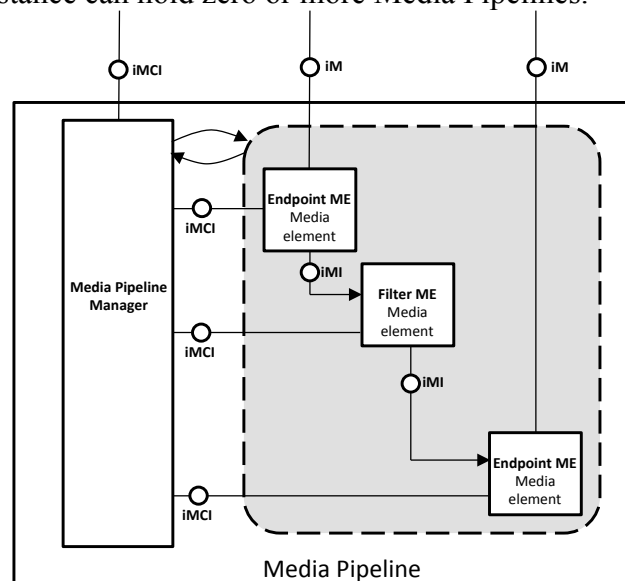


Figure 14. Media Pipelines (PMs) are graphs or interconnected Media Elements (MEs) that implement a specific media processing for an application. The Media Pipeline Manager controls the lifecycle and behavior

of its MEs by brokering control messages and events among the MEs and the Media Server components our of the MP.

The Media Server Manager component

As shown in Figure 11, the Media Server Manager (MSM) component manages the lifecycle and behavior of MPs and MEs on behalf of applications. For this, The Media Server Manager exposes two interfaces to the external world:

- The Media Control Interface (iMC): This interface is based on a request/response mechanism and enables applications to send commands to both Media Pipelines and Media Element instances. Through these commands, applications can access features such as:
 - Create and delete Media Pipelines and Media Elements
 - Execute specific built-in procedures on Media Pipelines or Media Elements
 - Configure the behavior of Media Pipelines and Media Elements
 - Subscribe to specific events on Media Pipelines and Media Elements

The Media Server Manager translates commands received on the iMC interface into invocations to the corresponding Media Elements and Media Pipelines through the iMCI interface.

- The Media Events interface (iME): This interface is based on a request/response mechanism and enables Media Element instances to send events to applications, that are then acknowledged. These events may include semantic media information (e.g. detected a face in a *FaceDetector* filter), media transport information (e.g. media negotiation completed), error information, or any other kind of information considered relevant by the Media Element developer. For this purpose, the Media Server Manager translates any events received in

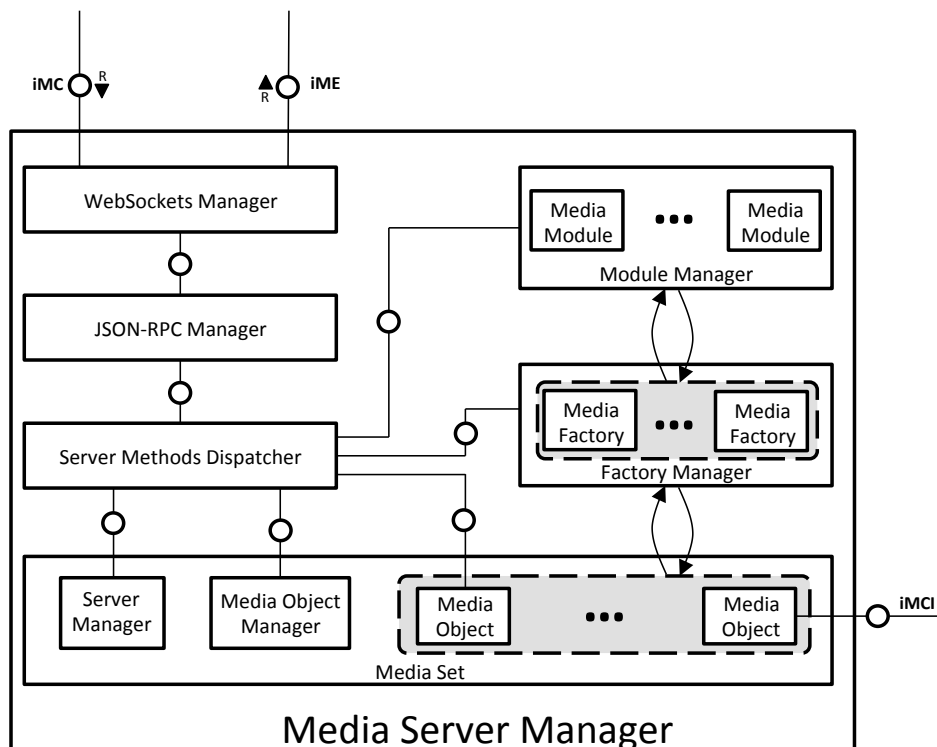


Figure 15: The Media Server Manager (MSM) holds the appropriate logic for performing two functions. The first is to enable communications with applications through the iMC control interface and the iME event publishing interface. The second is to provide the appropriate logic for dispatching, managing and providing semantics to control messages and events, including the ones in charge of media capabilities lifecycle management (e.g. create, release, etc.)

A detailed description of the Media Server Manager internals is shown on Figure 15. As it can be observed, it comprises several modules playing specific functions each.

