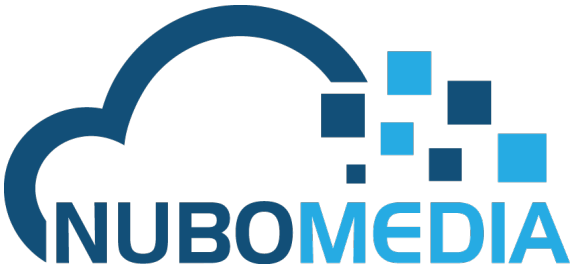
D3.3	
Version	1.0
Author	TUB
Dissemination	PU
Date	30/11/2016
Status	Final



D3.3 Cloud Platform v3

Project acronym:	NUBOMEDIA
Project title:	NUBOMEDIA: an elastic Platform as a Service (PaaS) cloud for interactive social multimedia
Project duration:	2014-02-01 to 2017-01-30
Project type:	STREP
Project reference:	610576
Project web page:	http://www.nubomedia.eu
Work package	WP3 Cloud Platform
WP leader	Giuseppe Carella
Deliverable nature:	Prototype
Lead editor:	Giuseppe Carella
Planned delivery date	11/2016
Actual delivery date	30/11/2016
Keywords	Elasticity, Cloud Computing, NFV, SDN, Management and Orchestration, Monitoring

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

Contributors:

Alin Calinciuc (USV)
Giuseppe Carella (TUB)
Alice Chaembe (FOKUS)
Flavio Murgia (FOKUS)
Luis Lopez (URJC)
Michael Pauls (TUB)
Cristian Spoiala (USV)
Lorenzo Tomasini (TUB)

Internal Reviewer(s):

Michael Pauls (TUB)

Version History

Version	Date	Authors	Comments
0.1	09.2016	Giuseppe Carella	Initial version
0.2	09.2016	Giuseppe Carella	Added content in some section
0.3	10.2016	Giuseppe Carella, Michael Pauls, Lorenzo Tomasini, Flavio Murgia, Alice Cheambe	Added content in most of the sections
0.4	11.2016	Giuseppe Carella, Michael Pauls	Added evaluation section
1.0	11.2016	Giuseppe Carella	Final version

Table of contents

List of Figures:

Acronyms and abbreviations:

API	**Application Programming Interface**
AS	Application Server (refers to function instance)
ASS	Application Server Services
CFM	Cloud Functions Manager
CMA	Connectivity Manager Agent
CN	Compute Node
CPU	Central Processing Unit
EMS	Element Management System
IaaS	Infrastructure as a Service
JSON	JavaScript Object Notation
ME	Media Element
ML2	Multi Layer 2
MP	Media Pipeline
MS	Media Server (refers to function instance)
NAI	NUBOMEDIA Autonomous Installer
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
PNF	Physical Network Function
QoS	Quality of Services
RFC	Request For Comments
RTC	Real-Time Communications
SaaS	Software as a Service
SLA	Service Level Agreement
UE	User Equipment
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNFM	VNF Manager
VNFR	Virtual Network Function Record
VXLAN	Virtual Extensible LAN
WS	WebSocket
WWW	World Wide Web

1 Executive summary

This document stands as a public report including contributions from Release 7,8, and 9 and constitutes Deliverable 3.3, providing the final details about the Cloud Platform components and their integration from WP3 perspective. It reports some of the changes made on the Software artifacts delivered out of WP3 for supporting new requirements coming from the NUBOMEDIA developer community. Most of the requirements listed in D2.3, have been analyzed by each WP3 and a solution has been provided for most of them.

To summarize, the final version of the NUBOMEDIA Cloud Platform has been composed by the following components:

- A Platform as a Service (PaaS) allowing the deployment and provisioning of several type of Applications and supporting services. It allows developers to benefit from scalability and elasticity mechanisms provided at each level (data and signaling plane).
- An ETSI NFV [1] compliant framework for the Management and Orchestration of Multimedia Network Functions have been designed and implemented. This framework allows the instantiation of multiple multimedia Slices on the same Physical Infrastructure.
- A Connectivity Manager engine able to provide different level of QoS to deployed applications.
- A powerful monitoring system capable of providing metrics from each level, Application, Media Elements, and Infrastructure have been designed. Several existing monitoring solutions have been evaluated, and one of them has been selected and integrated in the Cloud Platform.

Overall, this work-package has implemented and integrated together numerous components, offering a rich set of features to the Application developers providing the foundation for a PaaS of Multimedia Applications. Each component has been implemented using an Agile development methodology. Testing driven development allowed a smooth development process. Using standardized interfaces reduced the complexity of integration among all components.

2 The NUBOMEDIA functional Architecture

As already mentioned in D3.2, the main goal of WP3 is to design and develop a Cloud Platform supporting the on demand deployment of Multimedia Applications. In this section will be briefly give a recap of the NUBOMEDIA Architecture as already presented in D2.4.2 [2] and D2.4.3[2], and will be presented the changes made during the last releases of the platform for supporting the requirements coming from Application developers.

2.1 Functional Architecture Overview

From 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 Infrastructure as a Services (IaaS) composed by the infrastructure resources in terms of Compute Nodes (CN) and the Virtual Infrastructure Manager (VIM) providing the compute, storage and networking resources to the upper layers

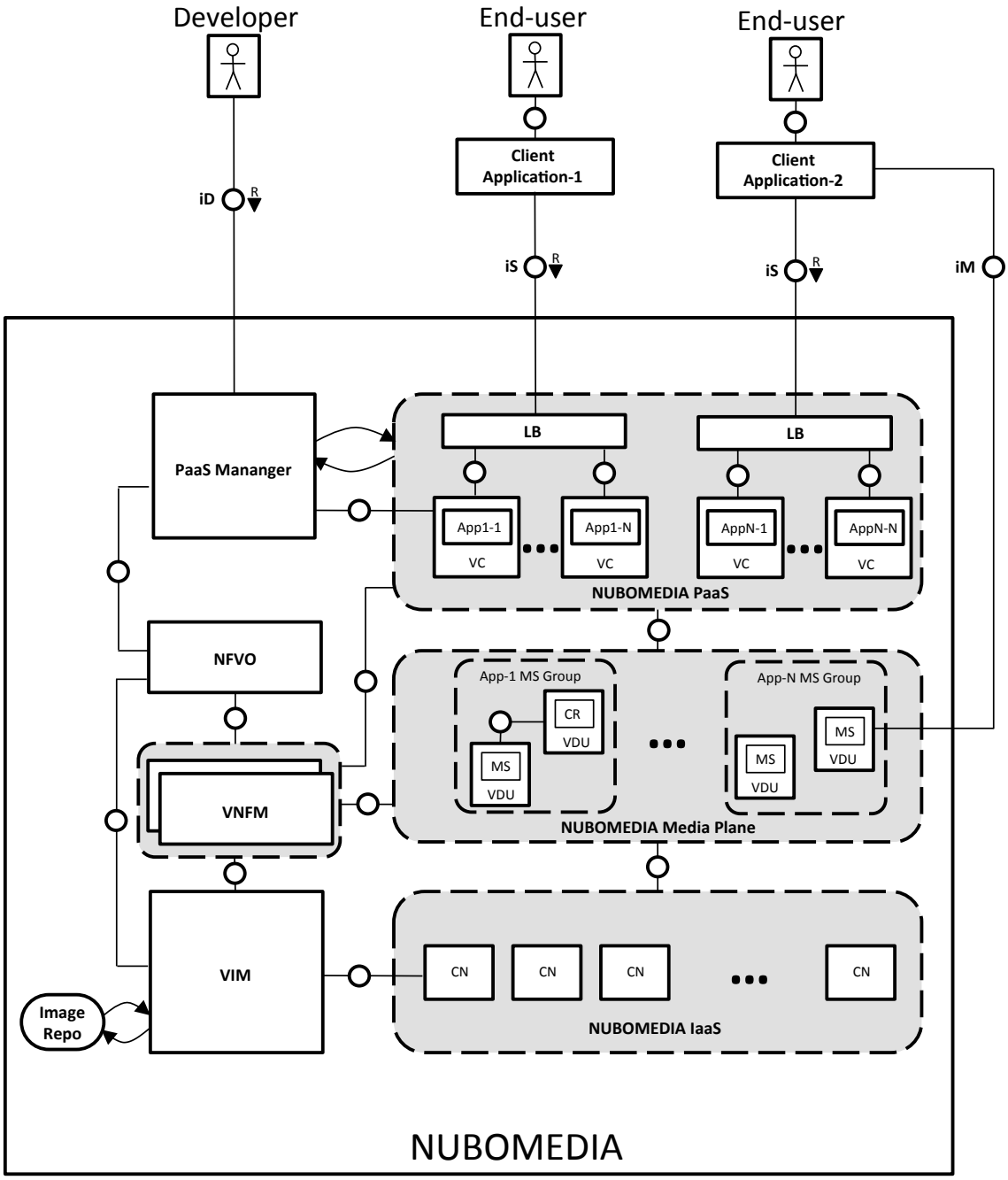


Figure 1. NUBOMEDIA Functional Architecture

In particular, WP3 focuses on the Management and Orchestration functionalities at the three different layers, while Media Functions and Applications are out of scope of WP3 and will be described in other deliverables.

During the last period currently under review, there have been no major changes to the NUBOMEDIA PaaS Architecture. Only some additional features have been introduced in order to support requirements from application developers. Those features are documented further in the following sections.

3 Requirement analysis

In this section it will be summarized the list of requirements taken from D2.3.3 which has been considered in the last year of the project driving the development of new features. Some of them have been implemented, while some others have been rejected either because of the large amount of work needed for supporting them or because an alternative solution was already available.
In particular, starting from the list of installation requirements, we had:

- IST_R1: Enable full installation of NUBOMEDIA on top of virtual computing resources. This requirement has been considered in the context of the Autonomous Installer and further described in Section 4.5.1.

From the list of debugging requirements:

- DBG_R1: Access to the environment where the application/media server is executing. This requirement has been rejected as similar to requirement DRT_R3.
- DBG_R2: Simplify media server logs has been considered and implemented, so now all developers have all the media-server logs available on the Debug section of their application on the PaaS GUI. For more details see Section 4.5.2.
- DBG_R3: Graphical tools for filtering the logs. This requirement has been considered in the context of the NUBOMEDIA Monitoring tools and further described in Section 4.5.2.

From the list of deployment & runtime settings requirements:

- DRT_R1: Control Specific KMS Instance. This requirement has been considered in the context of the NUBOMEDIA Media Plane and further described in Section 4.3.
- DRT_R2: Dynamically scale out limit settings. This requirement has been considered in the context of the NUBOMEDIA Media Plane, however it has not been considered for implementation as required several changes on many components. Further information is given in Section 4.3.
- DRT_R3: SSH into media server instances. This requirement has been considered in the context of the NUBOMEDIA PaaS and NUBOMEDIA and further described in Section 4.4.3.
- DRT_R4: Deploy additional services together with an application. This requirement has been considered in the context of the NUBOMEDIA PaaS and further described in Section 4.4.
- DRT_R5: Deploy Apps of any language. This requirement has been considered in the context of the NUBOMEDIA PaaS, however no further actions has been taken as the NUBOMEDIA PaaS support by default several programming languages being based on container technologies.
- DRT_R6: Pool size runtime settings. This requirement has been considered in the context of the NUBOMEDIA Media Plane and further described in Section 4.3.
- DRT_R7: Display App deployment time. This requirement has been considered in the context of the NUBOMEDIA PaaS and further described in Section 4.4.
- DRT_R8: Restart application. This requirement has been considered in the context of the NUBOMEDIA PaaS and further described in Section 4.4.
- DRT_R9: Redeploy Apps. This requirement has been considered in the context of the NUBOMEDIA PaaS and further described in Section 4.4.
- DRT_R10: KMS Buildpacks for custom module installation. This requirement has been on our backlog but hasn't been moved into selected for development

because we didn't have any application needing this kind of feature in the context of NUBOMEDIA.

- DRT_R11: Provide billing capabilities. This requirement has been considered in the context of the NUBOMEDIA PaaS, however no further actions have been taken as it would require the implementation of several additional technologies. It may be considered for future releases of the platform.
- DRT_R12: Provide fault tolerance capabilities. This requirement has been considered in the context of the NUBOMEDIA Media Plane, however no further actions have been taken as it was not required by the NUBOMEDIA Demonstrators. Anyway, an analysis of a possible implementation solution has been given in Section 4.3.
- DRT_R13: Geo-aware multi-site. This requirement has been considered in the context of the NUBOMEDIA Media Plane. Even though the implementation may have been straight forward as Open Baton provides multi-site deployment capabilities, it has not been satisfied as at the moment NUBOMEDIA is implemented in a single testbed location.

From the list of PaaS GUI requirements:

- PGUI_R1: default autoscaling behavior and units are obscure and not specified.
- PGUI_R2: scaleInOut parameter name in PaaS GUI is obscure.
- PGUI_R3: coolDownPeriod should also be configurable parameter in the PaaS GUI in relation to autoscaling behavior.
- PGUI_R4: scaleInThreshold should also be configurable parameter in the PaaS GUI in relation to autoscaling behavior.
- PGUI_R5: PaaS GUI does not specify scaling parameters of a deployed application.
- PGUI_R6: Monitor machines memory, media pipelines and media elements of a media-server.
- PGUI_R7: Waiting notification in PaaS GUI.
- PGUI_R8: Deployment status should be shown in application tab.

All those requirements have been considered in the context of the NUBOMEDIA PaaS component. In particular, the PaaS GUI has been redesigned in order to simplify the interaction between the Application developers and the NUBOMEDIA platform. Also the guide on how the autoscaling mechanism works has been improved and it's not part of the NUBOMEDIA developer guide[1]. More details are provided in the NUBOMEDIA PaaS Section 4.4.

[1] http://nubomedia.readthedocs.io/en/latest/paas/autoscaling/

4 Software Architecture

Considering that during release 7,8, and 9, there were no extensions on the NUBOMEDIA Functional Architecture, there were no changes on the NUBOMEDIA Software Architecture. In this section it will be briefly given a short summary of the Software Architecture as presented already in D3.2, and it will be given a description about the extensions which were made to the NUBOMEDIA Software components for supporting requirements provided by Application developers.

4.1 Software Architecture Overview

The NUBOMEDIA Software Architecture has been based on the distributed NUBOMEDIA Functional Architecture. Each Software Components has been mapped to a different functional element designed by the Functional Architecture. In particular, each of the layer of the Functional Architecture consists of a mix between existing open source activities, extensions to those existing tools, and completely newly implemented ones. Figure 2 shows the Software Architecture diagram.

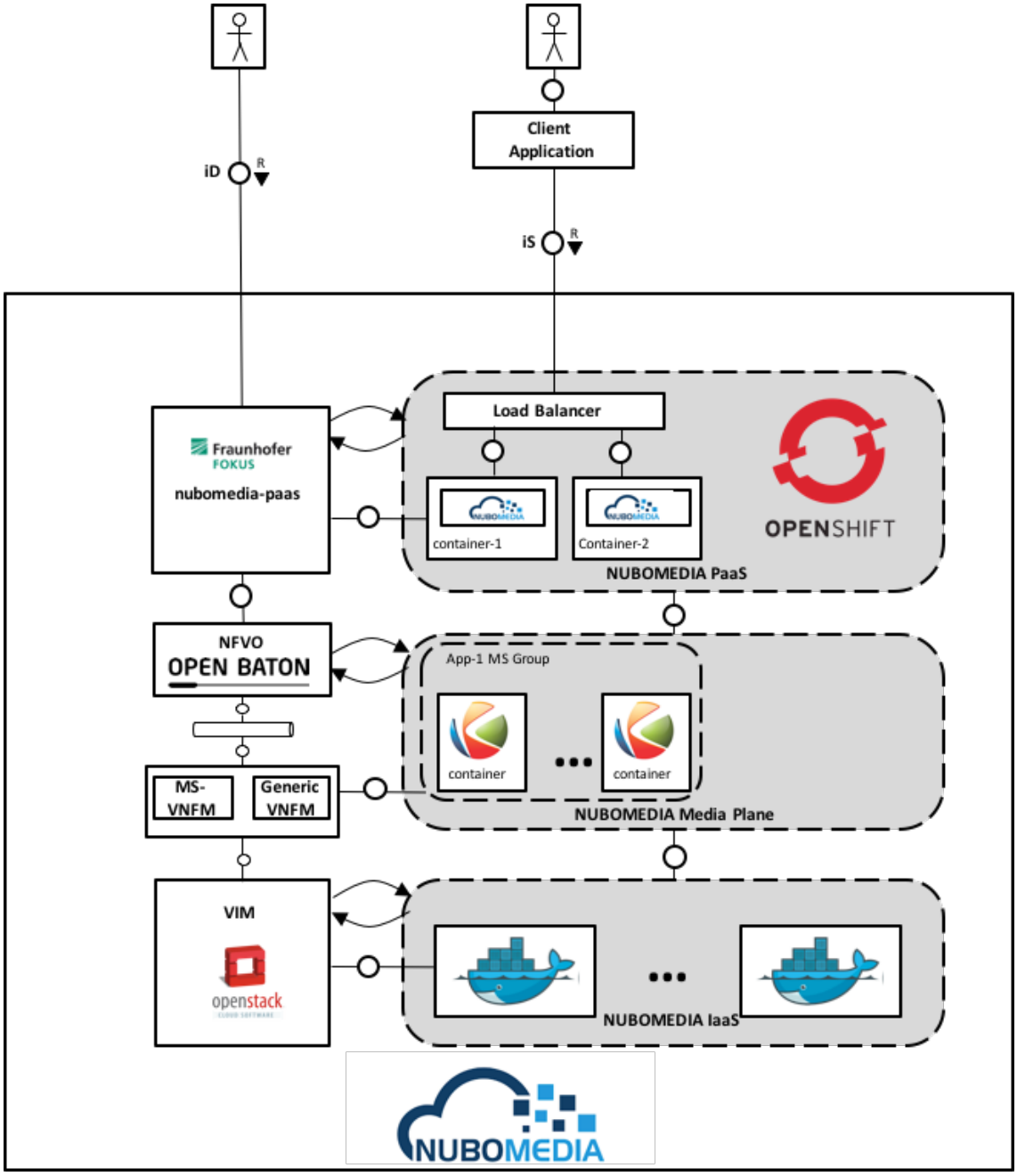


Figure 2. NUBOMEDIA Software Architecture

In particular, as per D3.2 description:

- The NUBOMEDIA IaaS has been composed mainly by OpenStack Services extended for fully supporting instantiation of docker containers on distributed Compute Nodes. Additionally, several monitoring and logging systems have been integrated into the OpenStack platform. No changes were made during Release 7, 8 and 9, and the OpenStack Kilo version has been mainly maintained with bugfixes.
- For the Media Plane control elements, Open Baton[3] has been selected and further extended as the only fully compliant NFV Orchestrator (NFVO) [5].In order to manage the Media Server a new VNFM (MS-VNFM) have been implemented, which extends the functionalities provided in previous releases by the EMM. While for the Cloud Repositories has been employed the Generic-VNFM, and a set of scripts have been implemented in order to control its lifecycle. Several changes have been required for supporting new requirements during Release 7,8, and 9. Those changes are documented in upcoming sections.

