D6.1.2	
Version	1.4
Author	USV
Dissemination	PU
Date	31/01/2016
Status	Final

D6.1.2: NUBOMEDIA Testbed and simulated load validation 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 2017-01-31
Project type:	STREP
Project reference:	610576
Project web page:	http://www.nubomedia.eu
Work package	WP6: Demonstrator
WP leader	Constantin Filote (USV)
Deliverable nature:	Demonstrator
Lead editor:	Alin Calinciuc (USV)
Planned delivery date	31/01/2016
Actual delivery date	31/01/2016
Keywords	Hardware infrastructure, testbed, Gitlab, Jenkins, continuous integration, nova-docker

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)
Cristian Spoiala (USV)
Constantin Filote (USV)

Internal Reviewer(s):

Giuseppe Carella (TUB)

Version History

Version	Date	Authors	Comments
1.0	18/12/2015	Cristian Spoiala (USV)	Added more details about Jenkins jobs, testplan.
1.1	15/01/2016	Alin Calinciuc (USV)	Restructured the delivery
1.2	23/01/2016	Cristian Spoiala (USV)	Added details for integration tests and developers guidelines
1.3	27/01/2016	Giuseppe Carella (TUB)	Internal review
1.4	30/01/2016	Alin Calinciuc (USV)	Final review

Table of contents

List of Figures:

Acronyms and abbreviations:

IaaS Infrastructure as a Service
CI Continuous Integration
ECC Error-Correcting Code
SAS Serial Attached SCSI
UPS Uninterruptible Power Supply
NTP Network Time Protocol
API Application Programming Interface
RDO RPM Distribution of OpenStack
PESQ Perceptual Evaluation of Speech Quality

1 Executive summary

NUBOMEDIA Testbed and simulated load validation v2 (D6.1.2) is a deliverable that provides a mature physical testbed where NUBOMEDIA instances are being deployed and validated. To successfully meet the requirement of Metric 1.1 General validity of architecture, USV provisioned a large testbed from 120 cores to 200 cores for simulated load tests.

The deliverable includes the documentation for the operation of the instance, based on virtual machines (VMs) and Docker machines (DMs). The choice of the Docker machines for NUBOMEDIA instances was decided based on the results of benchmark tests that are accurately described in D3.2 deliverable.

According to B1.1.4 Measuring degree of achievement of project objectives from DoW for validation of Objective 1, Metric 1.2 General validity of admin tools we present the Autonomus Installer of NUBOMEDIA into IaaS. We consider that the metric of this deliverable is fully successfully met due to the appropriate software tools that we developed, the time needed for NUBOMEDIA platform to be deployed and the creation of specific documentation.

This document also provides an overview of the NUBOMEDIA Testbed components. We will be focusing next on describing the hardware, software infrastructure, integration tests and the description of the procedure for developing software on top of the NUBOMEDIA platform, so that the partners can integrate the prototypes created during the RTD tasks into the testbed. Furthermore, we will cover the CI (Continuous Integration) tests and will describe the benchmark tests that we started to develop and run on top of the NUBOMEDIA testbed.

2 Hardware infrastructure

This section describes the current hardware infrastructure for OpenStack IaaS [1] at HPC DC of Stefan cel Mare University of Suceava (USV). This facility is being used for hosting the NUBOMEDIA testbed.

Hardware infrastructure is built around multiple units of IBM BladeCenter H on 2 x 42U racks.

Since the last release, we installed and configured OpenStack Kilo on 25 compute nodes having 200 CPUs and 400GB RAM with 1GB backbone networking, and 1GBps internet connectivity with 32 public IPs available for floating IP allocation.

Figure 1 USV Cluster for Testbed

2.1 Network connectivity

Internal connectivity inside a BladeCenter is achieved using a two Layer 2 IBM Connectivity Modules with 802.11Q VLAN tagging, and six external 1GbE ports. The connectivity with storage, external network and Internet is made using a Layer 2 Managed switch SMC8126L2 with 802.11Q VLAN tagging.