- The WebSockets manager provides connection-oriented full-duplex communications with applications basing on the WebSocket protocol. This module is in charge of the WebSocket line protocol and of transporting data on top of it.
- The JSON-RPC Manager provides JSON-RPC marshaling and unmarshaling of messages, accordingly to JSON-RPC v2 standard.
- The Server Methods Dispatcher (aka Dispatcher) module contains the logic for routing messages to their appropriate destinations, so that the message semantics can be provided. The Dispatcher can orchestrate actions on several other modules for obtaining that semantics.
- The Module Manager is where module information is held. A module is a collection of one or several media capabilities (i.e. Media Element). All the required information for using a module in the Media Server is called a Media Module in our architecture. Media modules are loaded upon Media Server startup. The Media Module contains the module meta information (e.g. module name, module locations, etc.) The Module Manager exposes a service for querying Media Module information to the rest of modules.
- The Factory Manager controls the lifecycle of Media Factories. Upon Media Module loading, the Module Manager registers one or several Media Factories into the Factory Manager. Each Media Factory has the ability of creating a specific media capability among the ones provided by the module. This creation process is mediated by Media Stubs, so that Media Factories only create Media Stubs and are the Media Stubs themselves which create the corresponding Media Element instances.
- The Media Set holds Media Objects, which are proxies to the media capabilities themselves (i.e. the real Media Elements and Media Pipelines). It also holds a number of facilities for managing them. In particular, the Media Object Manager translates media object references (as managed in the iMC interface) into object references pointing to the appropriate Media Objects. The Media Objects provide a mechanism for mediating between the Media Server Manager and the media capabilities themselves. All requests coming from the iMC interface targeting a specific Media Element or Pipeline are routed by the Dispatcher to the corresponding Media Object, which provides it semantics consuming the Media Element or Pipeline primitives through the iMCI interface.
- The Media Object Manager also takes the responsibility of media session management. Media sessions are an abstraction provided by the iMC interface through a media session id. All JSON-RPC messages exchanged in the context of a session carry out the same id, which is unique for the session. The Media Object Manager leverages media sessions for implementing a Distributed Garbage Collection mechanism. For this, the Media Object Manager maintains the list of Media Objects associated to a specific media session. Whenever a media session is not having any active WebSocket connection during a given time period it is considered as a death session. In this case, all its Media Objects are released together with their corresponding media capabilities.
- The Server Manager is an object enabling Media Server introspection. For this, it queries the Media Object Manager to recover the list of Media Objects, which represent the Media Pipelines and Media Elements, as well as their

interconnecting topologies. This makes possible for applications to monitor, instrument and debug what is happening inside the Media Server.

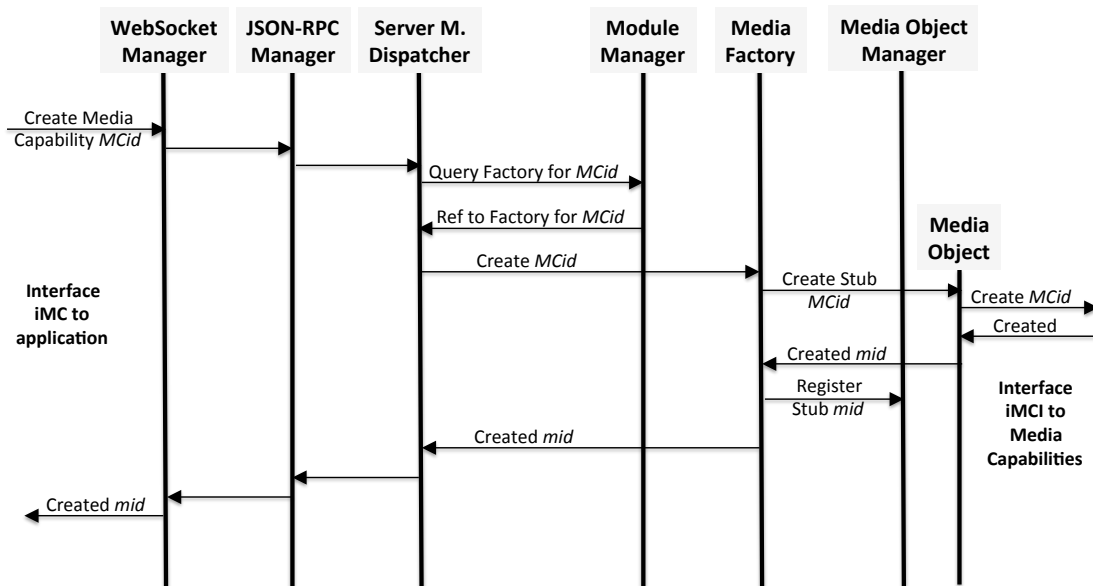


Figure 16. Flow diagram showing the interactions among the Media Server Manager modules for creating a new media capability (i.e. Media Element or Media Pipeline). The specific media capability type is identified by the string MCid (Media_Capability_id), that needs to be registered among modules capabilities into the Module Manager. Once the capability has been instantiated, it receives a unique id (mid or Media_id), which identifies it during its whole lifecycle

For understanding better the interactions among MSM modules let's observe Figure 16, where the sequence diagram for the instantiation of a specific media capability is shown. From top to bottom, the steps are the following:

- A message asking for media capability creating is received at the MSM WebSocket Manager. This message needs to identify the specific media capability type that is to be created through an ID, which we call MCid (Media_Capability_id) in the diagram.
- This message is given to the JSON-RPC Manager which unmarshalls it as a data structure.
- The unmarshalled message is given to the Dispatcher, which recognizes it as a capability creation message, and orchestrates its semantics.
- The Dispatcher queries the Module Manager for recovering the specific Media Factory knowing how to create MCid capabilities. We assume one Media Module has registered factories for MCid upon Media Server startup.
- The Module Manager provides the Dispatcher with a reference to the specific Media Factory creating MCid capabilities.
- The Dispatcher asks the Factory Manager to recover the appropriate Media Factory and commands it to create a MCid capability.
- The Media Factory Creates the appropriate Media Object associated to the MCid capability.
- The Media Object executes whatever actions are necessary for instantiating the MCid capability and for initializing it.
- The Media Factory receives the unique id of the newly created Media Object (mid).
- The Media Factory registers the new Media Object into the Media Object Manager, where it shall be recovered later when commands arrive to that specific stub.

- The mid is propagated to the application following the appropriate path (i.e. marshaling and WebSocket transport).