- The PaaS layer has been implemented with an existing open source component, OpenShift (add link), and a newly implemented software artifacts named nubomedia-paas mapped on top of the PaaS-API and PaaS-Manager functional element. Several changes have been required for supporting new requirements during Release 7,8, and 9. Those changes are documented in upcoming sections.

As a summary, the final Release of the NUBOMEDIA Cloud Platform has been composed by the following components and respective versions as listed in the table below.

Software Component	Functional Element	Version
OpenStack + nova-docker driver	Virtual Infrastructure Manager and NFV Infrastructure	Kilo
Open Baton	NFV Orchestrator	3.0.0
MS-VNFM	Media Server VNF Manager	3.0.0
Connectivity-Manager	Connectivity Manager	1.0.0
Connectivity-Manager-Agent	Connectivity Manager Agent	1.0.0
OpenShift	PaaS	
NUBOMEDIA PaaS	NUBOMEDIA PaaS	1.3.4
NUBOMEDIA Marketplace	Marketplace	1.1.0

In order to simplify the development of the Platform, it has been provided a single git repository where (almost) all the components developed or used in NUBOMEDIA have been added as submodules. The repository is available at: https://github.com/nubomedia/nubomedia-controller.
The Autonomous Installer has been further extended in order to simplify the deployment of the newest version of the Platform as per IST_R1. The description about the extended functionality is reported in the next sections. The documentation on how to use the Autonomous Installer is available at: http://nubomedia-developer-guidelines.readthedocs.io/en/latest/tools/autonomous-installer/

4.2 NUBOMEDIA IaaS

As already mentioned, no changes were needed to be applied to the NUBOMEDIA IaaS in order to support the new requirements. OpenStack Kilo resulted to be enough for supporting the new functionalities required by the Application developers.

4.3 NUBOMEDIA Media Plane

The Open Baton Media Plane components developed in the context of the NUBOMEDIA Cloud Platform, are mainly management and orchestration components providing functionalities for controlling the lifecycle of the Media Plane entities on top of the IaaS. Not many changes have been required for supporting new requirements coming from application developers, however for the latest release of NUBOMEDIA, Open Baton has been upgraded to the latest available release (3.0.0). The changes which have been developed in the context of NUBOMEDIA on the NFVO, have been released and made available also to the Open Baton open source community.

4.3.1 Network Function Virtualization Orchestrator (NFVO) extensions

The NFVO was further extended to support two main functionalities as requested by DRT_R1: Start/Stop a VNF Components, meaning Kurento Media Server instances.

As initial step the NFVO APIs were extended to expose those functionalities to the PaaS Manager. In particular, the PaaS Manager requests the NFVO where the NFVO forwards these requests to the Media Server VNFM which is in charge of controlling the VNF Components and its life cycle. After the initial deployment, all the KMS instances are started.

As shown in the Figure 3, in the first step the PaaS Manager requests the NFVO in order to stop a given Media Server. In the second step the NFVO forwards the requests to the Media Server VNFM. When the Media Server VNFM receives the request for stopping a specific instance, it removes it from the list of available media servers (step 3). Once the Media Server VNFM finished this action (step 4), it sends an acknowledgement back to the NFVO. Afterwards, this stopped instance is still running and serving requests from already registered session but they are not available anymore for new sessions. Nevertheless, already registered sessions, can be removed again by executing the common action for unregistering session. Since the request for stopping requests is asynchrony, the NFVO does not send any acknowledgements back to the PaaS Manager.

If an instance has been stopped, it can be started again by calling the start method on a specific instance (step 5). The NFVO requests again the Media Server VNFM in order to start the specific VNF Component (step 6). Once the Media Server VNFM received the request, it starts the instance and puts the given media server again into the list of available media servers (step 7). Finally, the manager sends an acknowledgement back to the NFVO. After this action, the media server is available again and can be used for assigning new sessions in order to consume capacity. Similarly to the process of stopping a media server, this method is asynchrony, so the NFVO does not send back any message to the PaaS Manager.

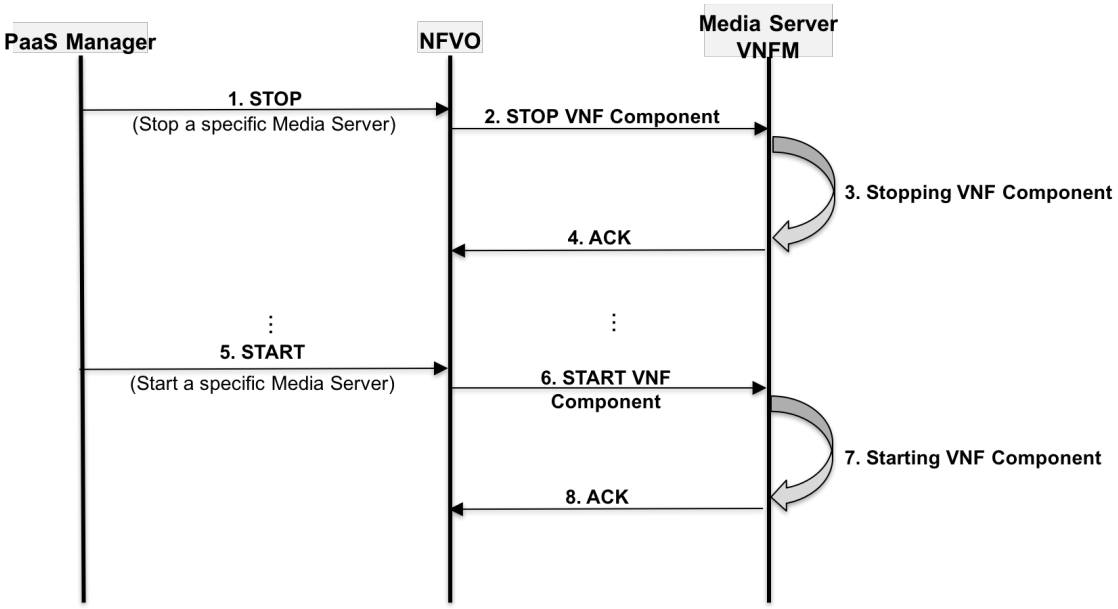


Figure 3. Sequence diagram showing the start and stop functionality

4.3.2 Cloud Repository role in the NUBOMEDIA infrastructure.

The Cloud Repository has not been extended in the last releases as its main functionalities were enough to support requirements from application developers. The Cloud Repository scripts and descriptors used for being deployed via Open Baton have been updated in order to work with release 3.0.0.

4.3.3 Media Server VNFM

The autoscaling capabilities of the Media Server VNFM were further extended in order to support an advanced scaling in mechanism. Usual scale in mechanisms are not considering sessions running on an existing instance, which may cause in some situations some loss of connectivity from the end user perspective. In the context of NUBOMEDIA, scaling in a media server having some running sessions would cause a termination of a multimedia flow with the end users. Therefore, the proposed mechanism considers the number of running sessions in a media server as a metric for the scale in procedure.

In particular, once the VNFM recognizes the need of scaling in any media server (based on a defined threshold), it checks first if scaling in can be performed. Scaling in a media server is only permitted if no sessions are running on the potential media server, which means that, the media server is idle and no capacity of the media server is consumed by any application. If the VNFM finds any media server which is not occupied by any sessions, scaling in is permitted and executed.

4.3.4 Connectivity Manager (CM)

The Connectivity Manager has been further extended to support autoscaling. It means that connectivity requirements are also enforced on virtual links which are instantiated runtime by the autoscaling system of the VNFM.

4.4 NUBOMEDIA PaaS

4.4.1 NUBOMEDIA PaaS Manager

In this section is explained how the PaaS layer software components have been extended to support additional requirements provided by the Application developers. Figure 4 shows the implementation of the PaaS Manager functional elements and its APIs. It is important to clarify that no changes on the functional architecture were required, and most of the work has been conducted on the Software implementation. A huge refactoring has been executed between Release 6 and 7 including the introduction of additional resilient mechanisms for improving the PaaS stability.
The PaaS Manager has been extended for supporting the following requirements: DRT_R4, DRT_R7, DRT_R8, DRT_R9. Before going into the details about the new functionalities implemented for supporting those additional requirements, it is important to briefly recap the PaaS Manager Architecture and the state diagram of a NUBOMEDIA application.

As already mentioned in D2.4.2 [2], the PaaS Manager simplify the way developers are instantiating their applications providing a high level abstraction of the information required by the lower levels. The main functionalities provided by the PaaS Manager are:

1. Building and deployment of Applications on NUBOMEDIA PaaS
2. Requesting the instantiation of Media Components interacting with the NUBOMEDIA Media Plane management components

The PaaS Manager is a Spring Boot application that uses Spring framework functionalities to expose REST APIs, perform REST requests to the NUBOMEDIA PaaS and store developers application useful data. It also uses NFVO SDK to interact with Open Baton for requesting the instantiation of Media Components.
NUBOMEDIA PaaS component is Openshift Origin v3, a PaaS platform developed by Redhat that offer containerization using Docker, Kubernetes and Project Atomic.
As shown in Figure 4,the PaaS Manager is composed by five main components:

- API: the external REST API where developers could send requests to perform authentication, application creation/deletion, etc.
- PaaS Manager: a component that interact with connectors to request media server allocation and application building and deployment
- PaaS Connector: is used to perform request to the PaaS through its REST API
- NFVO Connector: request to the NFVO for allocating Media Components on the NUBOMEDIA IaaS/Media Plane
- Repository: this component stores the Application data that will be used for deletion, query status and UI

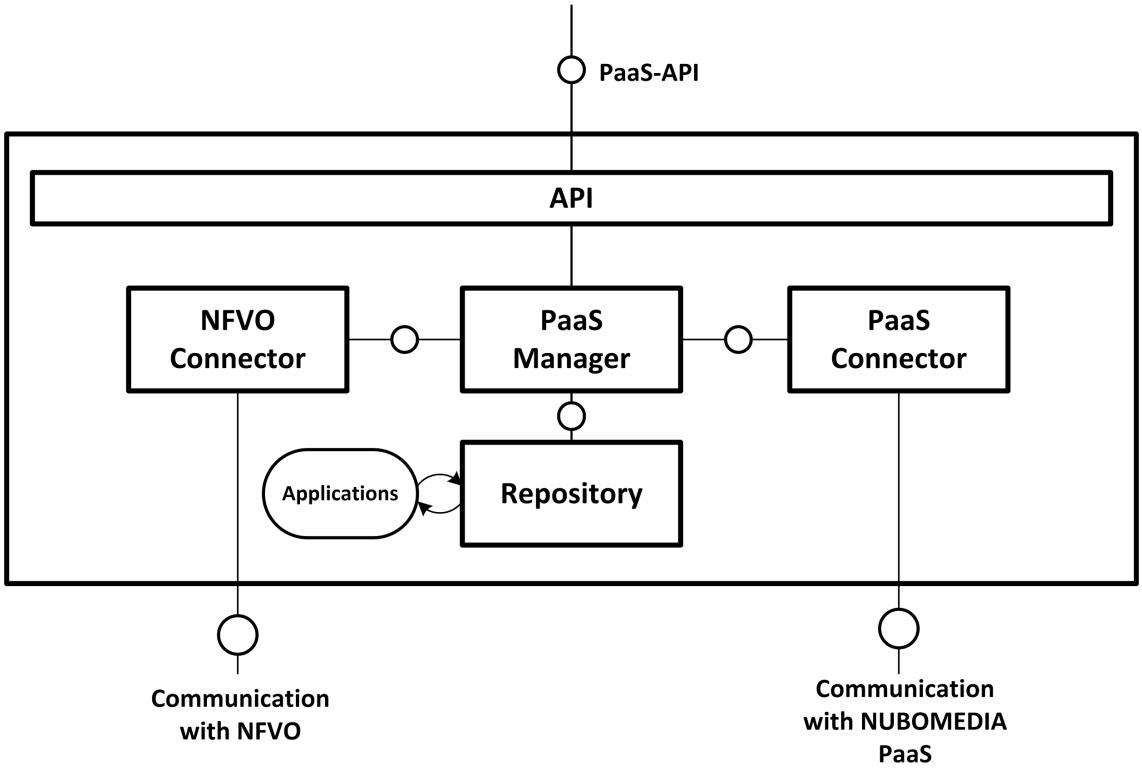


Figure 4. PaaS Manager architectural view

Figure 5 shows the entire life cycle of an application from the initial request up to the moment where the application is running. In step 1 the application developer triggers the deployment of an application by providing the application definition based on a JSON configuration file. What such an application request contains, is already described above. Once the application is persisted in the database, the application status goes to CREATED. Once the application is persisted in the database, required resources will be requested (step 2a) by requesting the NFVO. If the request was properly sent, the application status goes into INITIALISING. If not, the status will end up in FAILED (step 2b). In step 3a required resources will be allocated before the status goes to INITIALISED. If something went wrong during the allocation (e.g. no resources left), the status goes to FAILED. The INITIALISED status indicates that the virtual instances containing the kurento mediaserver are present and configured. Once the PaaS Manager received the notification from the NFVO, it starts to build the application by requesting OpenShift (step 4). If the building fails, it goes directly to FAILED (step 5b). If the building of the application finishes without any problems (step 5a), the deployment is triggered in step 6a. If problems during the deployment will occur, the application itself goes to FAILED. Finally, if the deployment finished without any exceptions, the application goes to the status RUNNING. Once it goes to RUNNING, the application itself will start. From this status it might happen that exceptions may occur at runtime. In this case it goes to FAILED (step 7) but here it is possible to recover the application and bring it back to status RUNNING (step 8).

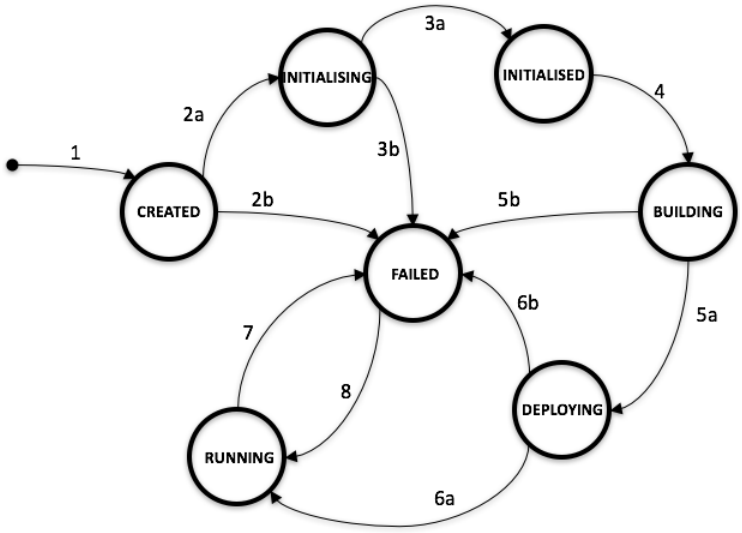


Figure 5. State diagram of an application

4.4.1 API V2

The API component uses the RestController to map request paths, request's body and/or path parameters. The root path for all the requests is "/api/v2/nubomedia/paas". It uses GSON as default de-serializer of the requests body.

Every request has to be authenticated with a "Auth-Token" in the request headers. The PaaS API are RESTful APIs accessible via HTTP(s) on the PaaS Manager server. These REST APIs expose abstractions over specific operations to developers allowing them to deploy and manage their end-user applications on the NUBOMEDIA PaaS. Table 1 gives an overview of the exposed REST APIs.