The Router running Red Hat Linux provides the routing for all 32 public IP addresses that will be configured as floating IPs on OpenStack. Concerning the protection layer (Firewall), the USV HPC DC has installed a firewall based on Red Hat Linux with 2 x Intel Xeon 3450 at 3 GHz, 8 GB RAM, 2 x 1 Gbit Ethernet. Firewall delivers an integrated family of applications that simplify and consolidate the network and security products that USV needs at the network gateway:

- Integrated with a common GUI, logging & reporting;
- Designed to run Intel/AMD hardware;
- Web Filter; Spam Blocker;
- Spyware Blocker;
- Protocol Control;
- Virus Blocker;
- Phish Blocker;
- Intrusion Prevention;
- Attack Blocker;
- Firewall, OpenVPN;
- Daily, weekly & monthly Reports for each application, in PDF or HTML format;
- Routing & QoS.

The connectivity capacity available is:

- 1 Gbps connectivity for the backbone = 1 Gbps connectivity for the backbone[1];
- (1 Gbps & 100 Mbps) Internet connectivity for end-users > 100 Mbps Internet connectivity for end-users;
- Firewall to ensure security;
- 32 (RoEduNet IPv4) Public IPs available for assignment.

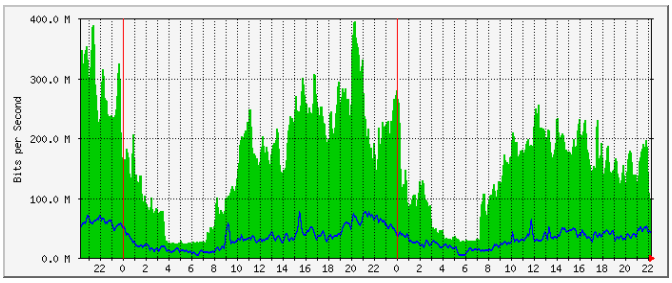

Figure 2 USV Internet Bandwitdh

2.2 Compute machines

The testbed is made up of 25 compute nodes, summing up 200 CPU cores and 400GB RAM. Compute nodes have either Intel Xeon or AMD Opteron processors. The configuration used in the testbed is one of the following:

AMD Opteron configuration:

- CPU: 2 x AMD Opteron Quad Core 2376 2.3GHz;
- RAM: 16GB PC2-5300 CL5 ECC DDR2 667MHz;
- LOCAL DISK: IBM 147GB SAS 10K HDD;
- Connectivity: 2 x Gigabit Ethernet cards.

Intel Xeon configuration:

- CPU: 2 x Intel(R) Xeon(R) E5345 @ 2.33GHz;
- RAM: 16GB PC2-5300 CL5 ECC DDR2 667MHz;
- LOCAL DISK: IBM 147GB SAS 10K HDD;
- Connectivity: 2 x Gigabit Ethernet cards.

2.3 Management machine

The management machine for NUBOMEDIA that hosts Jenkins CI tool, Git code repository running Gitlab, and the autonomous installer, is running on the following configuration:

- CPU: 2 x Intel Xeon Quad Core E5345 2.33GHz;
- RAM: 4GB ECC;
- Local Disk: 2 x IBM 73.4GB SAS HDD;
- Connectivity: 2 x Gigabit Ethernet cards.

[1] http://www.usv.ro/trafic/analize/192.168.100.100_10101.html

2.4 Storage system

The mass storage is provided by an IBM DS4700 Express Model 70 Storage. The available capacity is 2TB in RAID 10. On the storage system, we are hosting all NUBOMEDIA images that are required to be deployed in order to start a new NUBOMEDIA instance. Furthermore, we are keeping regular backups of the management machine.

2.5 Auxiliary Units

- UPS: each rack has 2 x IBM APC UPS7500XHV UPS.
- Cooling: for cooling purposes, servers' room has 3 x enterprise AC unit 20000 BTU.