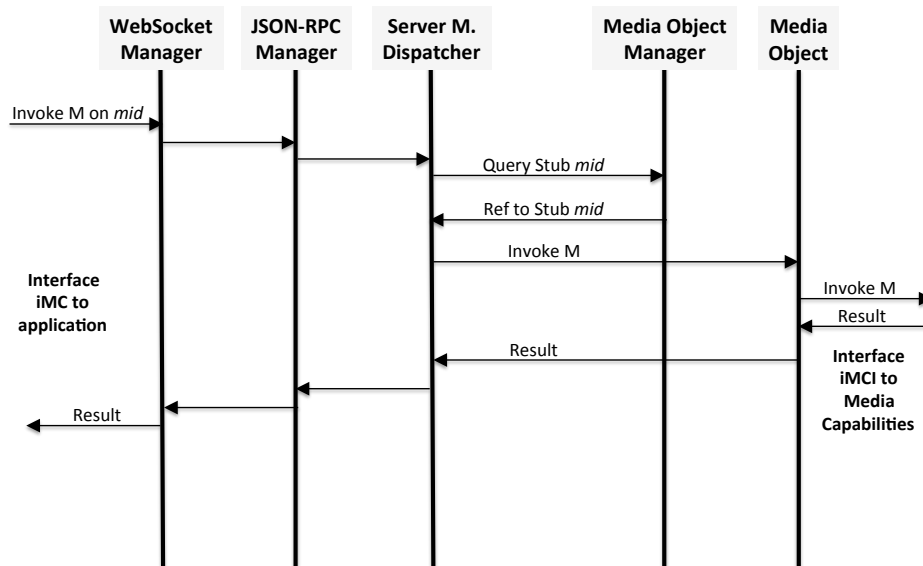


Figure 17. Flow diagram showing the interactions among the Media Server Manager modules for invoking a method (identified as *M*) into a specific Media Element (identified as *mid*)

Once a media capability (e.g. Media Element) has been created, it can be accessed and controlled following the scheme depicted in Figure 17. From top to bottom, the steps are the following:

- A request arrives from the application through interface iMC asking for invoking a specific primitive or method (identified as *M* in the diagram) onto the Media Element whose id is *mid*.
- The request is unmarshalled at the JSON-RPC module.
- The Dispatcher detects the request has invocation semantics and orchestrates the appropriate sequence for it.
- First, the Media Object Manager is queried for recovering the Media Object whose id is *mid*.
- Second, the invocation is launched to the specific Media Object holding that *mid*. That object knows how to reach the real Media Element and executes the appropriate actions through the iMCI interface for having the primitive executed.
- As a result, a return value may be generated. This return value is dispatched back to the application following the reverse path (i.e. marshaling and WebSocket transport).

4.3.7.2 Media Server main interactions

As introduced above, the main interactions between the Media Plane and the external world take place through three MS interfaces: iMC and iME for control and events and iM for media exchange.

Interactions through the iMC and iME interfaces take place through a request/response protocol based on JSON-RPC over WebSocket, as can be inferred by observing Figure 15. At the iMC interface, this protocol exposes the capability of negotiating media communications. The main entities and interactions associated to the iMC interface are the following:

Interacting with a Media Server: media negotiation and media exchange

In a media application involving a Media Server we identify three types of interacting entities:

- The Client Application: which involve the native media capabilities of the client platform plus the specific client-side application logic. The End-user in in contact with the Client Application.
- The Application Server: which is a container of procedures implementing the server-side application logic. The Application Server mediates between the Client Application and the Media Server in order to guarantee that the application requirements (e.g. security, auditability, dependability, etc.) are satisfied.
- The Media Server: which provides the media capabilities accessed by the Client Application.

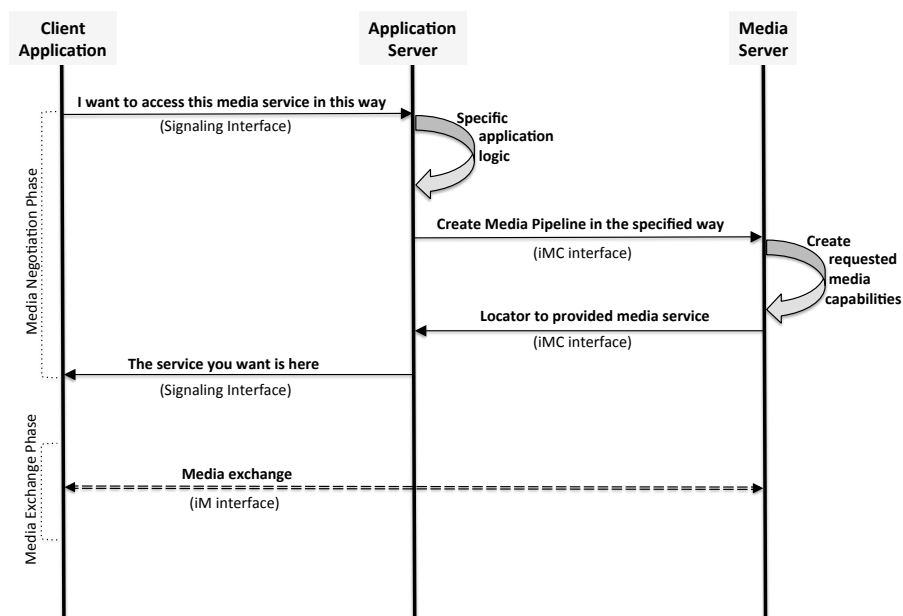


Figure 18. This diagram shows the typical entities and interactions involved in a media application based on a Media Server. In the case of NUBOMEDIA, the Application Server corresponds with the NUBOMEDIA PaaS component and, more specifically, with the container where the specific application logic is executing.

As shown in Figure 18, the interactions among these three entities typically occur in two phases: the media negotiation and the media exchange phase.

The media negotiation phase requires a signaling protocol that is used by the Client Application to issue messages to the Application Server requesting to access a specific media service. Typically those requests specify the specific media formats and transport mechanisms that the Client Application supports.

The Application Server provides semantics to these signaling messages by executing a server-side application logic. This logic can include procedures for Authentication, Authorization and Accounting (AAA), Call Detail Record (CDR) generation, resource provisioning and reservation, etc.

When the Application Server considers that the media service can be provided, and following the instructions provided by developers when creating the application logic, the Application Server commands the Media Server to instantiate and configure the appropriate media capabilities. Following the discussions in sections above, in the case

of NUBOMEDIA, this means that the Application Server creates a Media Pipeline holding and connecting the appropriate Media Element instances suitable for providing the required service.

To conclude, the Application Server receives from the Media Server the appropriate information suitable for identifying how and where the media service can be accessed. This information is forwarded to the Client Application as part of the signaling message answer.

Remark that during the above mentioned steps no media data is really exchanged. All interactions have the objective of negotiating the what's, how's, where's and when's of the media exchange.