Table 1. NUBOMEDIA PaaS REST API

Method	URL	Description
POST	/api/v2/nubomedia/paas/users	Create a new PaaS User
DELETE	/api/v2/nubomedia/paas/users/{name}	Deletes a PaaS User
POST	/api/v2/nubomedia/paas/app	Create (build and deploy) a new application
GET	/api/v2/nubomedia/paas/app/{id}	Retrieve the status of an application with the given id.
GET	/api/v2/nubomedia/paas/app	Return the list of all deployed projects
DELETE	/api/v2/nubomedia/paas/app/{id}	Delete project with the given id
POST	/api/v2/nubomedia/paas/secret	Create a secret[2]
DELETE	/api/v2/nubomedia/paas/secret/{name}	Deletes a secret
POST	/api/v2/nubomedia/paas/auth	Authenticate a PaaS User
GET	/api/v2/nubomedia/paas/app/{id}/logs/{podName}	Retrieve logs of specific containers
PUT	/api/v2/nubomedia/paas/app/{id}/media-server/{hostname}/start	Starts the given mediaserver
PUT	/api/v2/nubomedia/paas/app/{id}/media-server/{hostname}/stop	Stops the given mediaserver
POST	/api/v2/nubomedia/paas/app/{id}/media-server	Adds a new mediaserver (scaling-out)
DELETE	/api/v2/nubomedia/paas/app/{id}/media-server/{name}	Removes a (given) mediaserver

[2] The Secret object type provides a mechanism to hold sensitive information such as passwords, client config files, dockercfg files, private source repository

		(scaling-in)
GET	/api/v2/nubomedia/paas/app/{id}/media-server/{name}	Returns information about (given) mediaserver(s)

For creating an Application it is necessary to send a JSON object like the one shown below:

```
{
   "gitURL":"https://github.com/example/example-app.git",
   "name":"my-app-name",
   "cloudRepository":boolean,
   "cloudRepoPort":"cloud-repository-desired-port",
   "cdnConnector":boolean,
   "flavor":"SMALL/MEDIUM/LARGE",
   "ports": [
      {
         "port": 8443 (for example),
         "targetPort": 8443 (for example),
         "protocol": "TCP" (for example)
      }
   ],
   "replicasNumber":2,
   "secretName(optional)":"defined-secret-name",
   "numberOfInstances":3,
   "stunServerActivate":boolean,
   "stunServerIp(mandatory if enbaled)":"stun-server-ip",
   "stunServerPort(mandatory if enbaled)":"stun-server-port",
   "turnServerActivate":boolean,
   "turnServerUrl(mandatory if enbaled)":"turn-server-url",
   "turnServerUsername(mandatory if enbaled)":"username",
   "turnServerPassword(mandatory if enbaled)":"password",
   "scaleOutLimit":maximum-number-of-instances,
   "scale_out_threshold":average-capacity-consumed-scaling-out,
   "scale_in_threshold":average-capacity-consumed-scaling-in
   "qualityOfService(optional)":"BRONZE/SILVER/GOLD",
   "services": [
      {
         "name":"name-of-supporting-service",
         "dockerURL":"url-of-docker-image",
         "replicasNumber":1,
         "ports": [
            {
               "port":3306,
               "targetPort":3306,
               "protocol":"TCP"
            }
         ],
         "envVars": [
            {
               "name": "MYSQL_DATABASE",
               "value": "nubo"
            }
         ]
      }
   ]
}
```

Where the parameters have the following description:

- gitURL: git repository where the jar and Dockerfile (and even other files that are necessary for the application) are commited (if the repository is public the link has to be the https version, if is private has to be the ssh version)
- appName: the application name that will be used also to create the DNS entry to use your application

- cloudRepository: boolean value to require the Cloud Repository;
- qualityOfService: optional value to require the QoS for intra-mediaserver communication
- flavor: enumerative to set the flavor of the mediaserver
- ports: an object that maps the ports that are used from container to the ports that has to be exposed to outside, with relative protocol
- replicasNumber: the number of containers that has to be created by the PaaS after the building phase
- secretName (optional): the name of the secret that has to be used only if your application is on a private git repository
- turnServerActivate: boolean value to enable the turn server on the mediaserver, if turnServerIp, turnServerUsername and turnServerPassword are not specified the mediaserver will use the default one;
- turnServerUrl: the url of the turn server if different from the default one (example: turn:192.168.43.12:8080)
- turnServerUsername: the username of the turn server (mandatory in case is specified the turnServerIp)
- turnServerPassword: the password of the turn server (mandatory in case is specified the turnServerIp)
- stunServerActivate: boolean value to enable the stun server on the mediaserver, if stunServerAddress and stunServerPort are not specified the mediaserver will use the default one;
- stunServerIp: the stun server ip, if is settled also the stunServerPort has to be settled;
- stunServerPort: the stun server port (as string), is mandatory if the stunServerAddress is settled
- scaleOutLimit: the maximum number of mediaserver instances for scaling
- scale_in_threshold: the maximum capacity of the media server to scale in
- scale_out_threshold: the minimum capacity of the media server to scale out
- services: supporting services for providing additional services used by the main application, e.g. databases and others services.
    - name: human readable name of the supporting service
    - dockerURL: URL where the docker images is available, e.g. docker hub
    - replicasNumber: is a numeric value describing the number of containers
    - ports: indicates the transport protocol and ports exposed for the supporting service
    - envVars: list which contains key/value pairs that defines the environment variables used by the supporting service for configuration.

4.4.1.1 Identity Management

The identity management manages access rights of the PaaS Manager itself. The concept introduced is user-driven which means that identities, namely users, are assigned to one or more projects with specific roles. Hence, every user can have a specific role in a certain project. In turn, a certain project may contain several users which have access to all applications deployed under the project. User-role relations define the rights a user have. In particular, a user with the role “USER” has full access

to the assigned project. Full access mean more in detail, the user can deploy and remove applications whereas a user with the role “GUEST” can access the PaaS Manager and the assigned project but can only gather information, for instance, retrieve application details and information for using the application itself. However, a “GUEST”-user cannot deploy or remove any applications. Overall permissions are given to “ADMIN” users which have access to the entire PaaS Manager. Administrators can select all existing projects and moreover, admins have also access to the identity management in order to create, modify and remove projects, users and roles. A particular role is given to the “admin” project and user. Both the “admin” project and “admin” user are existing by default and cannot be deleted by any administrator. Additionally, the “admin” project shows all existing applications including also all applications from all the other projects. In addition, the “admin” user is the initial and only user which is present from the beginning in order to access the PaaS Manager.

4.4.1.2 Application logs

Application logs can be retrieved at both instantiation time and run time. Hence, the PaaS Manager communicates directly with OpenShift in order to request the logs while building, deploying and executing the application. These logs are provided back to the GUI of the PaaS Manager which gives the application developer/maintainer the ability to debug and observe the status of the application at any time.

4.4.1.3 Marketplace

The newly introduced marketplace serves as a platform in order to offer and distribute applications to the community. Hence, it contains the definitions of applications which can be downloaded and on-boarded to the PaaS Manager in a single step. The Marketplace is a Java Application making using an external database for storing application definitions. It exposes a REST API for CRUD operations upon application descriptors. Therefore, the PaaS Manager GUI directly interact with the Marketplace in order to expose a dynamic catalogue to application developers who could store or retrieve application descriptors anytime. Moreover, the PaaS Manager is configurable in order to change the marketplace URL by defining its IP. This enables also the usage of different marketplaces, either for using a public marketplace populated by the community or a private marketplace containing the definition of applications which should not be publicly available for the community.

4.4.1.4 Manual scaling of media servers

Manual scaling of media servers: The PaaS Manager allows manual scaling actions, in particular, scaling in and scaling out. Therefore, once the PaaS Manager received the request for scaling, it requests the NFVO in order to execute the given scaling action. The NFVO forwards the request to the Media Server Manager which executes finally the scaling action. If scaling out is requested, the Manager allocates a new mediaserver and puts it to the list of available instances which might be used directly for registering new sessions and occupying capacity. If a scaling in action is requested, the Media Server Manager terminates the given instance immediately without taking care about any applications or any sessions using this specific instance. Applications or sessions which used this instance will be removed from point of view of the Media Server Manager.

4.4.2 Deploy additional services with an application

As specified by requirement DRT_R4, supporting services are introduced in order to give the application developer an easy mechanism to make use of additional and existing services. These services provide additional functionalities which are consumed by the main application, e.g. database and other services. Information of these additional services are provided back to main application based on the name of the supporting service itself and the parameters provided. The URL for accessing the service are available in the main application composed by the name of the service and the keys of the environment variables. In case of the URL, for instance, <NAME>_HOST.

4.4.3 NUBOMEDIA PaaS GUI

In order to have the PaaS GUI implemented we needed to first gather the user requirements on D2.3.3. and then to identify what capabilities are needed to be implemented on the PaaS. We then needed to find what is the best place inside the interface. For this we first designed a mockup with just basic functionalities that was then subject for discussion and changes during two weeks of iterations. Some of the first mockup pages can be seen in the next two figures:

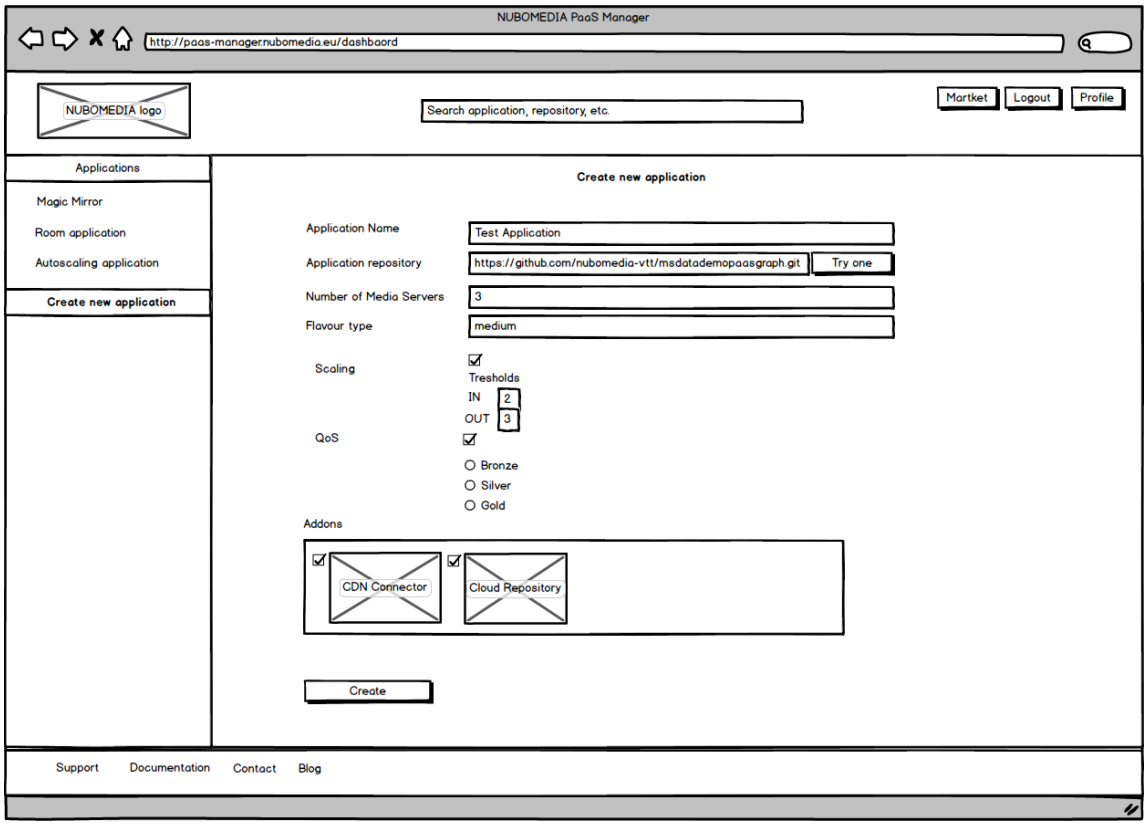


Figure 6. NUBOMEDIA application deployment - mockup

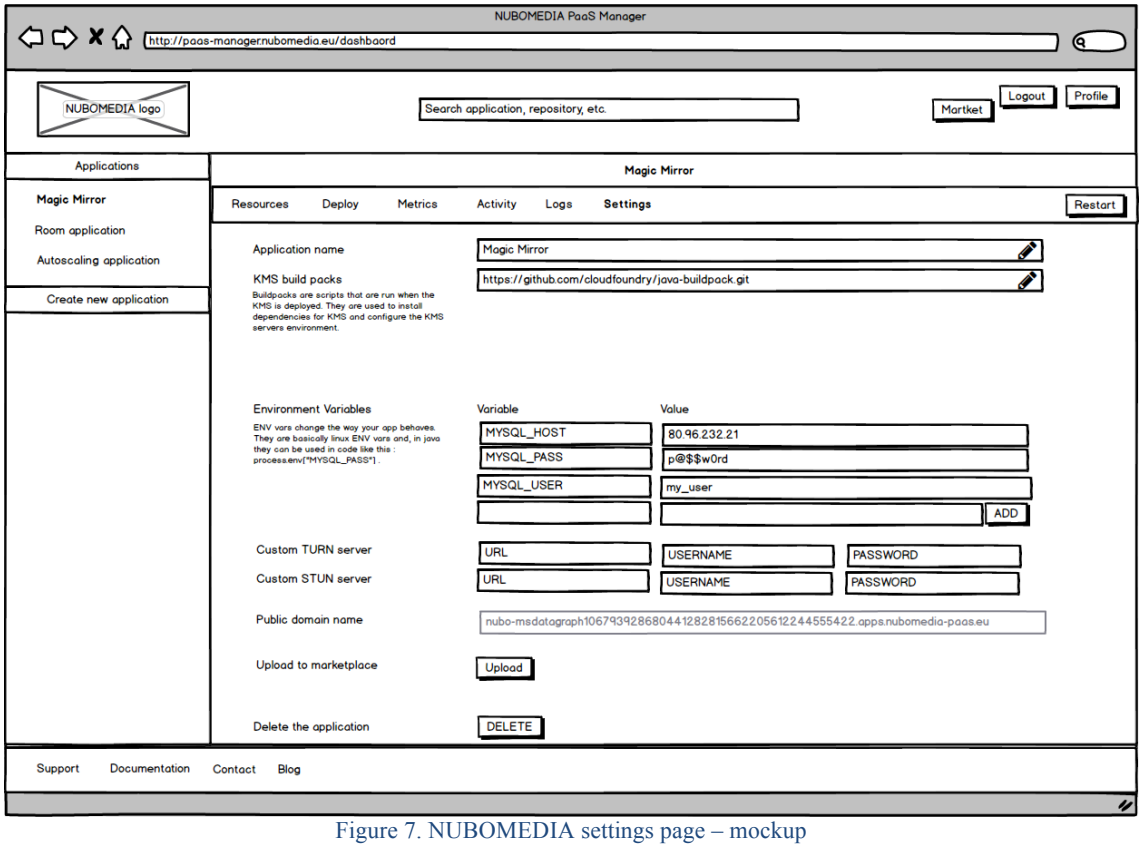


Figure 7. NUBOMEDIA settings page – mockup

After concluding on the functionalities part we then started the design phase, which had as an output an Invision[3] project.
Invision allows you to transform your designs into clickable, interactive prototypes that ca then be shared so you can then receive feedback directly on the design and you can collaborate with others. We've added figure 7 in order to present the proposed design in case Invision link is no longer working.