3 Setup testbed

This section focuses on providing a description on how to replicate NUBOMEDIA on a different setup. We will describe how an IaaS based on OpenStack should be installed and configured in order to support the deployment of NUBOMEDIA platform using the Autonomous Installer developed in WP3.

3.1 Setup OpenStack with RDO

To install OpenStack, multiple scenarios and methods are available, but the most important tools for deploying OpenStack are RDO from Red Hat and Mirantis Fuel. We describe their different advantages and disadvantages, next:

- **Mirantis Fuel**
  - Advantages:
    - One of the advantages of Fuel is that it comes with several pre-built deployment configurations that you can use to immediately build your own OpenStack cloud infrastructure [8];
    - Paid support offered by a professional OpenStack oriented enterprise;
    - Time to deploy smaller than required for RDO.
  - Disadvantages:
    - Few possibilities for adjusting the configuration or adding new capabilities on top of it;
    - Updates most of the time delayed.
- **Red Hat OpenStack**
  - Advantages:
    - It has an open source version, RDO, that can run production grade IaaS environments;
    - Red Hat is offering support from the Linux kernel and the KVM hypervisor to the top level OpenStack project components;
    - Adding new capabilities is easier because Red Hat offers multiple already packaged OpenStack projects, a full list of them can be found on the Projects in RDO page[2];
    - Customization is easier because you have low level access to all OpenStack components.
  - Disadvantages:
    - It requires more time and effort for deployment and maintenance;
    - Upgrades are done most of the time using custom scripts.

Considering this we decided that it will be best to have installed OpenStack using RDO tool in order to allow the customization of the IaaS environment in coherence with the NUBOMEDIA platform needs.

In this section, we describe the method that we used to install OpenStack Kilo on our testbed which can be replicated to create a similar IaaS. Deploying RDO is an easy process. Setting up an OpenStack cloud takes approximately 15 minutes. It can be as short as 3 steps if you want to deploy it on a single server, but if you want to deploy it

[2] https://www.rdoproject.org/rdo/projectsinrdo/

on multiple nodes, it can take longer, because you will need to configure the physical machines and networking equipment to meet the RDO requirements. The configuration file for the RDO packstack tool should also be customized. The deployment script by RDO is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported license [7].

3.1.1 Software prerequisites:

For installing OpenStack Kilo RDO, you will need a RHEL-based Linux distribution, such as CentOS, Scientific Linux, or Fedora 22 or later. For the NUBOMEDIA testbed, we used CentOS 7 x86_64.

3.1.2 Hardware prerequisites:

According to the OpenStack architecture example[3] found on the OpenStack.org, it is recommended to use a machine with at least 2GB of RAM, and hardware virtualization extension with at least 1 network adapter for single node deployment. For multi-node deployment, at least two network adapters are required.

For multi-node deployment, you will also need a Layer 2 Switch that supports 802.11Q VLANs (VLAN tagging), if you are using VLAN tagging for networking. But with the new Kilo release, we can use VXLANs for networking meaning that we can deploy OpenStack networking on top of any regular switch that has no management.

3.1.3 Operating system preparation for the master node

The master node should have at least 4GB of RAM, 2CPUs and 100GB of disk space for storing all the images on the glance-registry. On the master node, we are planning to run the following OpenStack services:

- o nova-scheduler service, to allocate VMs to the compute nodes;
- o cinder-scheduler service, to allocate block storage on the compute nodes;
- o glance-registry service, to manage the KVM and Docker images;
- o neutron-server service, to manage the network inside the instances;
- o heat-api service, to create and orchestrate services on OpenStack;
- o keystone service, to manage the identity inside OpenStack.