The media exchange phase starts when both the Client Application and the Media Server are aware on the specific conditions in which the media needs to be encoded and transported. At this point, and depending on the negotiated scheme (i.e simplex, full-duplex, etc.) the Client Application and the Media Server start sending media data to the other end, where it is in turn received. The Media Server processes the received media data following the specific Media Pipeline topology created during the negotiation phase and may generate, as a result, a processed media stream to be send back to Client Applications.

Interactions with WebRTC clients

A specific relevant case for NUBOMEDIA happens when dealing with WebRTC [WEBRTC] clients. In this case, the negotiation phase takes place through the exchange of Session Description Protocol [RFC4566] messages that are send from the Client Application to the Application Server through the signaling protocol and which are forwarded then to the Media Server instance through the iMC interface.

For WebRTC to be supported, a specific Media Element needs to be created supporting the WebRTC protocol suite. Following the accepted WebRTC naming conventions [ALVESTRAND], lets imagine that we call *WebRtcEndpoint* to such Media Element. In that case, the *WebRtcEndpoint* needs to provide SDP negotiation capabilities through the iMC interface as well as any other of the WebRTC features required for the negotiation (e.g. [TRICKLE ICE], etc.)

As a result of this negotiation, an SDP answer shall be generated from the MS *WebRtcEndpoint* to the Application Server. This will be forwarded them back to the Client Application that, in turn, can use it for starting the media exchange phase. This procedure is summarized in Figure 19.

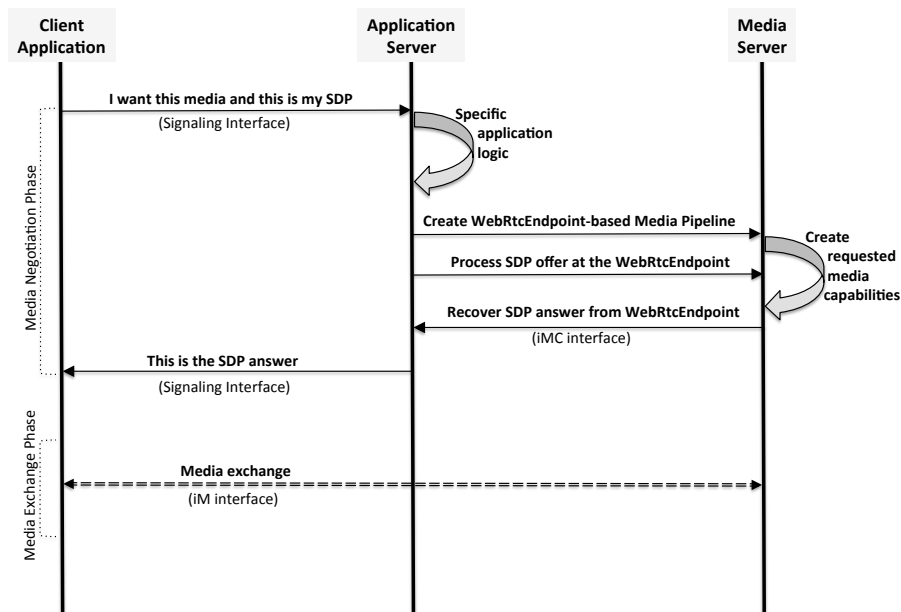


Figure 19. Flow diagram schematizing the different interactions taking place for establishing a WebRTC session between the Client Application and the Media Server.

4.3.7.3 The NUBOMEDIA Media Repository

The Cloud Repository provides scalable archiving capabilities to NUBOMEDIA. The Cloud Repository is able to store Repository Items, which comprise multimedia data and metadata. Repository Items can be recovered later querying them through ID or through any kind of information present into its metadata. Repository Item metadata consists on arbitrary JSON, which is quite convenient for developers. The components of the Cloud Repository can be seen in Figure 20 and are the following:

- The Cloud Repository Manager, which is responsible of orchestrating the repository behavior and of providing semantics to the Cloud Repository API that it exposes. This API is based on REST and exposes simple CRUD operations for creating, recovering, updating or deleting Repository Items. When a new Repository Item is created, the Repository Manager enables a unique URL at the HTTP Repository Connector where the multimedia data of the item can be send through HTTP POST. When a Repository Item is queried for reading, the Repository Manager offers a unique URL at the HTTP Repository Connector for receiving it through an HTTP GET.
- The HTTP Repository Connector provides the ability of communicating multimedia information with the external world through the HTTP protocol. Media ingesting takes place through HTTP POST requests, which support Chunked and multi-part encoding. Media is send out of the repository through HTTP GET requests. Every time a media exchange is to take place, the client needs to command the repository through the Cloud Repository API and the Cloud Repository Manager commands this connector for perform the operation in a unique and on-the-flight created URL.
- The Metadata Manager is module providing metadata query management capabilities, transforming the query requests received through the repository API into native queries of the Blob Database.

The Blob Database is a persistent elastic storage where Repository Items are stored. For this, it needs to provide the ability of recording binary blobs of arbitrary size and structured JSON metadata.

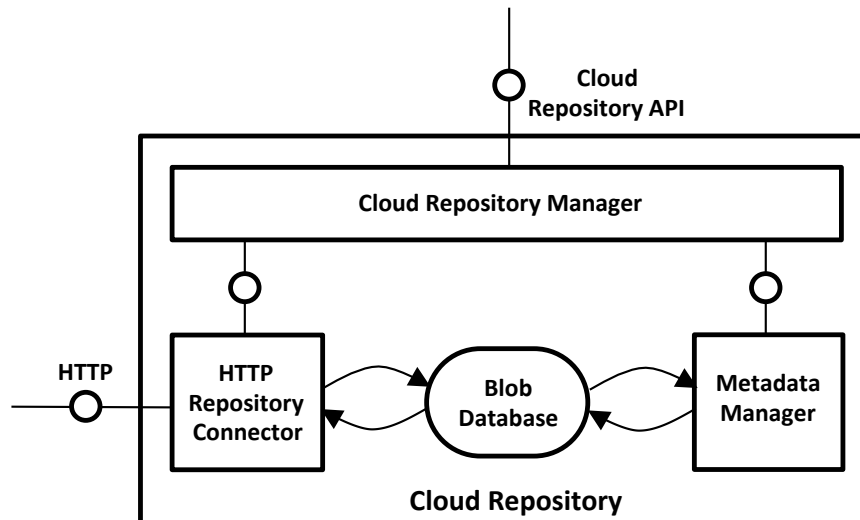


Figure 20. Cloud Repository architecture

4.3.7.4 Cloud Repository main interactions

The Cloud Repository main interactions take place through the Cloud Repository API. There are two types of main interactions: recording items into the repository and playing items from the repository. We analyze them separately.