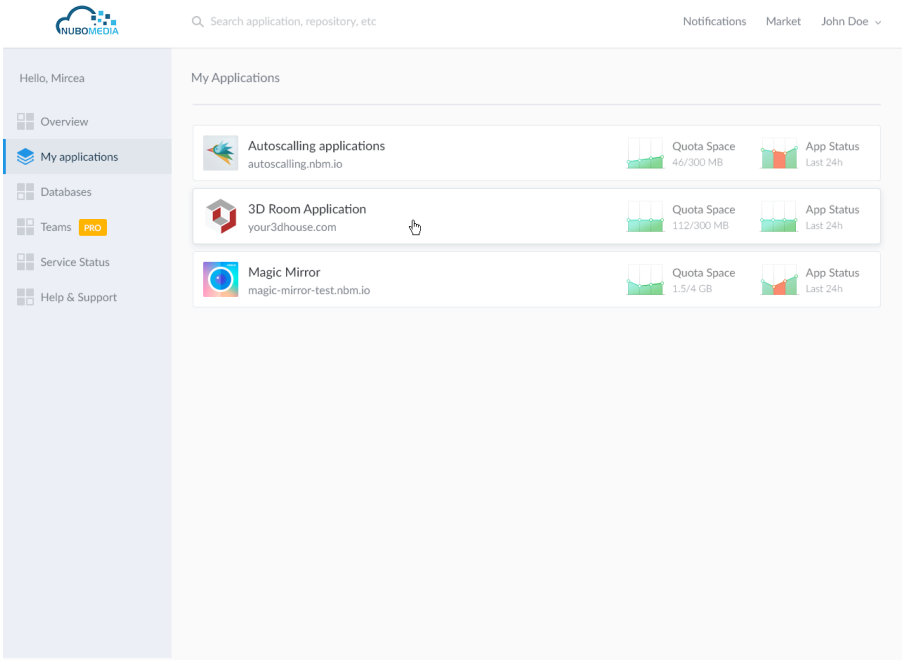


Figure 8. NUBOMEDIA PaaS GUI – My applications page design

[3] https://projects.invisionapp.com/share/W87RRH9CV#/screens/169667441_1

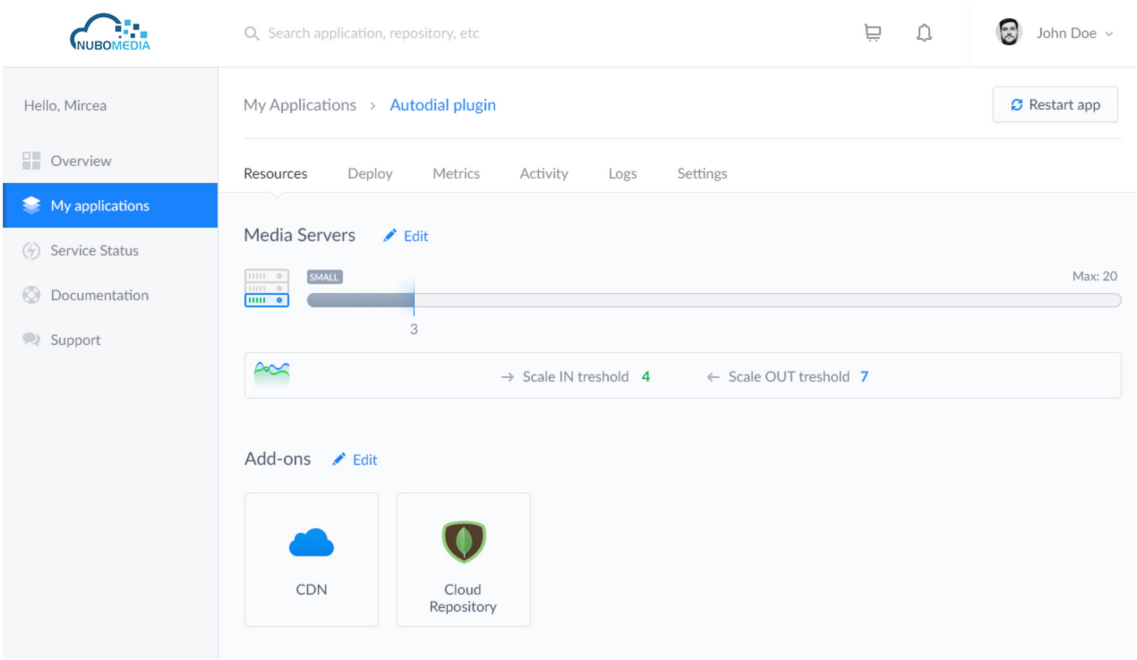


Figure 9 NUBOMEDIA PaaS GUI - Application overview

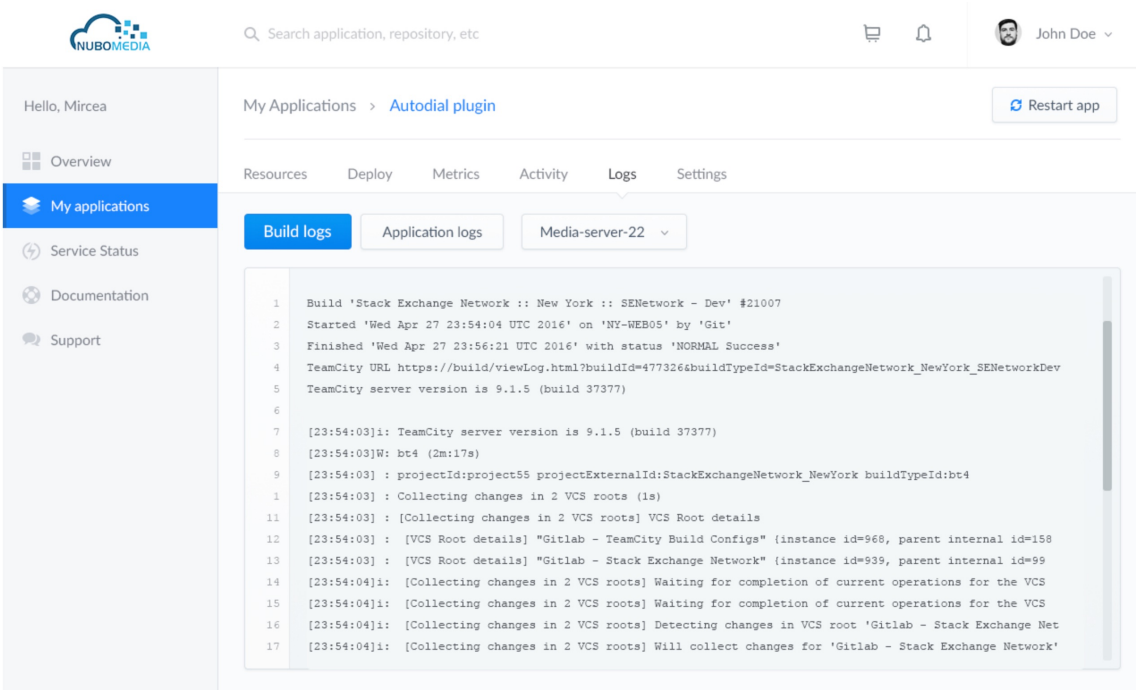


Figure 10 NUBOMEDIA PaaS GUI - Logs section

4.4.3.1 Technologies behind the NUBOMEDIA PaaS GUI

NUBOMEDIA PaaS GUI is a single page web application that was created using mainly AngularJS framework for JavaScript functionality and Bootstrap framework for designing components with HTML and CSS.

A single-page application (SPA) is a web application that fits on a single web page with the goal of providing a user experience similar to that of a desktop application. In an SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions. The page does not reload at any point in the process, nor does control transfer to another page, although the location hash can be used to provide the perception and navigability of separate logical pages in the application, as can the HTML5 pushState() API. Interaction with the single page application often involves dynamic communication with the web server behind the scenes.

Although Bootstrap was used, lots of components from the application were implemented using custom CSS and HTML. In order to have a clean and a modular approach the CSS was written using Sass preprocessor and in this way all the CSS components were stored in separated files giving the project a more robust and stable file structure.

Some of the ways that frameworks can help a project:

- Prevent repetition between projects
- Utilize responsive design to allow your website to adapt to various screen sizes – mobile, desktop, and everything in between
- Add consistency to design and code between projects and between developers
- Quickly and easily prototype new designs
- Ensure cross-browser compatibility

Others plugins were used in order to make the development much easier. For example, GULP was used to compile the Sass in CSS and to concatenate source files. Font awesome was used in order to have a set of icons that are represented as fonts and as a result can be scaled much easier.

We used Google charts in order to draw the applications main charts on the debug section.

4.4.3.2 NUBOMEDIA PaaS GUI presentation

In order to have access to the NUBOMEDIA PaaS system a user have to login first. The user journey starts with the login form, Figure 11. There are two types of users, admin and normal user.

Admin user has access to all the views of the application while a normal user has access only to the information that is related to him. His applications and the project that he belongs to. From now on the application will be presented from the admin point of view because the admin user as was said has access to all views. After the credentials are entered the user will see the first page of the application which is a dashboard with general information about users, projects and active applications.

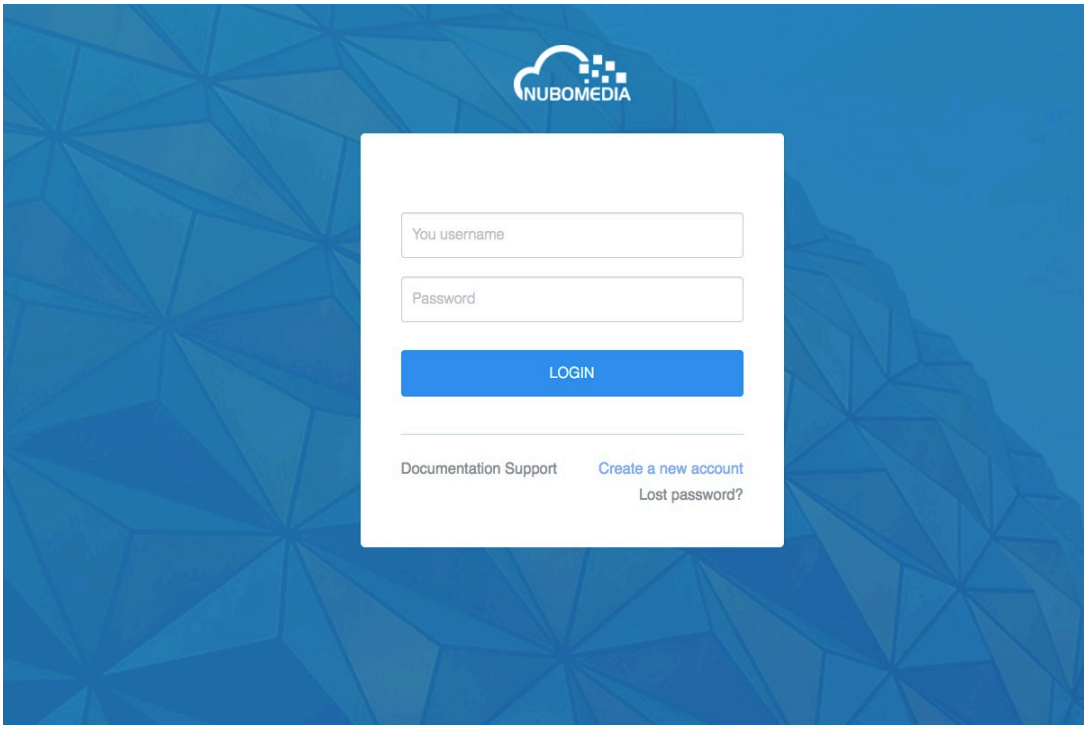


Figure 11. NUBOMEDIA Login page

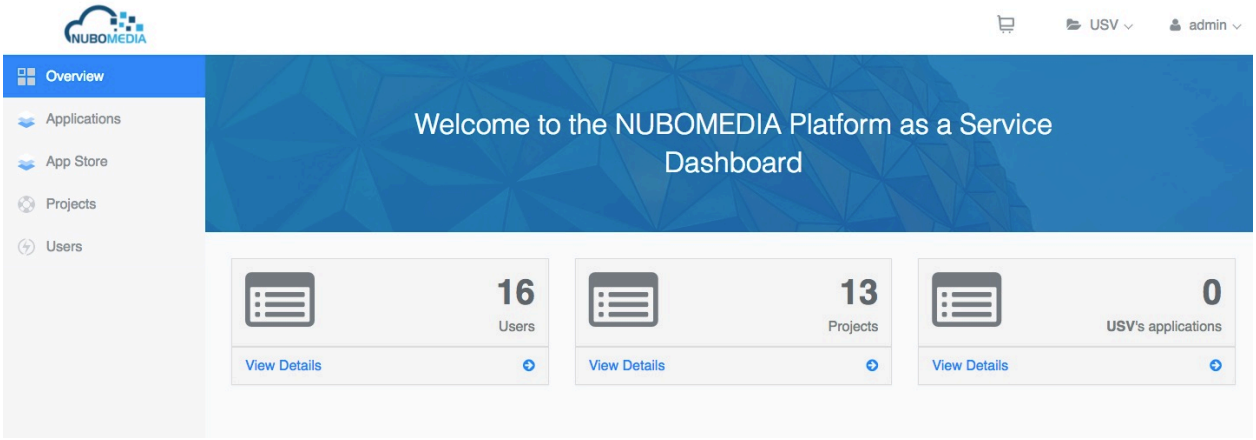


Figure 12. Dashboard view

As it can be seen in Figure 12, on the left we have the application's main navigation. In top right we have the user name, the current active project and a link to the market applications. In center we have some cards with general information about: users number, projects numbers and applications numbers. We are able to navigate to other pages using the main navigation from left or using the links from dashboard cards.

Next going to Applications page (Figure 13) we have the following view:

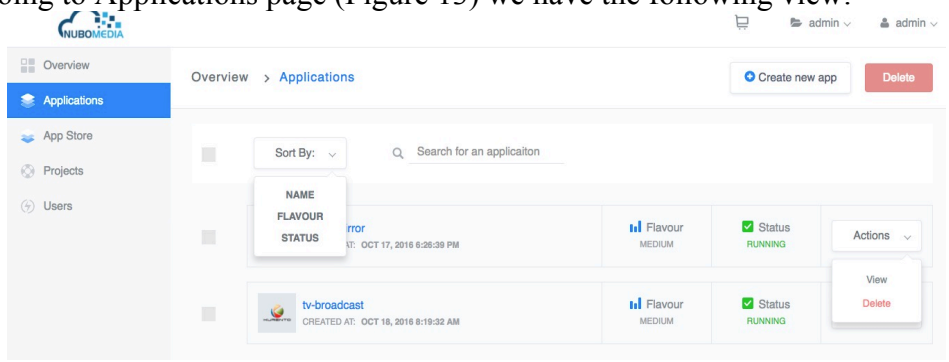


Figure 13. Applications list

In this view we can see the list of applications related to a specific project. We also can see here the application name, the date when the application was created, the application status and its flavour.

In this view we have the possibility to search for a specific application using the Search for an application form. The search form allows us to search not only by name but also by created date, flavour or application status. It does a smart search and the important aspect of this is that the search is not made using some predefined terms. What this means? It means that if we decide to display some extra information about an applications like for example the number of servers for every application. When we would use the search we could also search for an application that has a specific number of servers. All this without changing the search functionality.

Then we have the Sort by functionality that sorts the applications depending on what we are choosing from select box: name, flavour or status.

We also have some secondary actions buttons for every application. We can delete an application or we can view the full application description and all the fields that are related to that application, we will see an example of that.

Figure 14. Breadcrumbs and main actions buttons

On the left side of Figure 14 we have the breadcrumb that tells the user where he is and on the right we have the main actions buttons. Delete action that is enabled when one or more applications are checked permits the user to delete multiple applications. Create new app is redirecting the user to another view where he can create a new application that on completion will show up next to other applications. When clicking on create new app button the user is redirected to a new view where he can create a new custom application.

Create new application view is practically a form where the user will add specific information about the new application that he wants to create. This form is split in 3 main sections: General information, Services and Additional information.

4.4.3.2.1 Applications

In order to create a new application a user has to input the following fields in the general information section:

- Application name
- Git URL
- Replicas number
- Flavour
- Secret name

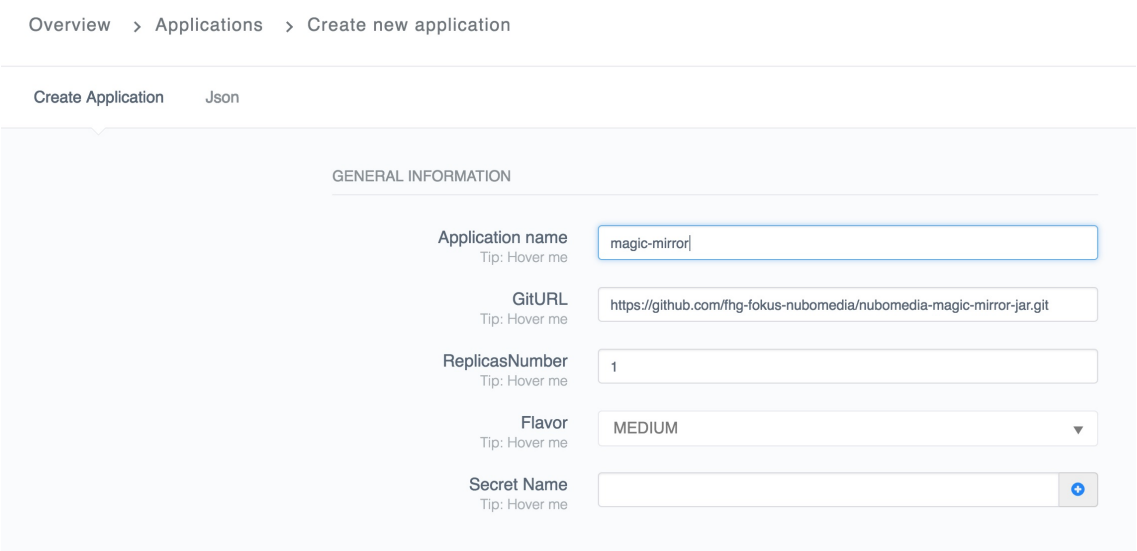


Figure 15 NUBOMEDIA PaaS GUI - Create new app view: general information section

The services section is optional and the user has the possibility to add multiple services using the Add a service button. In this section you should add the following details about the add-on:

- Docker URL
- Name
- Replicas number
- Multiple ports
- Multiple environment variables

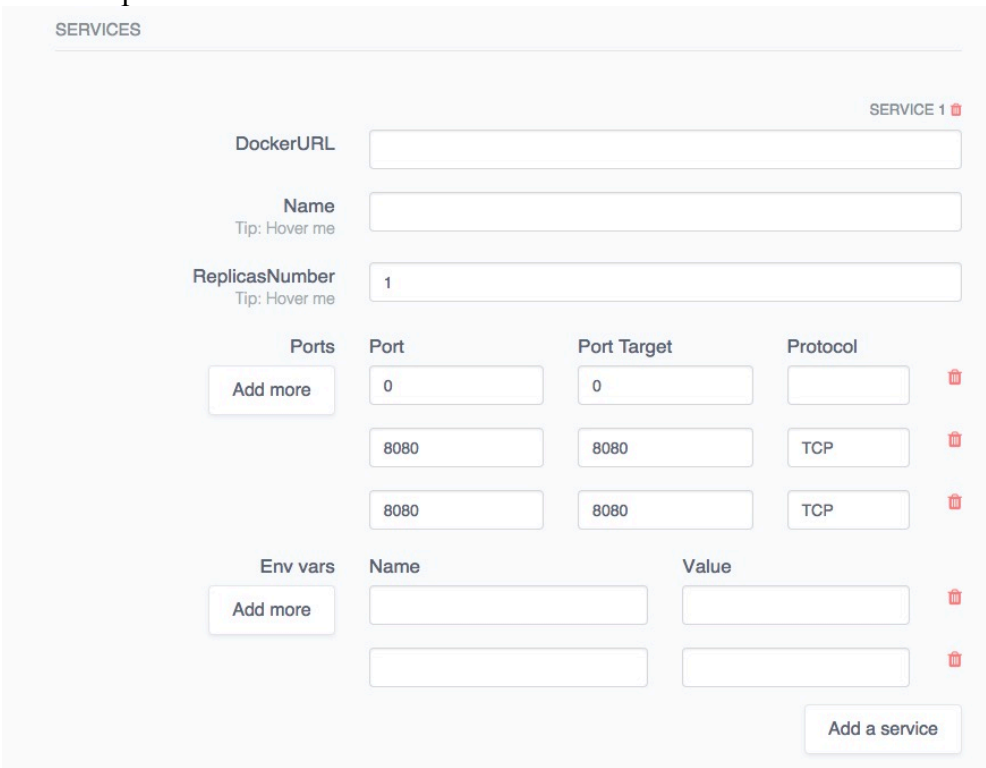


Figure 16 NUBOMEDIA PaaS GUI - Additional services

In Additional information section, the user can opt to enable auto scaling of the media servers by configuring the Scale out limit and Scale out trigger. We enabled the auto-scaling of the media servers considering the requirement PGUI_R1 gathered from internal and external users.