[3] http://docs.openstack.org/liberty/install-guide-rdo/overview.html#example-architecture

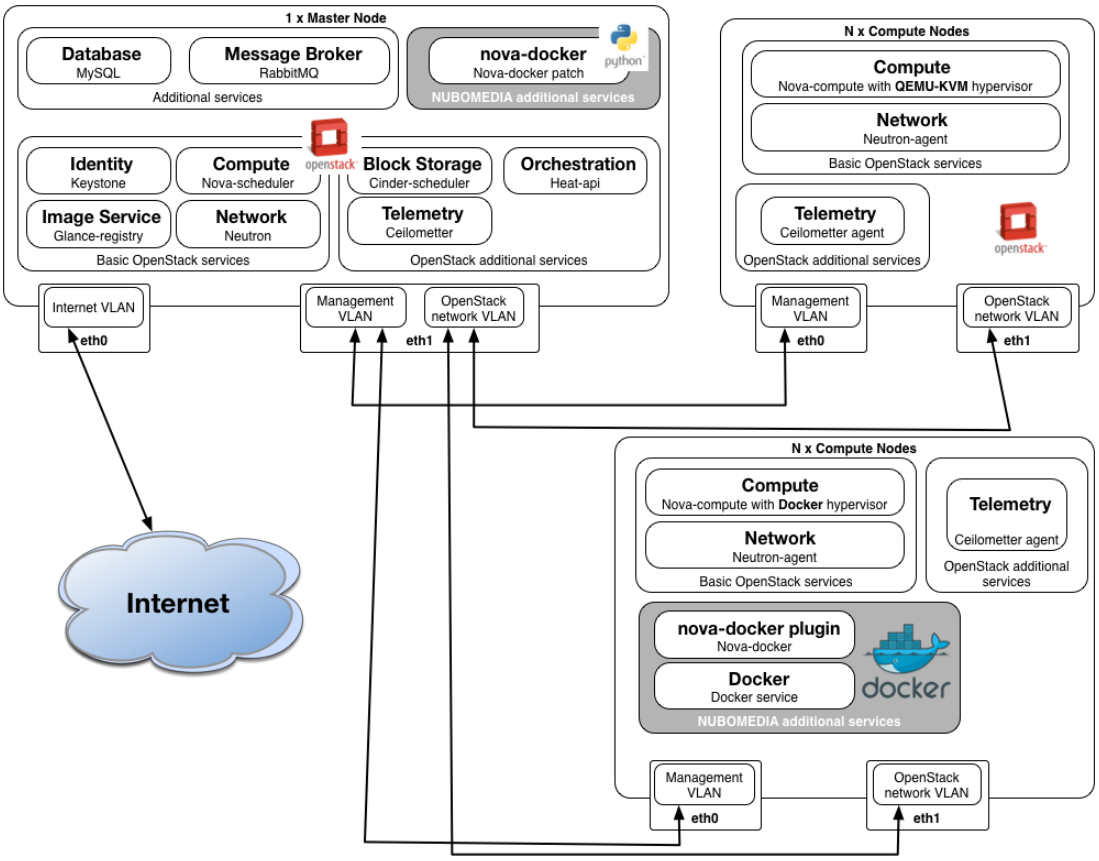

Figure 3 Testbed IaaS architecture

Figure 3 presents a high level overview of the underlying services that are running behind the IaaS testbed.

Next, we will present the specific steps that are needed in order to prepare the master node:

- You will first need to add RDO repositories:

```
yum install -y http://rdo.fedorapeople.org/rdo-release.rpm
```

- You will need to update your current packages using:

```
yum update -y
```

- Some basic tools should be installed

```
yum install net-tools gcc python-devel nano bash-completion -y
```

- Then you need to enable ssh key login:

```
cd ~
mkdir .ssh
chmod 700 .ssh
cd .ssh
nano -w authorized_keys # here you should add your public key
chmod 600 authorized_keys
restorecon -R -v /root/.ssh
```

- Disable selinux or set it in permissive mode (if there is a reason not to have it in enforcing mode).
  In file: /etc/selinux/config edit:

```
SELINUX=permissive
```

Next, if you do not want to reboot the system, you should edit:

```
setenforce 0
```

If you have previously disabled SELinux, you need to re-label the filesystem, since it does not happen for new files, and failing to re-label will likely cause many false positive issues. The easiest way is to do the following as root:

```
touch /.autorelabel
reboot
```