The flow diagram for recording a new item into the repository is depicted in Figure 21. At it can be observed, it gets through three steps:

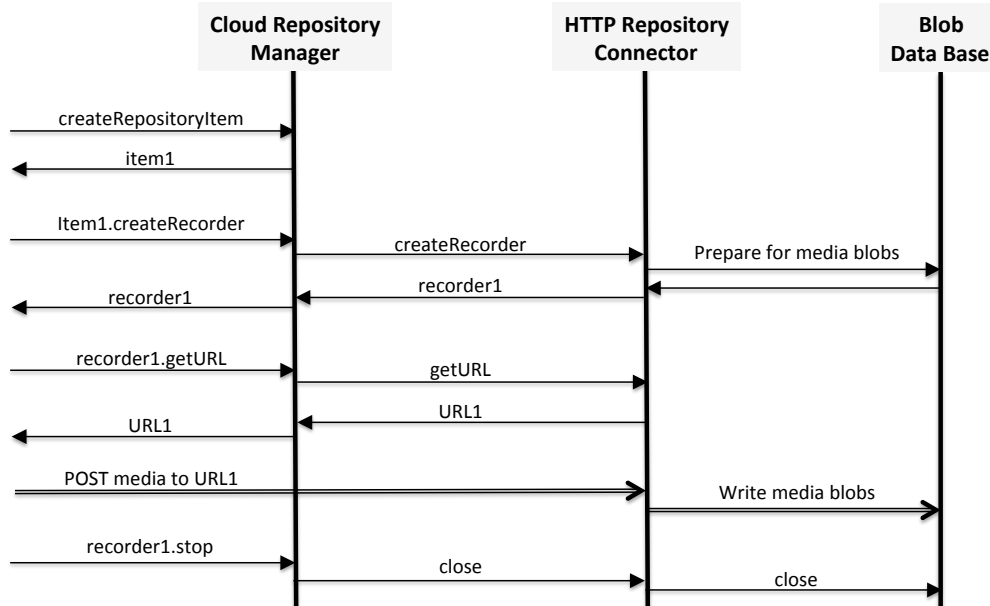


Figure 21. Main interactions for recording an item into the Cloud Media Repository.

- First, the client application uses the Cloud Repository API for creating a new Repository Item. Upon conclusion, the operation returns the unique id of the corresponding item.
- Second, the application developer can create a new recorder associated to this item. This recorder requires a number of resources in the HTTP Repository Connector, which needs to prepare for receiving the media stream and writing it to the Blob Database.

- Third, the media transfer takes place from the application to the HTTP Repository Connector using HTTP POST messages. The connector then slices the media into the appropriate blobs, which are written into the Blob Database.

The flow diagram for playing pre-recorded repository item streams is shown in Figure 22. As it can be observed, it also goes through 3 steps.

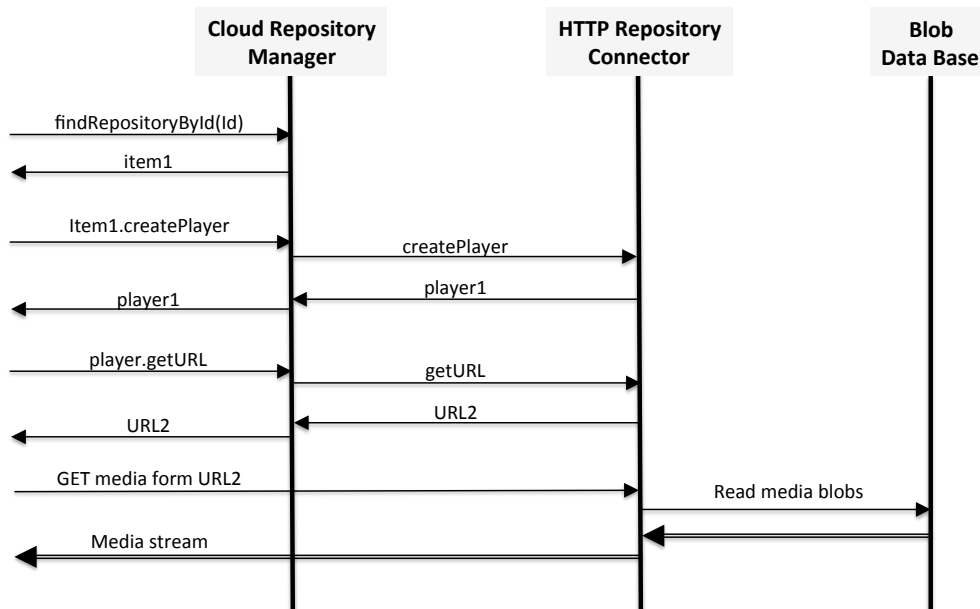


Figure 22. Main interactions for playing an item from the Cloud Media Repository.

- First, the application developer needs to obtain the id of the repository item. This can be done in different ways including querying for items whose metadata complies with a given specific pattern. Once the id has been obtained, we can recover a reference to it.
- With such reference, it is possible to create a player. The player has associated a unique URL where the media stream can be read from the HTTP Repository Connector.
- To conclude, the application can recover the media stream using an HTTP GET transaction. Upon reception of the GET request, the HTTP Repository Connector performs the appropriate operations for reading the blobs associated to the repository database in the appropriate order.