A user is also able to override the configurations for having a custom Turn or Stun server or define some custom ports that his application need to expose.
Another option is to have deployed along with the application also a CDN Connector or a Cloud Repository with APIs that can be consumed by the deployed application. All these custom options are illustrated in the following screenshot.

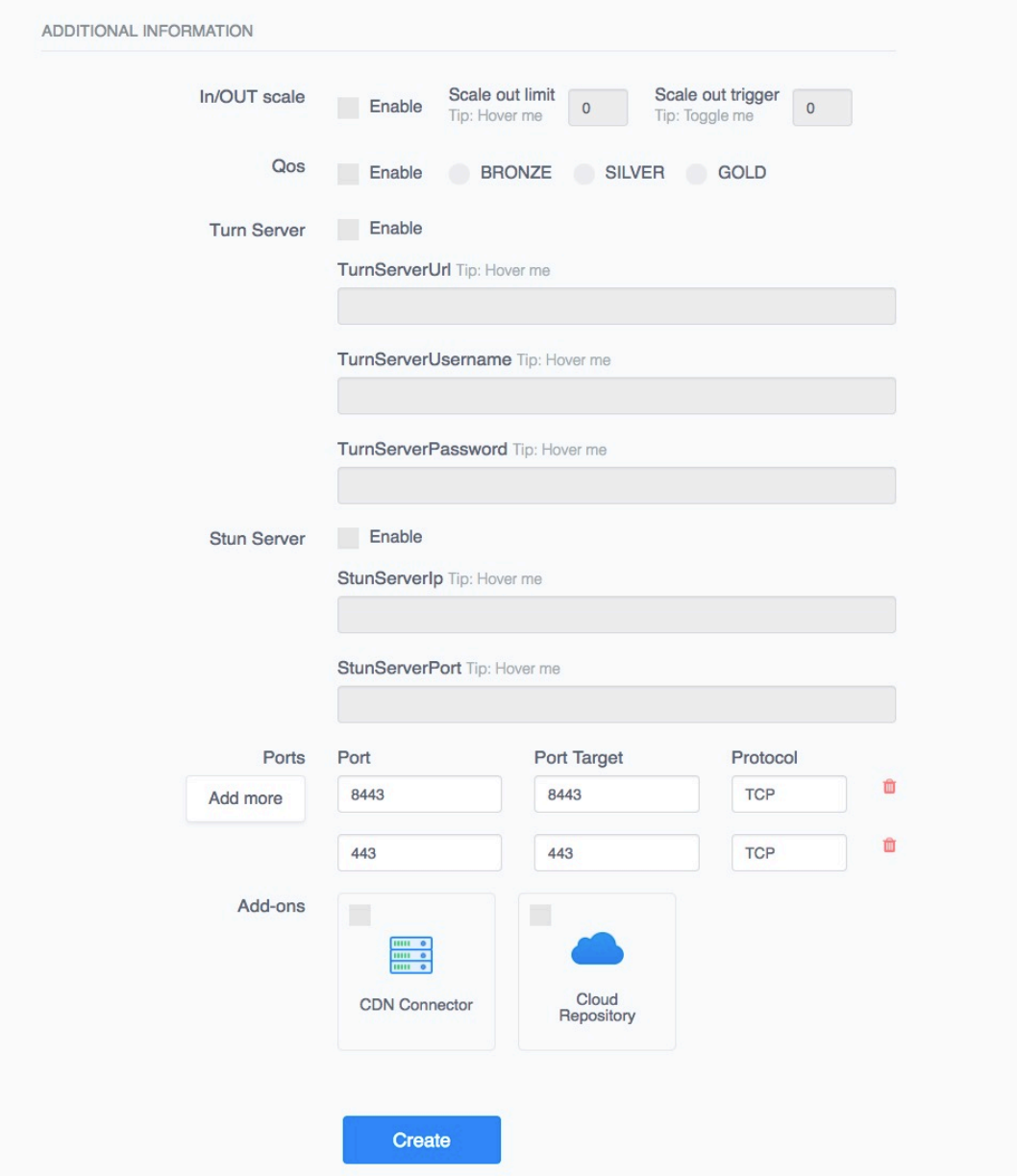


Figure 17 NUBOMEDIA PaaS GUI - Create application with additional information

After all the necessary fields are completed the user can click the create button and a new application will be created with the specified parameters.

An important aspect of this step is that the user has the possibility to create the same application but using the Json tab of the form. This tab of the form permits the user to create a new application by uploading a configuration json file from his computer.

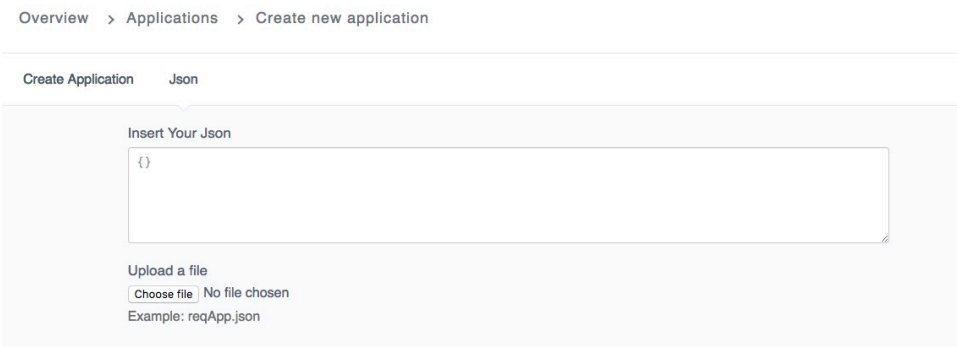


Figure 18 NUBOMEDIA PaaS GUI - Create application from Json

In application details view the user has all the information about an application separated in 4 tabs:

- Overview
- Metrics
- Debug
- Settings

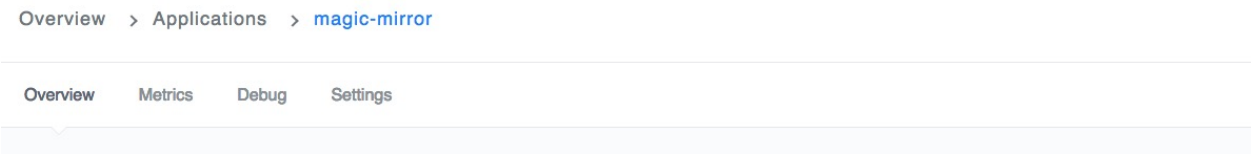


Figure 19 NUBOMEDIA PaaS GUI - Application Details

Application overview tab has the application URL, media servers information and the application scalability features like powering on or stopping one KMS instance or even scale OUT/IN the number of media-servers.

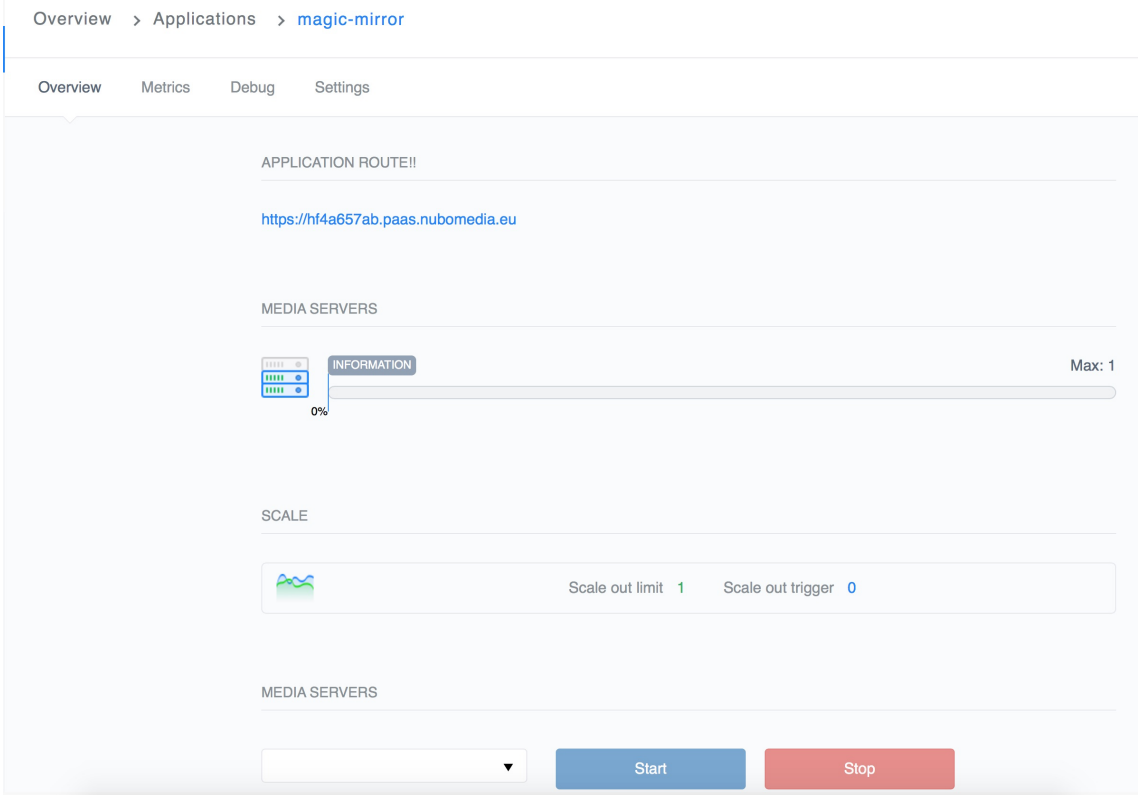


Figure 20 NUBOMEDIA PaaS GUI - Application overview

Metrics tab contains various charts with information requested in technical requirement gathered in D2.3.3, DRT_R11 having the following description: “it shall be possible to monitor the CPU, Memory, and other metrics available”.
Some of the collected metrics are the following:

- Numbers of KMS instances
- Total capacity used
- Media servers metrics:
  - Memory
  - Elements
  - Pipelines

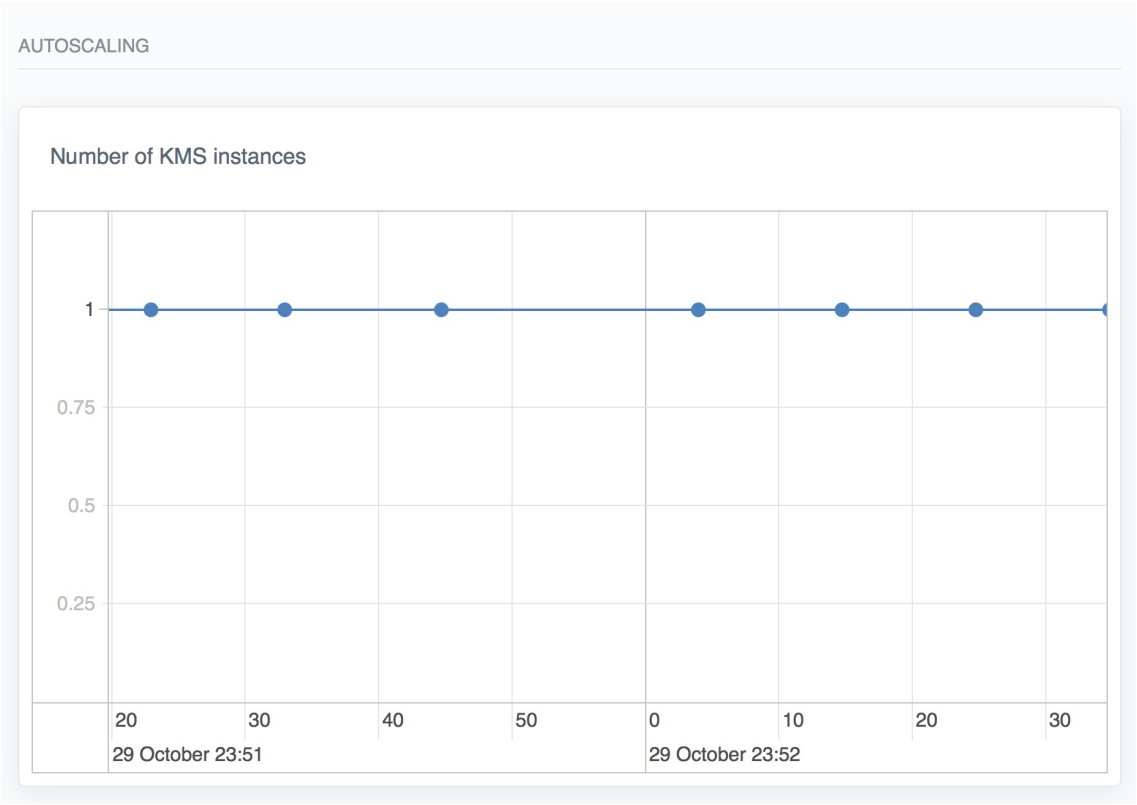


Figure 21 NUBOMEDIA PaaS GUI - Scaling informations

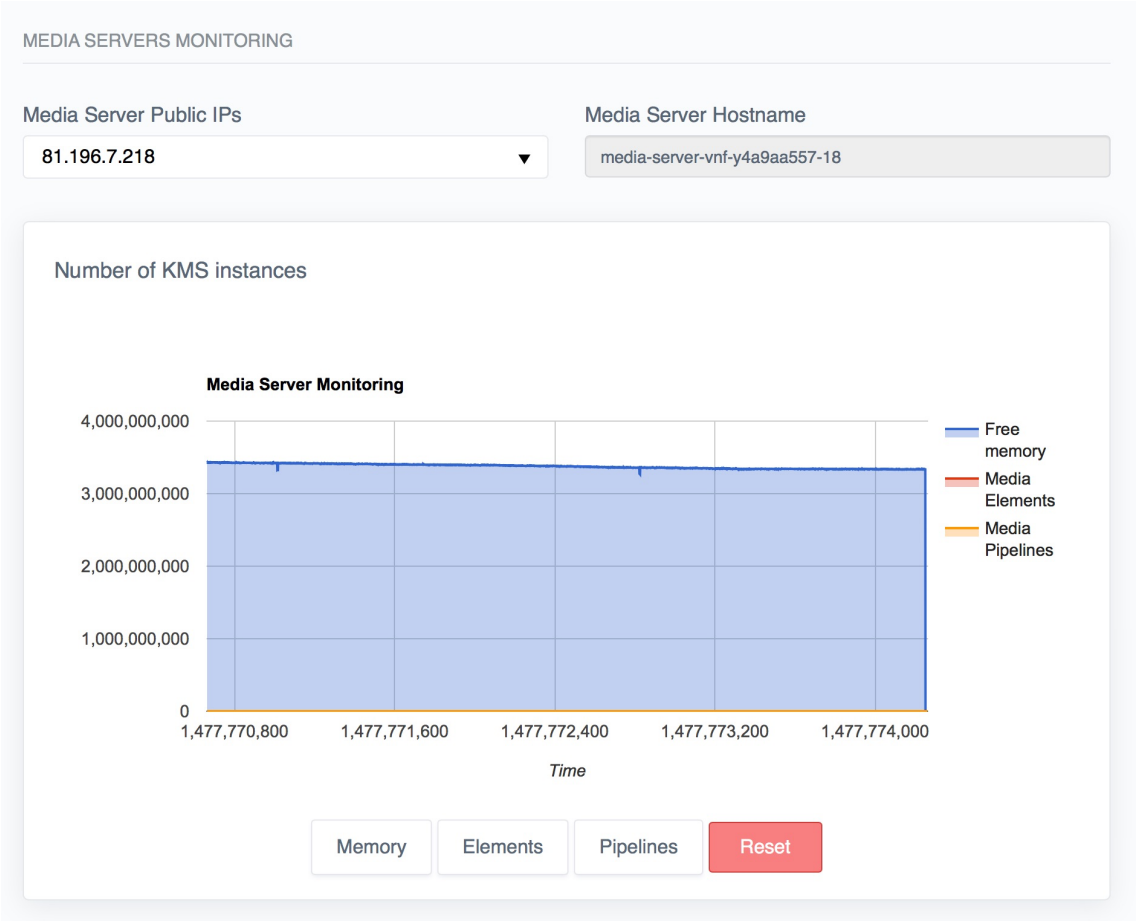


Figure 22 NUBOMEDIA PaaS GUI - Media Server monitoring informations

Debug tab contains more information about media servers. Here the user has the possibility to build application logs. Also, a user can start or stop a specific server.

In order to fulfill the developer requirement DRT_R3, to allow the developer to SSH into the KMS instances, we configured all media-server docker containers and instances with a username and password so anyone can login on any of the IP addresses of the media-servers found on the media-server monitoring tab. This feature is now not so important because we also have now the possibility to check the media-server logs directly on the PaaS GUI interface.