- After this, you should install NTP client on all servers because all servers should have date in sync with each other:

```
timedatectl set-timezone Europe/Bucharest
yum install ntp –y
chkconfig ntpd on
ntpdate pool.ntp.org
/etc/init.d/ntpd start
```

3.1.4 Setup the networking

- We should add the local network gateway in the */etc/sysconfig/network* and disable the NetworkManager in order to allow openVSwitch to manage the networking.

```
sed -i "\$aGATEWAY=10.30.11.1" /etc/sysconfig/network
systemctl stop NetworkManager && systemctl disable NetworkManager &&
systemctl enable network && systemctl start network && ifdown
enp4s0f0.11 && ifup enp4s0f0.11
systemctl stop NetworkManager && systemctl disable NetworkManager &&
systemctl enable network && systemctl start network && ifdown
enp4s0f0.9 && ifup enp4s0f0.9
systemctl stop NetworkManager && systemctl disable NetworkManager &&
systemctl enable network && systemctl start network && ifdown
enp4s0f1.9 && ifup enp4s0f1.9
```

The enp4s0f0.9 network, which is a tagged VLAN 9 network on eth0 is connected to the internet through USV router. On this network, we have configured a public IP address (80.96.122.48) in order to make the IaaS available to the consortium members.
enp4s0f1.9 is also a network connected to the internet using VLAN 9 tag but on the eth1. This network will be used by Neutron to allow the floating IPs to communicate with the internet through the USV gateway.
enp4s0f0.11 is the internal network with VLAN tag 11 that is used by all compute nodes to access the internet for package installation, updates, etc. 10.30.11.1 is a local router that uses NAT to forward all packages from the compute nodes to the internet.
enp4s0f0.10 is another network that uses tagged VLAN 10 on top of eth0 with IP addresses in the 10.30.10.0/24 range. All the internal VXLAN networks are made on top of this interface, so all the internal traffic between compute nodes is made using this network.

3.1.5 Install with Packstack

In order to install OpenStack, we will use the deployment tool from RDO named Packstack. This tool can be installed and configured by using the procedure described above.

- You should first install packstack using the following command:

```
yum install -y packstack
```

- You should generate the configuration file for the deployment with the following command:

```
packstack –gen-answer-file=kilo_deployment_vxlan.cfg
```

- After this, you should edit the configuration file accordingly with your hardware configuration and IP addresses specific to your environment and also configure the deployment location for every service, if you use multi-node deployment.

```
vi kilo_deployment_vxlan.cfg
```

- Next, you should run packstack to deploy OpenStack RDO to all instances configured:

```
packstack –answer-file=icehouse_deployment_vlan.cfg
```

During this process, you will be required to type the root password for all nodes that you use in your deployment in order for OpenStack to be able to add its public key to each one of them.

Once the process is complete, you can log in to the OpenStack web interface "Horizon" by going to http://$YOURIP/dashboard.
The username is "admin". The password can be found in the file keystonerc_admin in the /root/ directory of the control node.

- After the deployment of OpenStack is completed, you should remove the external networking from the brint interface and add it to the br-ex interface. There is a bug on the packstack deployment tool that doesn't do this thing properly.

```
ovs-vsctl del-port brint enp4s0f1.9
ovs-vsctl add-port br-ex enp4s0f1.9
```