4.3.8 NUBOMEDIA IaaS

NUBOMEDIA PaaS hides the underlying complexity of an IaaS platform that enables developers to build multimedia applications without infrastructure knowledge. The elastic nature of the IaaS enables developers to build multimedia apps that scale easily with an increase of usage. IaaS capabilities are exposed through APIs that are used by the Virtual Infrastructure Manager to provide hardware resources to the upper levels of the NUBOMEDIA IaaS. Resources provided by the NUBOMEDIA IaaS are the following:

- APIs
- Core services
 - Management console
 - Compute capabilities

- Network capabilities
- Storage capabilities
- Image service
- Additional capabilities
 - Metric system
 - Alerting system
 - Centralized logging system
 - Object and Block Storage

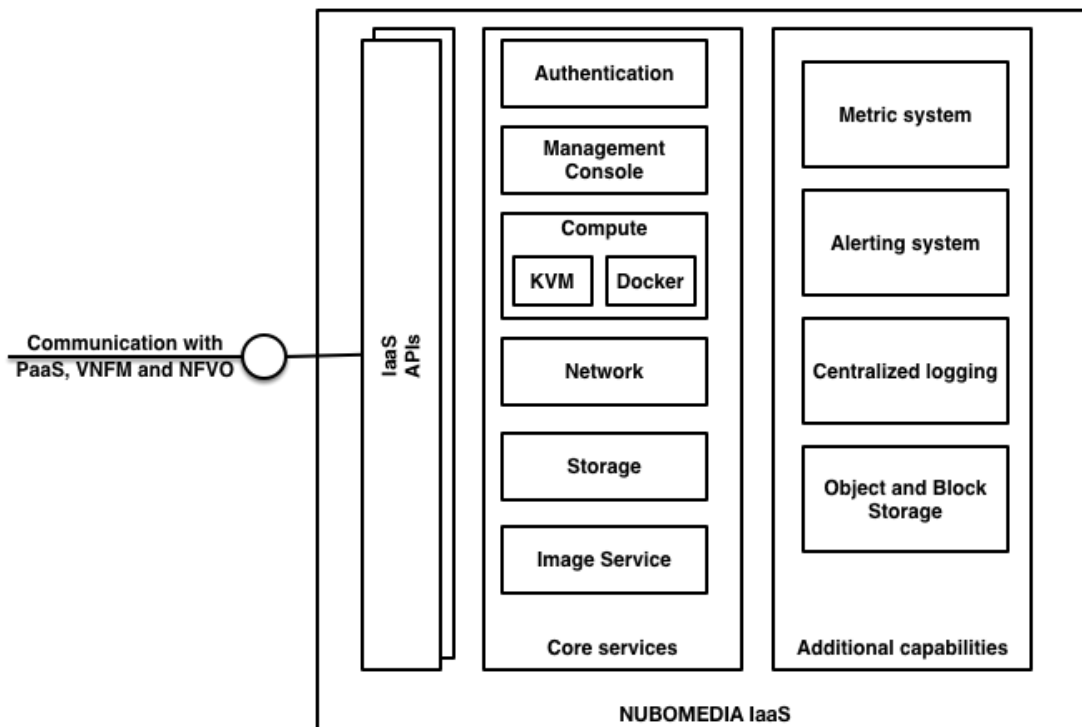


Figure 23 Functional architecture of NUBOMEDIA IaaS

4.3.8.1 IaaS APIs

Using the NUBOMEDIA IaaS APIs you can launch server instances, create images, manage networking resources, assign metadata to instances and images, create containers, push metrics to the monitoring system, forward all logs to a centralized logging system where it can be further analyzed, see the health status of the software and hardware infrastructure, and complete other actions in NUBOMEDIA cloud. Using REST APIs the Virtual Infrastructure Manager can request hardware resources from IaaS that are consumed by the upper levels of the NUBOMEDIA PaaS.

4.3.8.2 Core services

Authentication: Interaction of the NUBOMEDIA components with the IaaS are executed through users. The IaaS user is assigned to a tenant which is the same to a project and represents a set of resource allocation in the IaaS. Every user must authenticate with either user and password or with token in order to access the system. API requests are usually made through tokens that have expire date and can be reissued on demand. Once authenticated, the role assigned to the user determines what he is authorized to do.

Management Console: The NUBOMEDIA's IaaS management console provides a web based user interface to all the IaaS components allowing the NUBOMEDIA platform administrator to manage it's components and check the status of the platform.

Compute: This is the main part of an IaaS system. It is designed to manage and automate pools of computer resources and works with both KVM and Docker as hypervisors. It is written in Python and uses many external libraries such as Eventlet (for concurrent programming), Kombu (for AMQP communication), and SQLAlchemy (for database access). Compute's architecture is designed to scale horizontally on standard hardware with no proprietary hardware or software requirements and provide the ability to integrate with legacy systems and third-party technologies.

Network: The network service is made to manage networks and IP addresses inside the IaaS. It ensures the network is not a bottleneck or limiting factor in the NUBOMEDIA cloud, and gives users self-service ability, even over network configurations, allowing the platform administrator to configure the firewall for NUBOMEDIA instance in order to allow traffic only to relevant ports.

Storage: The block storage provides persistent block-level storage devices to be used on compute instances as additional disk space. The block storage system manages the creation, attaching and detaching of the block devices for both KVM and docker instances. Block storage volumes are fully integrated into the compute service and the management console allowing the platform administrator to easy manage the storage needs. Snapshot management provides powerful functionality for backing up instances on block storage volumes. Snapshots can be restored or used to create a new instance or block storage volume.

Image Service: The image service provides discovery, registration, and delivery services for disk server images and container images. Stored images can be used as a template.

4.3.8.3 Additional capabilities

Metric System: The metric system gathers statistics about the system as it is running and stores this information. Those statistics can then be used to find current performance bottlenecks (i.e. performance analysis) and provide this information to the VNFM through its API so it can take decisions whether to scale up or down resources inside the NUBOMEDIA platform.

Alerting system: It monitors the entire NUBOMEDIA infrastructure from physical components to service health to ensure everything is functioning properly. In the event of a failure, the alarm system can alert technical staff of the problem, allowing them to take actions fast, before it affects end-users, or customers.

Centralized logging system: The logging system is a tool for keeping all the Docker instances events and logs centralized, so they can be further accessed by the platform manager or even by the NUBOMEDIA user. We use it to collect logs, parse them, and store them for later use. All the logs are stored in a time series database in order to allow fast searching on them. The system also provides an interface where you can view and analyze them in a flexible analytics and real-time way.

Object and Block Storage: The object and block storage system consists on a NoSQL database which is configured to take advantage of load balancing and data replication features over multiple machines for storing files.