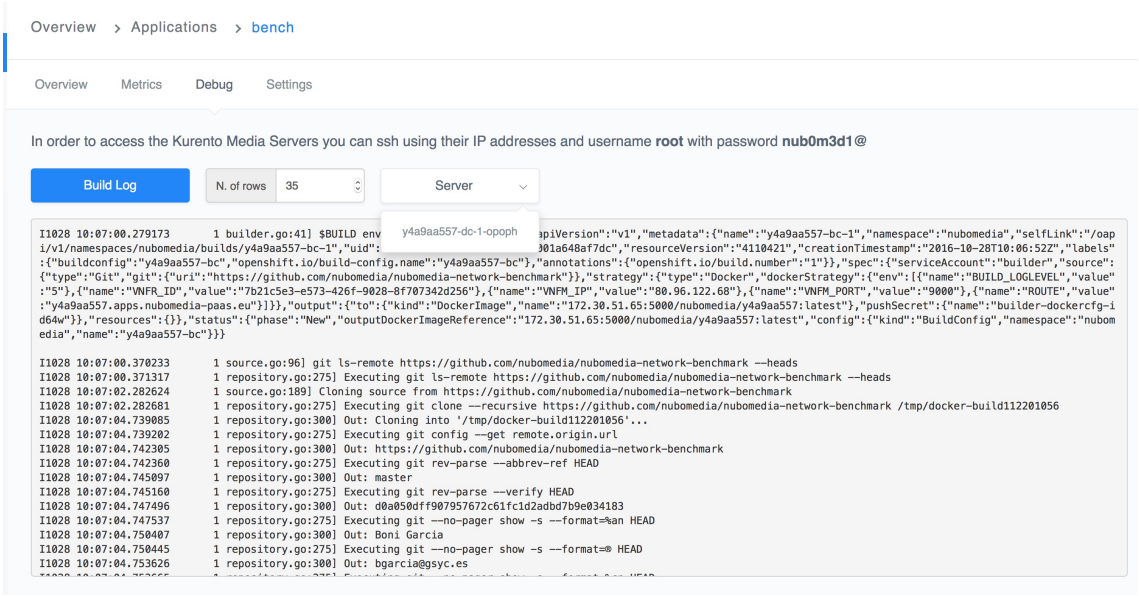


Figure 23 NUBOMEDIA PaaS GUI - Debug section

In settings tab user have access to other application information. All the information about the application is also available in a Json format. All the fields in this tab are split in 4 sections:

4.4.3.2.2 Projects

Projects are basically separated workspace environments. You can combine users with projects in order to have the desired matrix of permissions.

As an admin the projects page gives the user the possibility to view all the projects. The user can filter the projects by project name or id. Also, here we can search for a specific project. We can create or delete a project.

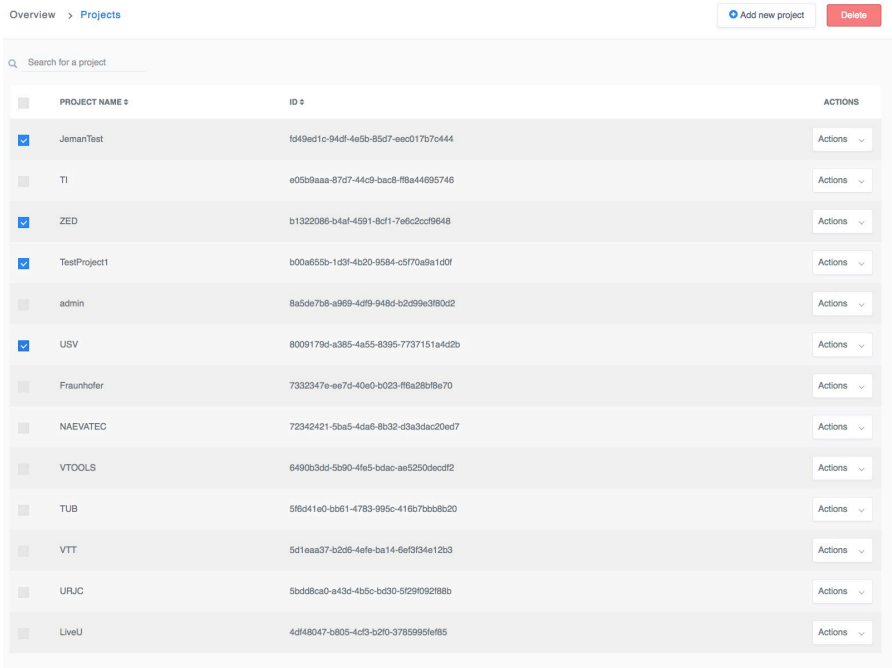


Figure 24 NUBOMEDIA PaaS GUI - Projects

The form for creating a new project is simple. There are only two small sections:

- General

- Name
- Description

- Users
  - User
  - User type (guest, user, admin)

We can add multiple users to a project and set their type from a drop down list.

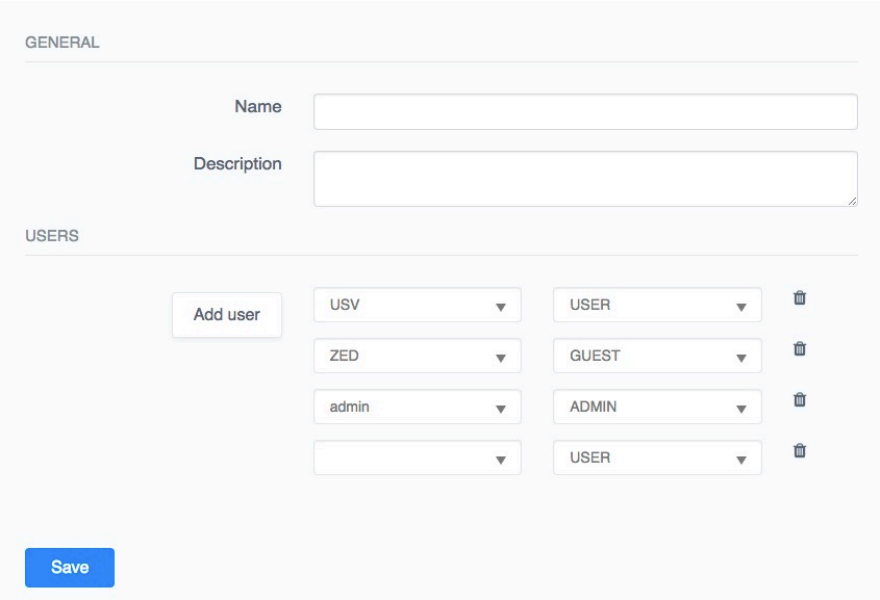


Figure 25 NUBOMEDIA PaaS GUI - Create new project

4.4.3.2.3 Users

There are three types of users on the NUBOMEDIA PaaS.
Guest are users that can only see deployed applications.
Normal users can login into their account and can access information related to their applications and projects. A normal user can't change his password, in order to achieve that he must send an email to NUBOMEDIA admin that will perform that action for him.
Admin users have access to all views of the applications, can perform various application actions on different projects.

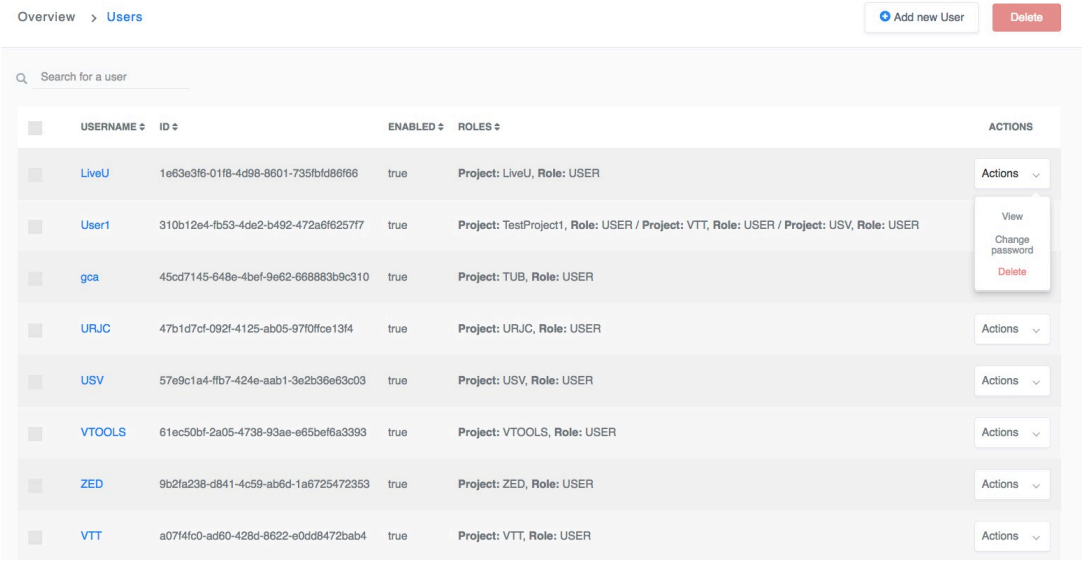


Figure 26 NUBOMEDIA PaaS GUI - Users list

This view is can be accessed only by an admin user. Here an admin can:

- Create a new user

- Delete an existing user
- Delete multiple users
- View user's details
- Change user's password

A specific user can be find using the search form. Also, the users can be filtered by username, id or roles.

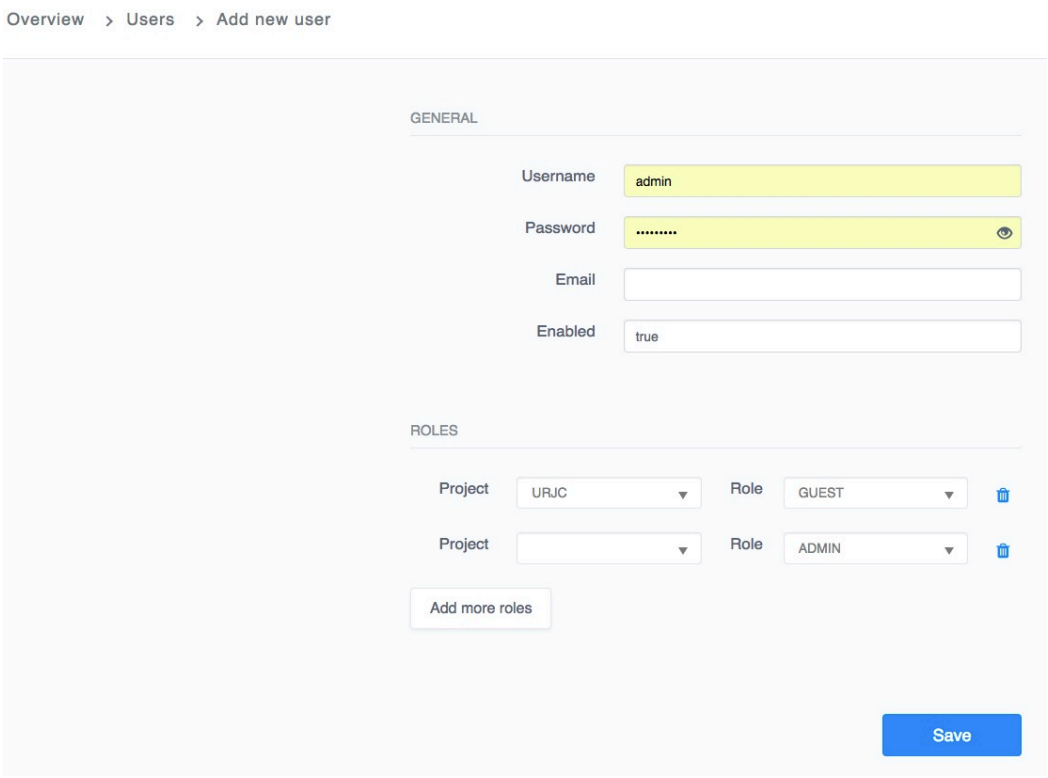


Figure 27 NUBOMEDIA PaaS GUI - Create user form

When creating a new user, the admin has to complete the followings fields:

- Username
- Password
- Email
- Enable
- Roles
    - Project
    - Role

In user details view, the admin user can view specific user information, he can also delete the account or view it in Json format.

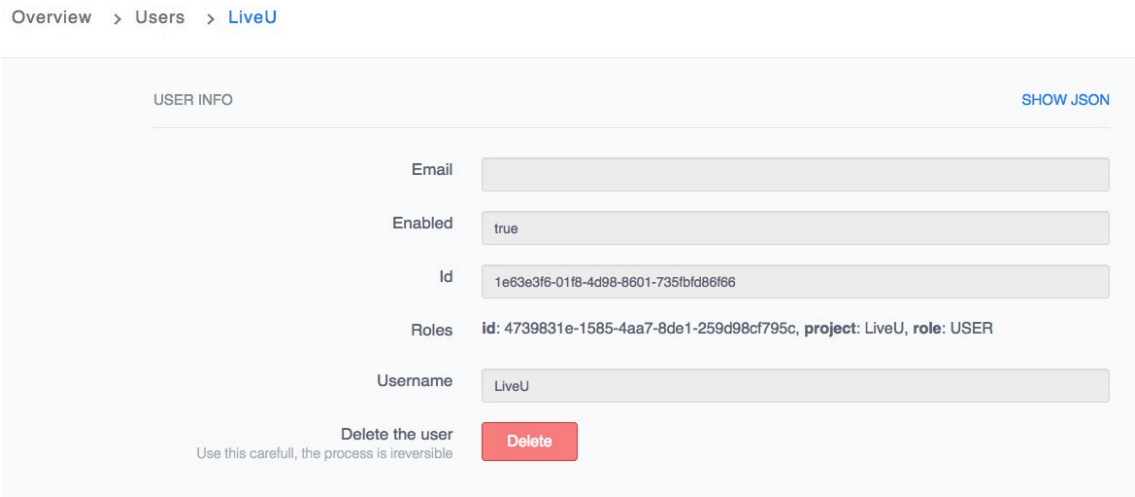


Figure 28 NUBOMEDIA PaaS GUI - User details

4.4.3.2.4 Market

The NUBOMEDIA Market place is meant to gather a list of basic applications that any user on the platform is able to launch and test. Any logged in user is able to add a new application to the market by clicking on the Store app button top right side of the Store view.

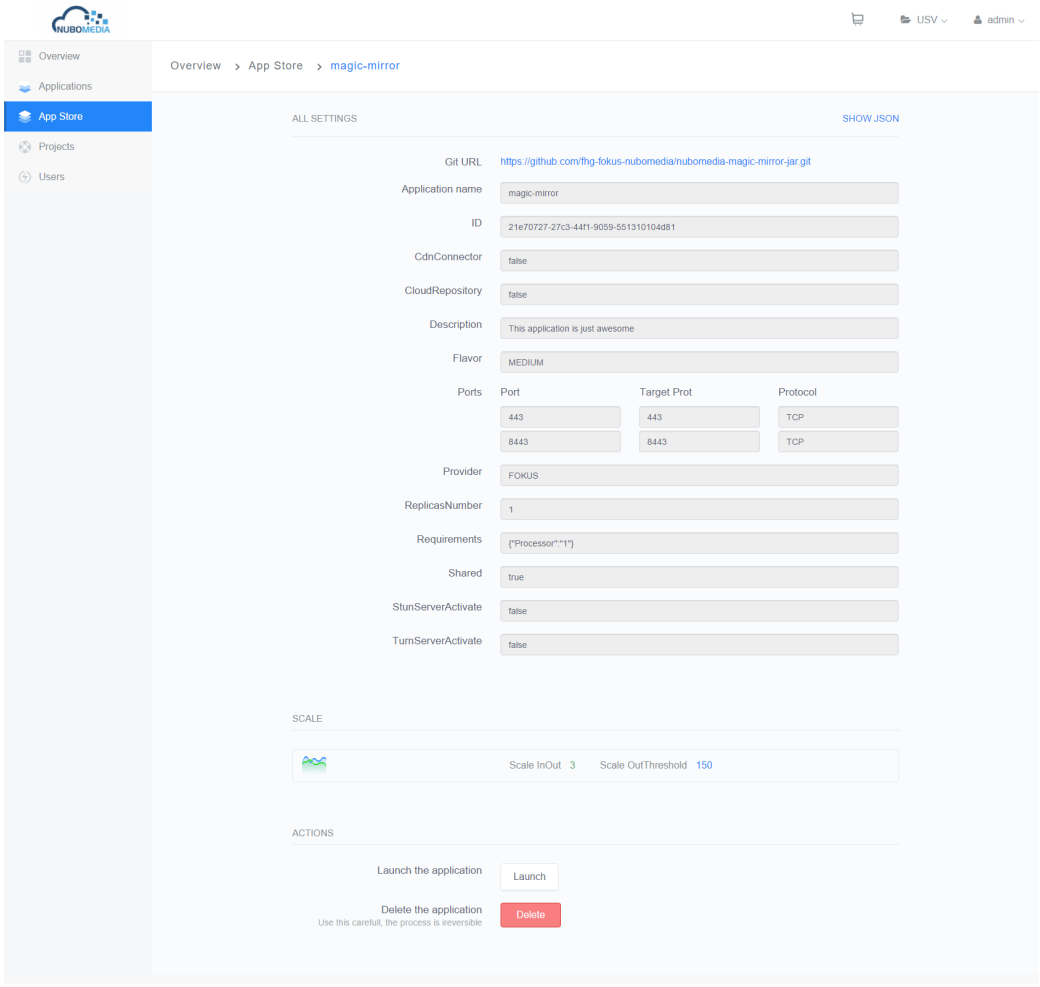


Figure 29 NUBOMEDIA PaaS GUI - Application Store

4.5 NUBOMEDIA Management Tools

The NUBOMEDIA management tools are meant to help monitoring and deploying the NUBOMEDIA platform in an easy and efficient way. For this we researched, designed and prototyped a monitoring system based on Free Open Source tools like LogStash, Kibana, Graphite and Icinga. We then created an autonomous installer using the native APIs of OpenStack that enables the deployment of the entire NUBOMEDIA platform in rapid time without having to know the internals of it.

4.5.1 NUBOMEDIA Autonomous Installer

The Autonomous Installer is meant to deploy a new NUBOMEDIA instance into an IaaS environment based on OpenStack and a PaaS environment based on OpenShift Origin v3.