- On the master node, you should update the firewall configuration to accept

```
# Glance open port 9292
-N glance
-I INPUT -s 0/0 -p tcp --dport 9292 -j glance
-I glance -j ACCEPT
# Neutron open port 9696
-N neutron
-I INPUT -s 0/0 -p tcp --dport 9696 -j neutron
-I neutron -j ACCEPT
# Heat open port 8004
-N heat
-I INPUT -s 0/0 -p tcp --dport 8004 -j heat
-I heat -j ACCEPT
# Heat CloudFormation open port 8000
-N heatcloudformation
-I INPUT -s 0/0 -p tcp --dport 8000 -j heatcloudformation
-I heatcloudformation -j ACCEPT
# Heat CloudWatch open port 8003
-N heatcloudwatch
-I INPUT -s 0/0 -p tcp --dport 8003 -j heatcloudwatch
-I heatcloudwatch -j ACCEPT
# Nova VNC open port 6080
-N novavnc
-I INPUT -s 0/0 -p tcp --dport 6080 -j novavnc
-I novavnc -j ACCEPT
# GLANCE
-N GLANCE
-I INPUT -s 0/0 -p tcp --dport 8776 -j GLANCE
-I GLANCE -s 0/0 -j ACCEPT
```

  connections from anywhere to all the OpenStack end-points by adding the following lines at the end of /etc/sysconfig/iptables

3.1.6 Compute node configurations

On the compute nodes, we will be running nova-compute, nova-docker and neutron-agent-service. For running all these services, the compute nodes should meet the minimum specifications: 2 CPUs, 100GB disk and 8GB of RAM.

Next, we describe all the customizations that need to be done in order to have installed and configured compute nodes with both QEMU-KVM and Docker hypervisors.

- On the compute nodes running QEMU-KVM hypervisor, you can configure the availability of the **VNC** on the Console tab for each instance on OpenStack. For this some adjustments of the Nova configuration are necessary. On the */etc/nova/nova.conf* file, you should add the novncproxy_base_url pointing to the external IP address of the IaaS, the *vncserver_listen* should be listening on all ip addresses, and the *vncserver_proxyclient_address* should be the internal IP address of the compute node. Here is an example of Nova configuration that should be adjusted on the compute nodes:

```
novncproxy_base_url=http://80.96.122.48:6080/vnc_auto.html
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=10.30.11.205
```

- We should also install *nrpe* and *nagios-plugins-nrpe*, configure the IP of the monitoring machine, and then open the necessary ports in */etc/sysconfig/iptables*:

```
-N NRPE
-I INPUT -s 0/0 -p tcp --dport 5666 -j NRPE
-I INPUT -s 0/0 -p tcp --dport 4949 -j NRPE
-I NRPE -s 0/0 -j ACCEPT
```

- On our testbed, we decided to have also compute nodes running Docker as a hypervisor. For this end, the following adjustments are required:
  - Install docker on the compute node;
  - Configure Nova to support docker as hypervisor;
  - Configure Glance to support docker images;
  - Configure Host Aggregates;
  - Configure all flavors to either accept docker or KVM hypervisors;
  - Add the needed docker images on all compute nodes running Docker as hypervisor;
  - Create a docker image that runs sshd and supports adding of a ssh key at start and that can run the User-Data at start.

3.1.6.1 Install docker as hypervisor

- On the compute node, we should first install docker 1.7.0, at least.

```
curl -O -sSL https://get.docker.com/rpm/1.7.0/centos-7/RPMS/x86_64/docker-
engine-1.7.0-1.el7.centos.x86_64.rpm
yum localinstall --nogpgcheck docker-engine-1.7.0-1.el7.centos.x86_64.rpm
service docker start
chkconfig docker on
```

- On the compute node, we should add support on Nova for docker, consequently we will install nova-docker.

```
yum install git python-pip -y
git clone https://github.com/stackforge/nova-docker.git
cd nova-docker/
git pull origin stable/kilo
git checkout stable/kilo
python setup.py install
mkdir -p /etc/nova/rootwrap.d
usermod -G docker nova
```

- We should create the file /etc/nova/rootwrap.d/docker.filters with the following content:

```
# nova-rootwrap command filters for setting up network in the docker driver
# This file should be owned by (and only-writeable by) the root user
[Filters] # nova/virt/docker/driver.py: 'ln', '-sf', '/var/run/netns/.*'
ln: CommandFilter, /bin/ln, root
```

- In /etc/nova/nova.conf we should change the compute driver compute_driver to novadocker.virt.docker.DockerDriver.