5 NUBOMEDIA Development APIs

As specified in sections above, from the developer's perspective, NUBOMEDIA capabilities are accessed through a set of APIs inspired in the three tier development model, as depicted in Figure 1. Hence, for creating NUBOMEDIA enabled applications developers just need to understand these NUBOMEDIA Development APIs so that they may use them for creating their rich RTC media applications. The NUBOMEDIA Development API stack is architected following the scheme depicted in Figure 24. As it can be observed, this stack offers a complete set of APIs that can be classified in three groups that we call: Media Capabilities APIs, Signaling APIs and Abstract Communication APIs.

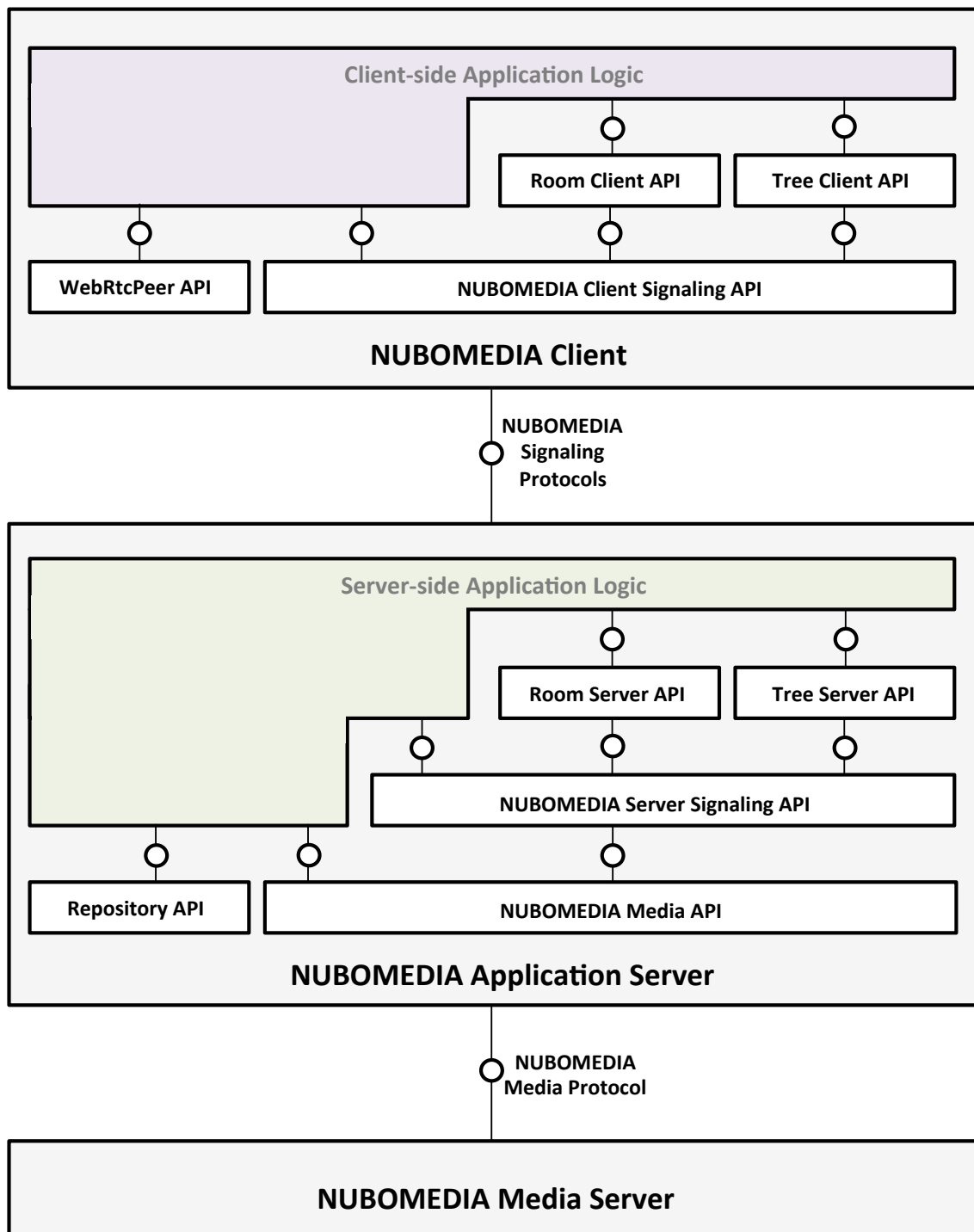


Figure 24. Architectural diagram showing the NUBOMEDIA API stack which comprises three types of APIs: Media Capability APIs (NUBOMEDIA Media and Repository APIs), Signaling APIs (JSON-RPC over

WebSocket client and server APIs), and Abstract Communication APIs (Room and Tree client and server APIs). In this picture, the NUBOMEDIA Media Protocol corresponds to the iMC and iME interfaces described in Figure 11, while the NUBOMEDIA Signaling Protocols correspond to the iS interface described in Figure 2.

5.1.1 Media Capabilities APIs

These APIs expose to the application logic the low-level media capabilities of NUBOMEDIA. These capabilities are basically the ones offered by the NUBOMEDIA Media Server and the NUBOMEDIA Media Repository, as described in sections above. As a result, this group includes two APIs:

- The NUBOMEDIA Media API, which enables developers consuming the NUBOMEDIA Media Server capabilities among which we can find media transport, media archiving, media processing, media transcoding, etc. This API is based on two main concepts: Media Elements and Media Pipelines. This API is shipped for developers as an SDK abstracting the iMC and iME interfaces specified in Figure 11.
- The NUBOMEDIA Repository API, which makes possible to access an elastic scalable media repository for archiving media information and meta-information. It is based on the notion of Repository Item: an object with a unique identity which may contain media data and meta-data. This API is shipped for developers as an SDK abstracting the REST Cloud Repository API specified in Figure 20.

In addition, in this group we also include a NUBOMEDIA API abstracting the client side media capabilities for developers. In current state-of-the-art, these capabilities basically correspond to WebRTC and comprise both the ability of capturing and rendering media plus the ability of communicating RTC media. In NUBOMEDIA, we have abstracted all these capabilities behind the WebRtcPeer API.