The internal developers have requested the support for also take into consideration the installation of OpenShift as part of the autonomous installer on installation requirement IST_R1. We started to find a stable way to deploy OpenShift Origin v3. We researched the way to deploy it by using Ansimble scripts available on the official repository but we had no success. We have worked on this path for almost 1 month but we got into problems one after another and we tried to get help from the OpenShift community on their Github issue[4] track repository and Freenode channel #openshift. In the end we concluded that the installation of OpenShift can be a complex work and should be done by experts, and we, in context of the NUBOMEDIA, should have OpenShift credentials as part of initial requirements. The good part of this is that we were contacted by one O'Relly Media producer in order to provide feedback for a book that was planned to be released, the book is now released with the following name: "OpenShift Enterprise for Developers" by Steven Pousty and Grant Shipley.

The OpenStack must be running the Kilo release alongside with the nova-docker plugin for the compute nodes. For this you will first have to uninstall the python-pbr package installed for the nova-compute, then install the version 0.10.1 needed for nova-docker installation. After this you will have to download the latest version of the nova-docker Kilo release that is available here : https://github.com/openstack/nova-docker/releases/tag/kilo-eol . Next you will have to follow the steps for installing and configuring it and then you should update back the python-pbr to version 1.8.1-2.

The PaaS should be running OpenShift Origin v3 version 3.1 with the following adjustment: To relax the security in your cluster so that images are not forced to run as a pre-allocated UID, without granting everyone access to the privileged SCC. In order to make this changes you have to run the following command:

```
oc edit scc restricted
```

This will open you the configuration in edit mode. Here you will have to update the runAsUser parameter in the following way:

```
runAsUser:
  type: MustRunAsRange
```

Having a constantly evolving architecture and components, creating a NUBOMEDIA installer that could install every piece of software from binaries was not feasible. Considering this, during this review period we kept maintaining the NAI functionality

[4] https://github.com/openshift/openshift-ansible/issues/1338

and extended it in order to support deployment of new capabilities that were developed during the review period. Some of the newly capabilities added are the NUBOMEDIA Cloud Repository, the NUBOMEDIA Development GUI and the NUBOMEDIA Market Place.

During this review period, we added support for installation of the entire platform in interactive mode. This functionality would help an expert system administrator installing NUBOMEDIA in an interactive mode, allowing him to input every installation details like MySQL username and password that would be used when creating the database for the NUBOMEDIA PaaS Manager or even configure the public IP addresses that would be assigned to each of the services hosts.

4.5.1.1 NAI Prerequisites

In order to be able to run the NUBOMEDIA Autonomous Installer you have to follow the process flow presented in Figure 30.
You should start by checking that your current python version is 2.7.XX using the following command:

```
python --version
```

Then you should install pip and after that you should install the dependencies using the following commands. When asked for any kind read it first and then confirm:

```
easy_install pip
pip install –r requirements.txt --upgrade
```

If you want to use the docker images for the Kurento Media Server you should have root access to the OpenStack environment in order to install and configure the nova-docker hypervisor and add the patch for it from (https://github.com/usvpublic/nubomedia-nova-docker) on the compute nodes that run docker as a hypervisor. The production ready Docker images are available here : https://hub.docker.com/r/nubomedia/kurento-media-server/ and the development version is available on https://hub.docker.com/r/nubomedia/kurento-media-server-dev/ .

4.5.1.2 Update the configuration file

Before starting the installation, you must first rename the *variables-example.py* to *variables.py* and then gather the required variables:

- **enabled_logging** can take either true or false. If you set this to true it will create a file name installer.log that will give you verbose logging of the installation;
- **iaas_ip** is the OpenStack public IP address, where the endpoint for nova, glance, cinder and neutron are available;
- **auth_url** is the authentication endpoint of Keystone;
- **username** is the username of the OpenStack platform;
- **password** is the password for the previous declared user;
- **tenant_name** is the project name inside OpenStack;
- **glance_endpoint** is the API endpoint of the Glance image service. The list of all endpoints can be found on the “Access and Security > API Access” page of your OpenStack deployment;
- **master_ip, master_user, master_pass, master_key** are credentials that are needed in case you will use the docker image for Kurento Media Server. With the following credentials the installer will connect to all compute nodes running docker as a hypervisor and will pull the latest development and production docker images of KMS;

- **floating_ip_pool** is the name of the external network of OpenStack;
- **private_key** is the name of the Public Key file that will be attached to all the instances started by the NUBOMEDIA Autonomous Installer and then, the NUBOMEDIA Platform;
- **openshift_ip** is the IP address of the OpenShift environment on top of which all the applications will be run;
- **openshift_keystore** represents the Keystore holding the OpenShift SSL certificate. You can create the Keystore by using Portacle.
- **openshift_domain** represents the wildcard domain name that will be used by as URLs for the applications that will be deployed inside OpenShift; For this you will have to define a wildcard DNS record on the IP address of OpenShift;
- **openshift_token** will be used to authenticate the API calls from the PaaS manager to OpenShift API endpoint. In order to optain the token you will have to ssh to the OpenShift instance and then get the list of all secrets available by using the following command : "*oc get secrets"* . Then you will have to type "*oc describe secrets PROJECT-NAME-TOKEN-XXX"* in order to get the token;
- **nubomedia_admin_paas** represents the password for user admin of the NUBOMEDIA PaaS that will be deployed by the Autonomous Installer;
- **use_kurento_on_docker** can take either true or false, representing the choice between KMS on Docker and KMS on KVM;

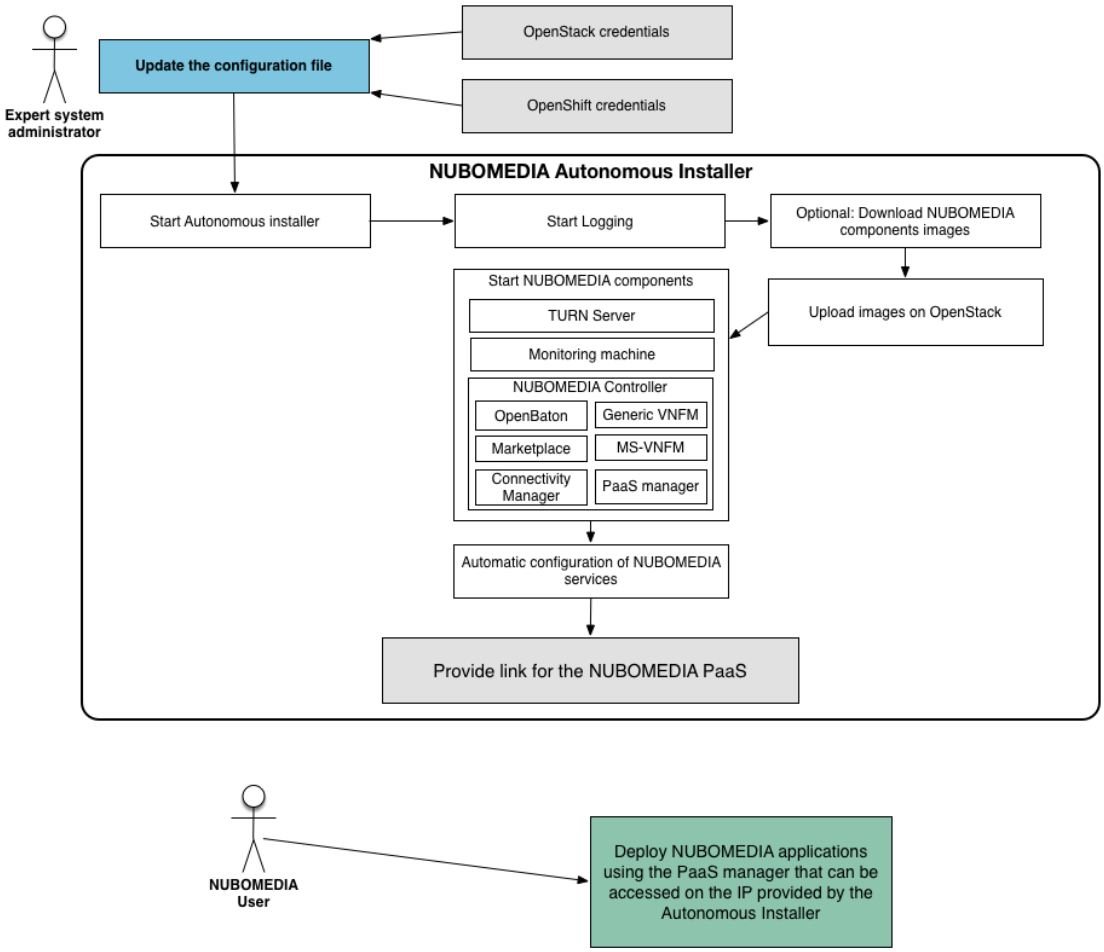


Figure 30 Autonomos Installer - Process flow

4.5.1.3 Start the installer

At this point you should have all the necessary configurations steps ready you can start the installer the actual installation of the NUBOMEDIA platform by running the following command:

```
python main.py
```

In order to check the logs you should use the command tail to monitor the installer.log file.

When the installer has completed you should receive a success message indicating the URL for the PaaS GUI where you can login with user admin and the password configured on the variables.py file, nubomedia_admin_paas.

4.5.2 NUBOMEDIA Monitoring Tools

Its main aim is to monitor the distributed execution of the NUBOMEDIA computing units: Media Server instances. During the previous review period we developed a Java based monitoring system that was then evolved on this review period by adding new capabilities to it for collecting WebRTC metrics from the media-servers and pushing them to Graphite on the monitoring infrastructure.

The monitoring capabilities are offered to the orchestration layer of the platform using a RESTful API. Using these metrics, the VNFM can take actions like scaling out or in.

Considering the technical requirements KMS_R1 coming from the internal developers we updated the monitoring tools by adding WebRTC metrics like:

- media elements represent the number of unit performing a specific action on a media stream;
- media pipelines represent the number of active pipelines inside the media server;
- inbound jitter represents the time difference between package arrival measured in seconds;
- inbound byte count represents the number of bytes received, not including headers or padding;
- inbound packet lost represents the total number of RTP packet lost;
- inbound delta nacks represents the number of not acknowledged packets;
- inbound delta plis represents the Picture Loss Indication at inbound;
- inbound fraction lost represents the average packet Fraction Lost (packet lost / total packets ratio);
- outbound byte count represents the total number of byte count for the media-server;
- outbound Rtt represents the Round Trip Time and it's measured in seconds;
- outbound delta plis represents the Picture Loss Indication at outbound;
- outbound delta nacks represents the number of not acknowledged packets at outbound;
- outbound target bitrate is the current target bitrate configured calculated in bits / second;

On the monitoring system, we considered the developer requirement DGB_R3 and implemented deployed Kibana 4 to help us gather the logs from all the media-server and then provide a RESTful API to the NUBOMEDIA PaaS GUI in order to offer developers a better way to diagnose any problem that their application may have. We also wanted to provide the possibility to filter the logs by different log types like debug,

error or warning thing that can be helpful when debugging media-server issues. Kibana also offers the possibility filter logs using regular expressions which can be helpful for more in depth analysis.

Figure 31 presents logs coming directly from one media-server.

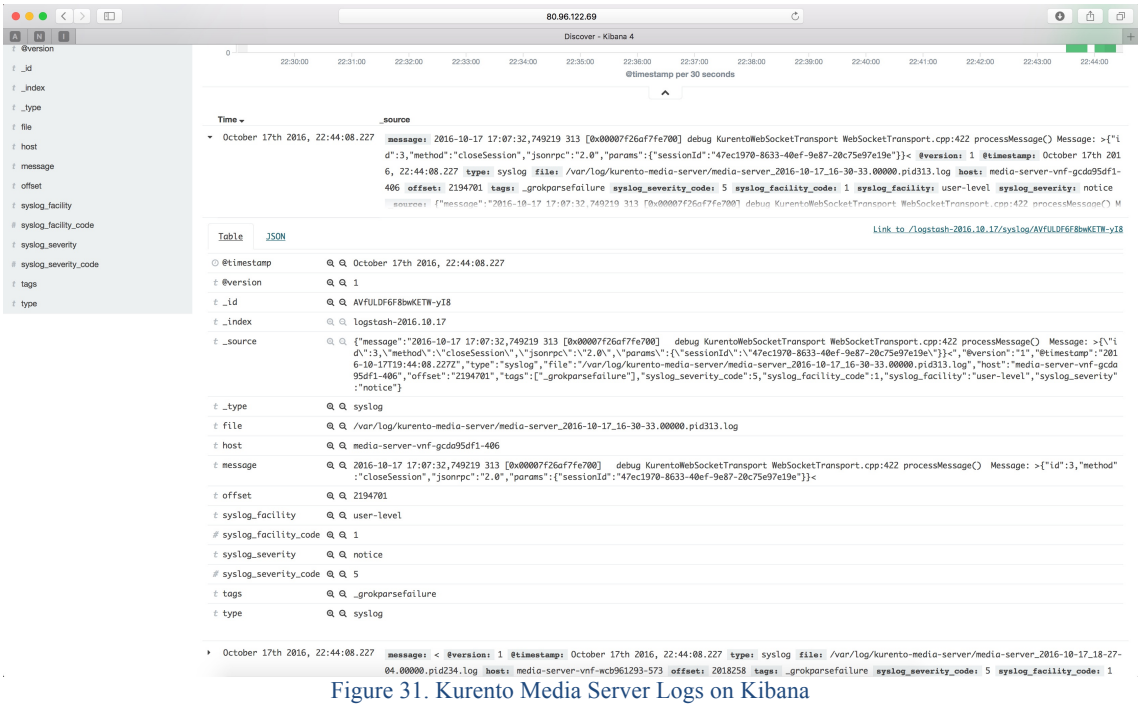


Figure 31. Kurento Media Server Logs on Kibana

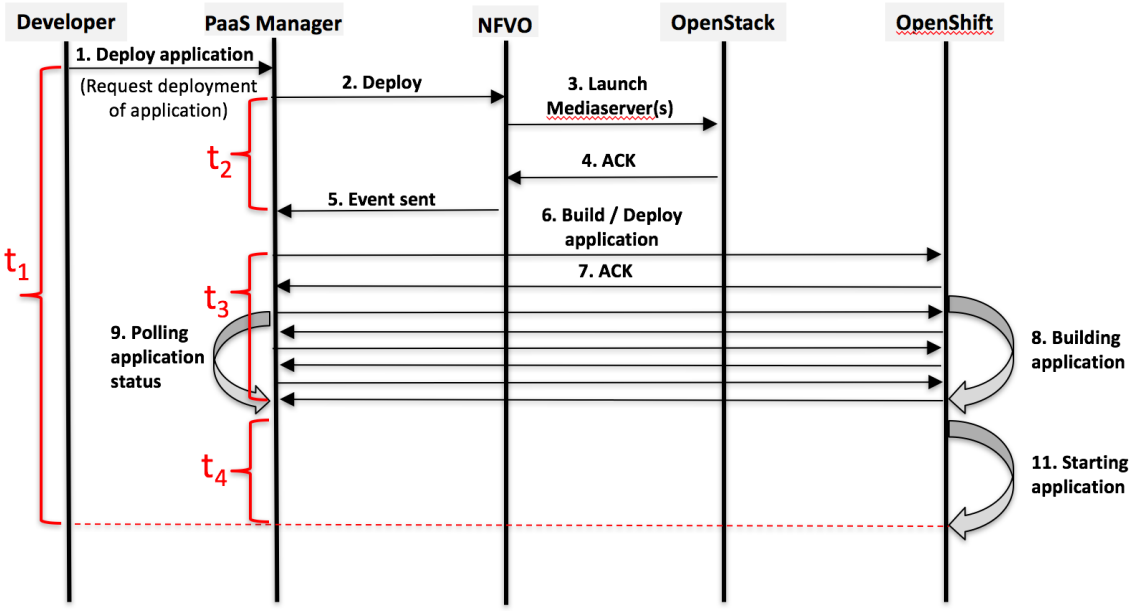
5 Evaluation

In this section are presented the results of the evaluation of some operations executed by the NUBOMEDIA PaaS. Some sequence diagrams for the most important procedures of the NUBOMEDIA components are provided for simplifying the understanding the different steps executed. In particular, the deployment of applications and the scaling out procedure of media servers.

5.1 Deployment of an application

In this evaluation scenario it is measured how much time it takes to deploy an application in different ways. As depicted by Sequence diagram 1 the deployment of an application involves several components and includes basically three main steps. Hence, three time intervals are considered in order to measure the latency of the management operations. In the end five measurements per test case were executed to build the average values of certain steps. In both cases it is used the same application. Differences are only in the Dockerfile which defines the way of preparing and configuring the container for the application execution and the type of application where we distinguish between a source code version and a binary version.

Time interval t_2 depicts the time which is needed for allocating and configuring the virtual containers. Time interval t_3 covers the step where the PaaS Manager requests OpenShift for building and deploying the application. After the initial request, a polling mechanism starts in order to check the current application status. Interval t_3 ends once the application is in status RUNNING. The final execution of running the application, so that the application is also available for the consumer, depends on the type of application. Here we differentiate between applications where the source code must be compiled or an application which provides already a pre-compiled binary version. Using option 2 avoids the time consumption where the application must be compiled. Therefore, in the following the two different types of application are used in this evaluation part. This time interval is depicted by t_4. The entire time covering all three steps is t_2



Sequence diagram 1. Sequence diagram depicting time intervals measured

As depicted by Table 2 the overall time for deploying an application is around 5 minutes. Most of the time is consumed by building and deploying the application itself which requires the major time with 4:25 minutes. Minor time is needed for the deployment of the virtual resources on OpenStack. In fact, this takes around 29 seconds only whereas the start of the application takes even less with 9 seconds.