- On master, we should modify the /etc/glance/glance-api.conf to support docker container images, adding:

```
container_formats=ami,ari,aki,bare,ovf,ova,docker
```

- For configuring the flavors to support docker hypervisor, we should set the hypervisor_type=docker for all flavors that should run on top of docker and hypervisor_type=QEMU for the rest of them. We should also unset the availability_zone metadata from the flavors. Example:

```
nova flavor-key d1.large set hypervisor_type=docker
nova flavor-key d1.large unset availability_zone
nova flavor-key m1.large set hypervisor_type=QEMU
nova flavor-key m1.large unset availability_zone
```

3.1.6.2 Docker image privisioning

- In order to add a docker container image on glance, you should run the following command:

```
docker save nubomedia/kurento-media-server | glance image-create --is-
public=True --container-format=docker --disk-format=raw --name
nubomedia/kurento-media-server
```

- In order to be able to run the newly added docker image on all compute nodes, we should have the docker image available on that node. For this reason, we created a python application that can be run using a jenkins job to automatically pull all the docker images on all compute nodes running docker hypervisor. The application can be found on the following URL:

- https://github.com/usv-public/nubomedia-nova-docker

3.1.6.3 Creating an OpenStack docker image

To create an OpenStack docker image, you should first install OpenSSH on the image, expose the 22 port and make the sshd server start "boot". Next, we have to add the public ssh key defined for the instance on /root/.ssh/authorized_keys and run the User-Data script to make any customization needed on launch.
An example of such a base image for OpenStack is compound of the following Dockerfile and fix_docker_on_openstack.sh plus fix_docker_on_openstack.conf.[4]

[4] https://github.com/usv-public/nubomedia-docker-images/

4 Software infrastructure

This section will describe software solutions used for working on the testbed.

4.1 Gitlab

GitLab Community Edition is open source software for developers to collaborate on code. It allows creation of projects, repositories, access, and code reviews. GitLab CE is distributed under MIT license.

4.1.1 Features

The main features of the GitLab are the following [9]:

- Maintaining security of your code on your own server;
- Managing repositories, users and access permissions;
- Executing code review with merge requests;
- Extended permission system with 5 access levels and branch protection;
- Managing users efficiently by creating projects groups and users teams;
- Integrating the existing system or implementing the ticketing system included in GitLab;
- Allowing line discussions in merge requests and diffs;
- Providing a wiki backed up by a separate git repository for each project;
- Allowing JIRA, Redmine, Slack like external system integrations.

4.1.2 Dashboard

GitLab CE has a web dashboard for developers to collaborate together.