5.1.2 Signaling APIs

The Media Capabilities APIs introduced above are signaling agnostic, meaning that they neither require nor assume any kind of specific characteristic for signaling. Hence, NUBOMEDIA capabilities can be accessed through any kind of signaling protocol including SIP, XMPP or REST. However, for simplifying common development tasks, in our architecture we propose a simple signaling protocol based on JSON-RPCs that is suitable for most applications not requiring specific interoperability features. This protocol has been wrapped through an abstract API which enables full-duplex signaling exchanges between Application Server and Clients. To this aim, the API is split into two complementary parts:

- The NUBOMEDIA Client Signaling API. This is a client-side API enabling the creation of JSON-RPC sessions and the sending and receiving of messages through them.
- The NUBOMEDIA Server Signaling API. This is a server-side API enabling the creation of JSON-RPC sessions and the implementation of server-side business logic for messages following a request/response communication scheme.

5.1.3 Abstract Communication APIs

Developers typically use RTC media capabilities for creating applications devoted to person-to-person communications. In this case, the application business logic needs to manage the communication topologies among participants in an RTC multimedia session. In the general case, this logic needs to be programmed by the application developers. However, there are a number of communication topologies that are quite

popular and appear systematically in applications. Due to this, in our architecture, we propose specific communication API abstracting the low level details of the media logic on these topologies so that developers may use them in an agile and efficient manner. In particular, we have identified two of these common topologies: the Room Topology and the Tree Topology. For them two specific APIs have been implemented:

- The Room API. This API has two complementary sides, the Room Server and the Room Client APIs. Through them, this API makes possible for developers to create applications basing on Room Topologies. Room topologies are based on the intuitive idea of a physical room where people may communicate. If you enter into the room you may talk to others and others may talk to you. Hence, the main characteristic of a Room Topology is that the multimedia session enables a full-duplex communication exchange of media among all participants. Hence, the session is called informally room and all room members may publish their media streams to the room or subscribe to the streams of the rest of room members.
- The Tree API. Like the Room API, it also has Server and Client sides. However, this API is based on a Tree Topology. In our context, a tree can be seen as a one-way (i.e. simplex) mechanism providing one-to-many RTC media. The interesting aspect of this topology is that the number of leafs of the tree may be large, which makes convenient for providing real-time media broadcasting capabilities to large audiences.

The NUBOMEDIA API stack architecture above mentioned is the result of a complex design process where we have prosecuted to enable simplicity without scarifying flexibility and expressiveness. This stack has been created for abstraction, understood as the ability of making simple things simple and complex things possible for developers. For this, the rationale behind its design is based on a number of principles application developers need to understand in order to choose the appropriate APIs for each implementation objective.

In NUBOMEDIA, the most abstract APIs are the Room and Tree APIs. These are abstract because they hide most of the low level details of the media complexities and expose to developers high-level notions such as “Rooms” and “Trees”, that are logical objects suitable for managing media transport through specific communication topologies. The NUBOMEDIA internal use-cases analysis (see NUBOMEDIA Project Deliverable D2.1.2), as well as the accumulated experience of developers worldwide out of NUBOMEDIA evidence that Room and Tree topologies are at the base of a significant fraction of RTC media services. Hence, developers wishing to create applications just providing room group communications or tree one-to-many media broadcasting services can use these APIs directly without requiring any further understanding on the rest of the API stack.

However, there is still a fraction of developers for which the Room and Tree APIs do not fit. These may include the ones with specific interoperability requirements (e.g. SIP or legacy RTP) or needing special features (e.g. custom media processing, non-common communication topologies, non-linear media logic, etc.) In that case, lower-level NUBOMEDIA APIs might be needed.

The NUBOMEDIA Signaling APIs makes possible for developers to create custom signaling protocols in a seamless and efficient way. Hence, this API might be useful whenever specific signaling mechanisms beyond rooms and trees are required. Just for illustration, let's imagine a video surveillance application with the ability of detecting

intruders in a specific area of the camera viewport. Whenever an intruder is detected an alarm needs to be fired to all the connected clients, so that they may rewind the streams and visualize it again for assessing the severity of the incident.

Clearly, this type of logic cannot be provided through the Room or Tree APIs, which do not have alarm-sending capabilities nor the ability of seeking the media for a playback. Hence, a developers creating this application needs a custom signaling protocols. This developer may use the signaling protocol she wishes given the PaaS nature of NUBOMEDIA. However, among the available options, a natural choice is the NUBOMEDIA Signaling API. This API makes possible to create a custom signaling protocol just by defining some simple JSON-RPC messages, which is quite convenient given the familiarity of developers with this format. Once this is done, the API makes straightforward to send such messages and to implement the appropriate business logic upon their reception. The only limitation of this scheme is that the API mandates the protocol to be based upon JSON-RPC over WebSocket transport. Hence, whenever this restriction is not an impediment, the NUBOMEDIA Signaling API shall be useful for creating specific and customized signaling mechanisms.

The NUBOMEDIA Media API, in turn, exposes the low level media capabilities. Through this API developers can create arbitrary and dynamic topologies combining them with media processing, transcoding or archiving. The Room and Tree topologies are just particular cases of what this API can provide. Hence, mastering this API is a must for all developers wishing to take advantage of all NUBOMEDIA features. This API is complemented with the Repository API which enables media data and metadata persistence onto an elastic scalable repositories. Hence, the Repository API may be of help whenever large amounts of media information need to be recorded and recovered.

All in all, and as Figure 24 shows, developers are free to combine the NUBOMEDIA APIs without restrictions so that for example, an application may consume at the same time the Room API, the Signaling API and the Media API if it is needed.

References

[ALVESTRAND] <https://tools.ietf.org/html/draft-ietf-rtcweb-overview-14>

[ETSI_WP] ETSI. Network functions virtualisation - introductory white paper, October 2012. At the SDN and OpenFlow World Congress, Darmstadt Germany.

[FMC] <http://www.f-m-c.org>

[MANO] Network Function Virtualization Management and Orchestration. (2014). Retrieved December 14, 2015 from http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf

[RFC4566] <https://tools.ietf.org/html/rfc4566>

[RFC2119] Key words for use in RFCs to Indicate Requirement Levels. <https://www.ietf.org/rfc/rfc2119.txt>

[RFC2616] Hypertext Transfer Protocol -- HTTP/1.1. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>

[TRICKLE ICE] <https://tools.ietf.org/html/draft-ietf-mmusic-trickle-ice-02>

[WEBRTC] <http://www.webrtc.org>