	Deploy virtual resources t_2	Building/deploying application t_3	Starting application t_4	**Total** t_1
1	00:25	04:25	00:07	**04:57**
2	00:30	04:25	00:14	**04:57**
3	00:30	04:25	00:10	**05:05**
4	00:30	04:20	00:05	**04:55**
5	00:30	04:30	00:10	**05:10**
avg	**00:29**	**04:25**	**00:09**	**05:03**

Table 2. Measurements of the deployment of an application where the code is present as binary

Table 3 depicts the measurements which were taken by deploying an application including also the compilation of its source code. Hence, the application must be compiled and this takes obviously more time than the case shown where the precompiled jar archive is already provided. In fact, this step lasts now 2:37 minutes in average. The build and deployment of the application takes 3:48 minutes. The time is reduced here due to the fact that less steps are required for preparing the image stream and containers. Finally, the source code version of the application requires around 7 minutes which is two minutes more than the binary version.

	Deploy virtual resources t_2	Building/deploying application t_3	Starting application t_4	**Total** t_1
1	00:30	03:35	02:42	**06:47**
2	00:35	03:55	02:53	**07:23**
3	00:30	03:55	02:42	**07:07**
4	00:40	03:40	02:07	**06:27**
5	00:35	03:55	02:44	**07:14**
avg	**00:34**	**03:48**	**02:37**	**06:59**

Table 3. Measurements of the deployment of an application where the code is present as source code

As shown, the deployment of an application depends strongly on the application provided in the meaning of how it is provided. The two main options provided are:

- The application source code is precompiled
- The application has to be compiled on demand when starting it

To summarize, deployment times may also vary and be improved based on different aspects. This may affect all the time intervals used before. The deployment of virtual resources depends on the cloud infrastructure. In our test cases we used container based deployment of media server instances. For instance, using a hypervisor based deployment will increase the time needed for launching the virtual machines.
The results of t_3 might be affected by changing the base image used inside the containers running in OpenShift. In the given scenarios prepared images were used which were already present in the local docker registry of OpenShift. Other cases might

be that the base image must be down/up-loaded and registered or even the creation of an image from the scratch.
As already seen, time interval t_4 depends on the way the application is started. It was shown that the precompiled version of the application required much less time for starting the application whereas the source code version spend a lot of time for compiling the application before executing it. But this is only one scenario where we used the same application. Other applications may take less or more time for compiling the source code or even for executing the application until it is available for the consumer.

5.2 Deployment of an application

In the following it is evaluated the needed time for scaling out new components. In the first scenario, it is shown how long it takes to scale out components without using the proposed pool mechanism whereas the section after evaluates the times which are needed by using the pool mechanism.

As depicted by both figures (see Figure 32 and Figure 33) the detection of the need to scale varies from 56ms to 78ms, where the gap is basically caused by the response time of the Monitoring System. The next step of requesting the Decision-maker is around 27ms and almost the same for both. The main difference comes when executing the scaling actions. Obviously, the reason is the number of components to be scaled-out. Scaling-out a single component takes 16570ms whereas scaling-out five components takes 83372ms. In fact, when scaling-out 5 components in a row, each scaling-out operation takes around 16674ms in average.

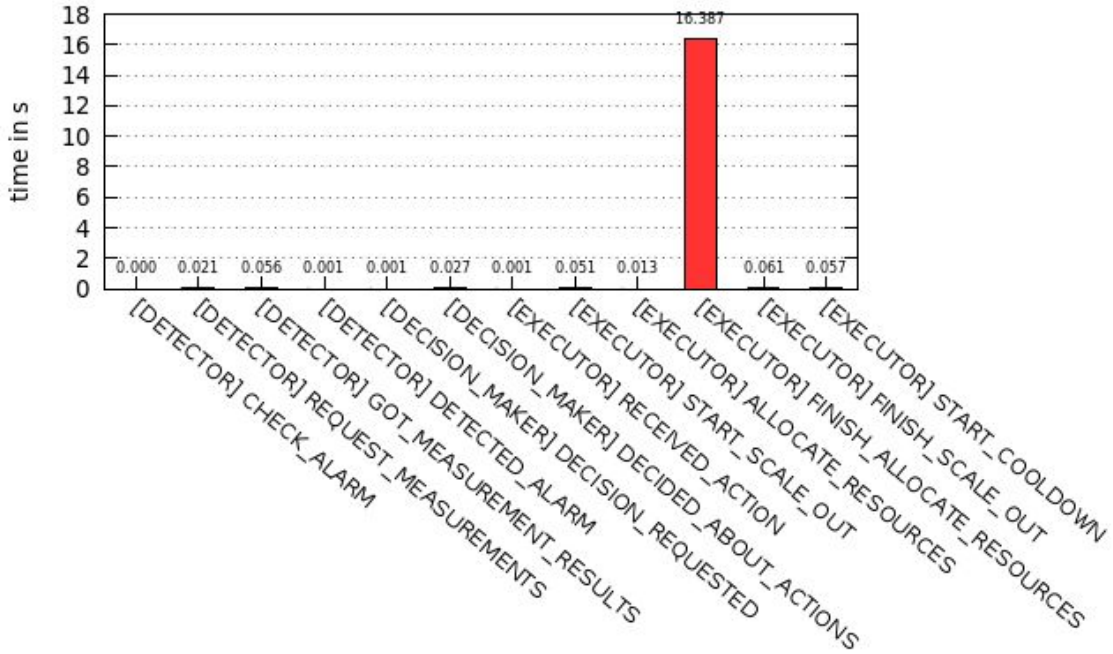


Figure 32. Timings scaling-out 1 instance without pool mechanism

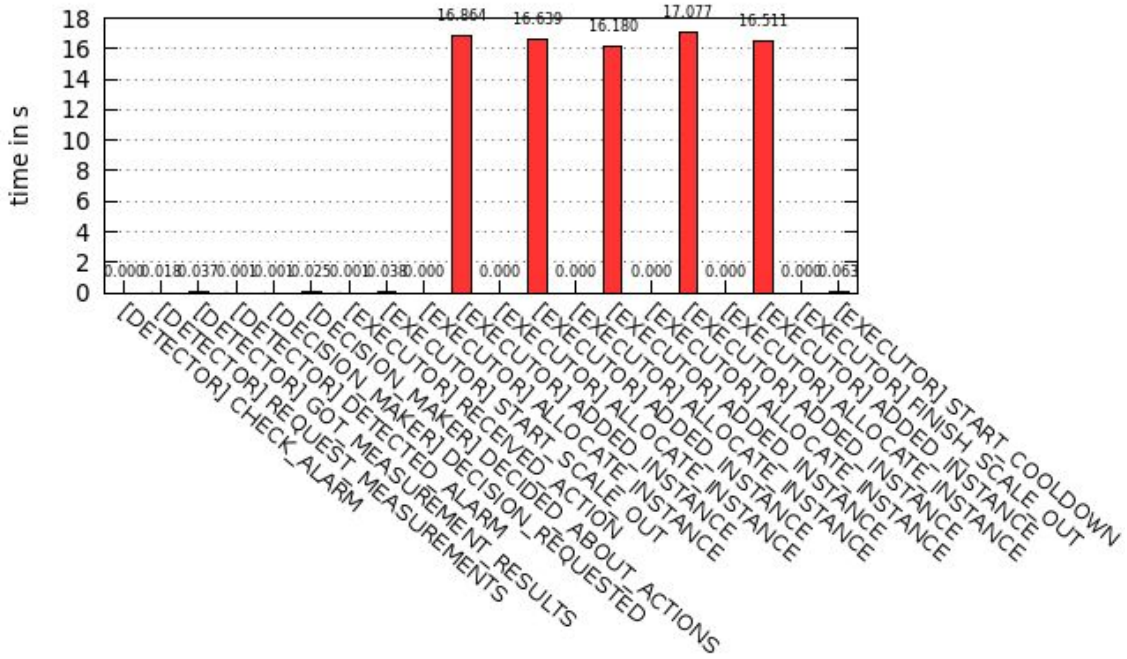


Figure 33. Timings scaling-out 5 instances without pool mechanism

Concluding, in this approach without the pool mechanism the time of detecting the need to scale and deciding about actions is very short. Both operations together need around 100ms whereas the scaling operation itself needs roughly 16000ms to 17000ms for scaling-out a single component. In case of scaling-out a single component it takes 99,36% of the overall time from detecting to finish scaling. In the scenario of scaling-out five components in a row the scaling takes even 99,9% of the entire time. As seen, scaling-out five components in a row needs five times more and is caused by scaling-out step-by-step. Obviously, the time needed for scaling-out 5 components can be improved by parallelizing the scaling operations. Another promising approach is the usage of the proposed pool mechanism that is evaluated in the next section.

The following section covers the evaluation of the proposed pool mechanism. Two scenarios were taken into account equally to the two scenarios described before. One test covers the scaling-out of a single component and the other scenario scales out five components step-by-step. Mainly, the pool mechanism was introduced to decrease the required time needed for launching new components. As seen in the previous section the time for launching new components takes most of the time of the entire scaling process from detecting need to scale, over making scaling decision and finally the scaling operation itself, and therefore, it decreases the reaction time of the AutoScaling System dramatically in order to provide new instances on demand.

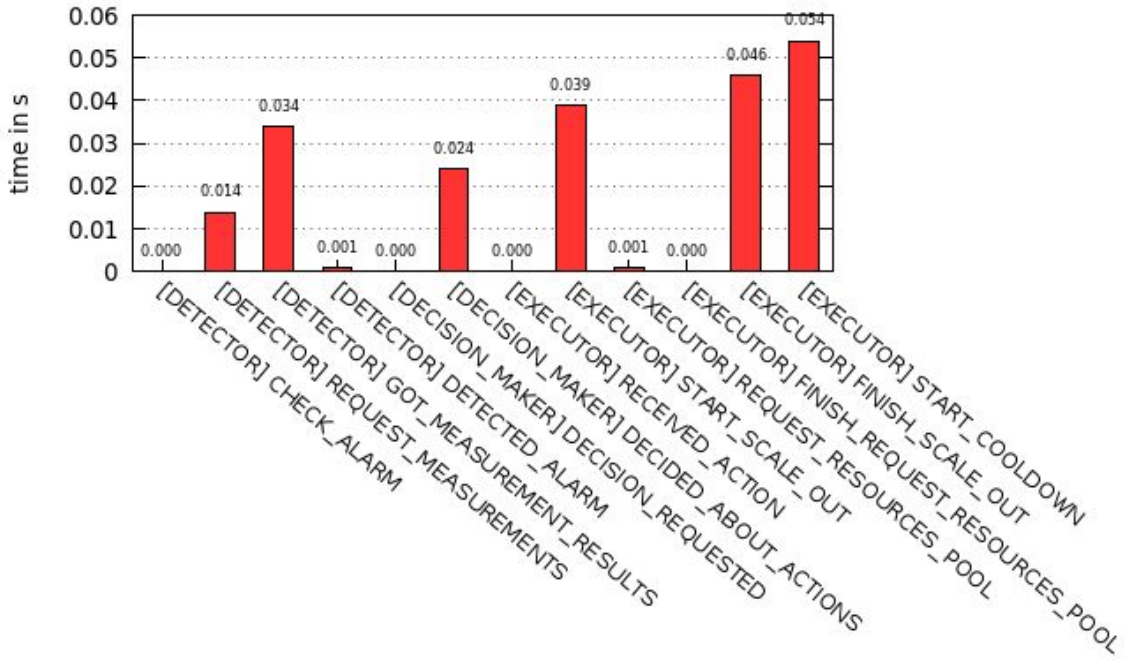


Figure 34. Timings scaling-out 1 instance using pool mechanism

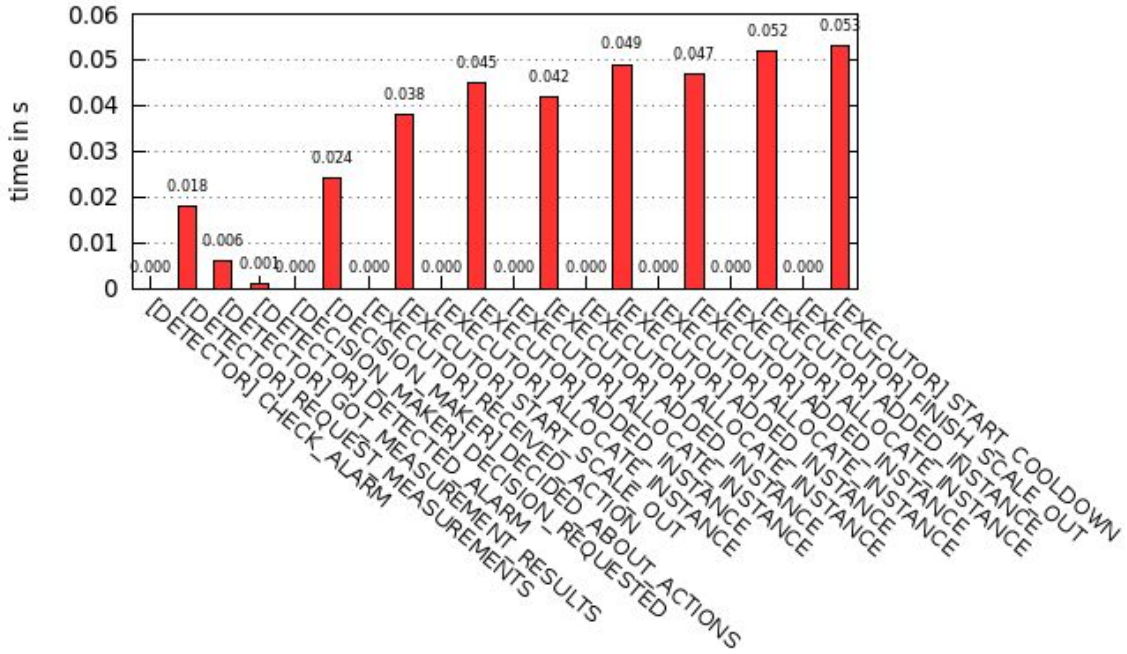


Figure 35. Timings scaling-out 5 instances using pool mechanism

Figure 34 shows the measurements when scaling-out a single component and Figure 35 shows the case of scaling-out five components. Similar to the scenario before without the pool mechanism, in both scenarios the time of detecting and decision-making takes around 24ms. But in contrast to the previous test the Executor does not request the VIM for allocating new resources on demand but it requests the Pool Manager in order to get a prepared VNF component. As depicted in Figure 34 in case of adding a single component, the whole scaling operation needs 140ms. Figure 35 reveals that scaling-out five components in a row takes 350ms only.

Compared with the time needed from detecting the need to scale to finish the scaling, the scaling procedure itself in case of adding a single component takes 140ms and

65,7% of the time whereas scaling-out five components takes 350ms and 87,93%. Considering the time needed in the scenarios without the pool mechanism, scaling-out a single component and five components in a row is improved dramatically. In fact, the scaling-out action to add a new component takes this time 0,85% of the time that was needed without the pool mechanism. Scaling-out five components take this time 0,42% only compared with the time in the previous scenario. Additionally, after each scaling operation the VNFR is updated by requesting the NFVO and this time is also included in each scaling action that takes most of the time.

6 Conclusion

This deliverable presented the latest extensions implemented in the context of WP3 for supporting additional requirements coming from application developers. Most of the requirements have been supported without applying any modification to the NUBOMEDIA functional architecture.

Major changes have been applied mainly in the context of the NUBOMEDIA PaaS Manager, being the entry point for application developers. A new version of its API has been developed and implemented, with also a new refactoring of the PaaS GUI exposing more functionalities for controlling the lifecycle of the application.

The good evaluation results obtained also demonstrates that the design decisions were correctly taken, and application developers can benefit from the NUBOMEDIA PaaS for the management of their application lifecycle. They can deploy and expose to end users a complete elastic application without requiring any knowledge on the infrastructure itself.

References

[1] 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

[2] D2.4.2 NUBOMEDIA Architecture v2

[3] D2.4.3 NUBOMEDIA Architecture v3

[4] Open Baton: An open source Network Function Virtualisation Orchestrator (NFVO) fully compliant with the ETSI NFV MANO specification http://openbaton.github.io/

[5] D2.2.2: State-of-the-art revision document v2

[6] Openstack: Open source software for creating public and private clouds. See http://www.openstack.org/.

[7] D3.3.1: NUBOMEDIA Cross-Layer Connectivity Manager v1, https://www.nubomedia.eu/sites/default/deliverables/WP3/D3.3.1_CrossLayerConnectivityManager_V1.0_27-01-2015_FINAL-PC.pdf

[8] https://www.openstack.org/assets/pdf-downloads/Containers-and-OpenStack.pdf

[9] Graphite: An open source graphing and time-series database for storing realtime metrics https://graphite.readthedocs.org/en/latest/

[10] Backstop is an open source REST API to submit data to Graphite https://github.com/obfuscurity/backstop

[11] Logstash is an opensource tool to centralize and process logs https://www.elastic.co/products/logstash

[12] Icinga is an opensource tool based on Nagios and improved by open source community for monitoring systems https://www.icinga.org

[13] D3.2.1: Cloud Repository

[14] http://cloudinit.readthedocs.org/en/latest/

[15] https://github.com/tub-nubomedia/cloud-repository-scripts

[16] https://docs.mongodb.org/ecosystem/drivers/

[17] Sefraoui, O.; Aissaoui, M.; Eleuldj, M., Management platform for Cloud Computing, International Conference on Technologies and Applications (CloudTech), (2015):1-5.

[18] ETSI TR 103 126 V1.1.1 (2012-11), CLOUD; Cloud private-sector user recommendations, Technical Report (TR), ETSI Technical Committee CLOUD (CLOUD), (2012).