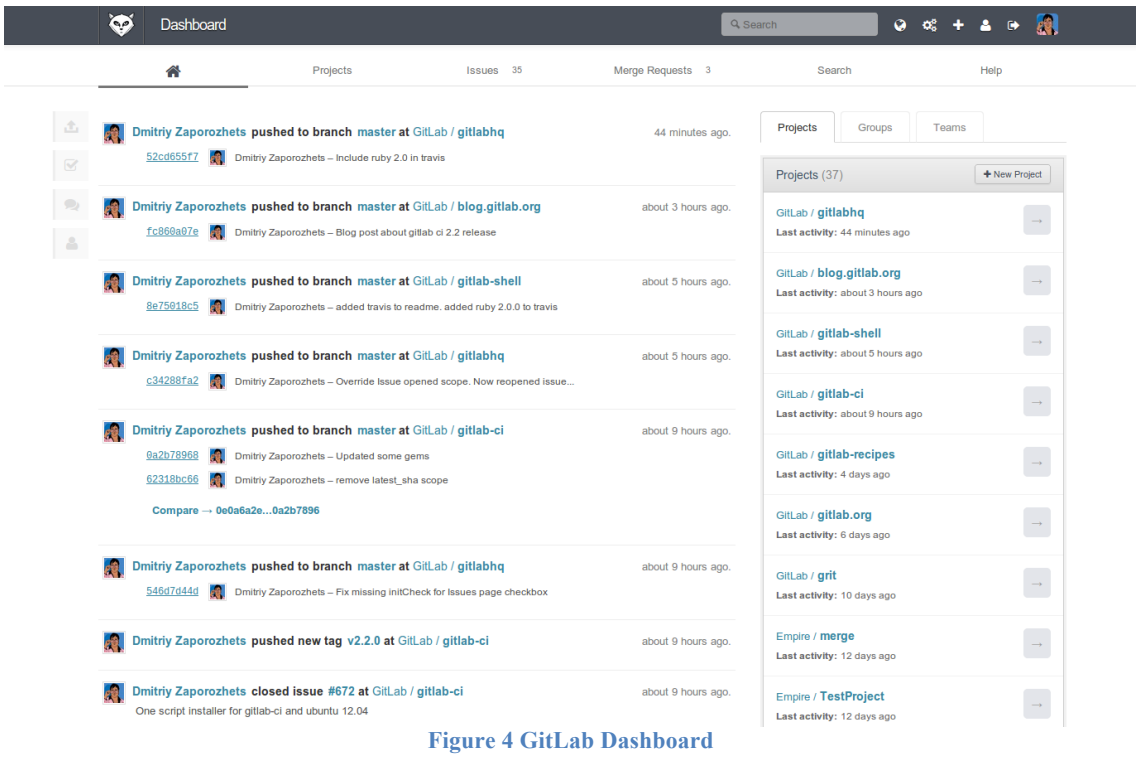

Figure 4 GitLab Dashboard

4.2 Jenkins

Jenkins is a Continuous Integration server that monitors executions of repeated jobs, such as building a software project. In a nutshell, Jenkins provides an easy-to-use system making easy for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

Jenkins was initially a fork from Hudson after disagreements with Oracle who controls Hudson.

Jenkins is distributed under a MIT license.

4.2.1 Features

Jenkins offers the following features [10]:

- **Easy installation**: java -jar jenkins.war can be used as such or it can be deployed in a servlet container, thus removing the need for additional install or database.
- **Easy configuration**: the complete configuration of Jenkins can be achieved using its web GUI providing extensive on-the-fly error checks and inline help. Thus, the need to tweak XML manually is removed, although still available.
- **Change set support**: Jenkins is able to produce a list of build changes from Subversion/CVS. This also entails an efficient reduction of the load on the repository.
- **Permanent links**: Jenkins allows readable page URLs, together with permalinks like "latest build"/"latest successful build", subsequently they can be linked from somewhere else.
- **RSS/E-mail/IM Integration**: RSS or e-mail monitoring of build results to get failures notifications in real-time.
- **After-the-fact tagging**: It allows tagging builds long after their completion.
- **JUnit/TestNG test reporting**: It allows tabulating, summarizing, and displaying JUnit test reports using history information. A graph is created to offer a plotted history trend.
- **Distributed builds**: Jenkins can distribute build/test loads to multiple computers, which permits to exploit the most out of some idle workstations of developers' desks.
- **File fingerprinting**: Jenkins can monitor and record the “pairs” of builds and corresponding jars in terms of production and usage, etc. This applies even for jars that are produced outside Jenkins, which makes it recommended for projects to track dependency.
- **Plugin Support**: It allows the extension of Jenkins using 3rd party plugins. Plugins can be written to create Jenkins support tools/processes needed by team uses.

4.3 Ubuntu repository

We have installed and configured an Ubuntu repository for hosting the NUBOMEDIA artifacts developed by different partners. The repository can be used after running the following commands:

```
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys F04B5A6F
add-apt-repository "deb http://repository.nubomedia.eu/ trusty main"
apt-get update -y
```

The repository can be found at the following address: http://repository.nubomedia.eu/ . On http://repository.nubomedia.eu/apps/files/kurento-tutorial-java/ there are the current versions of the Kurento tutorials for Java developers including the needed resources for starting the any app.

4.4 TURN server

WebRTC enables peer to peer communication, but it still needs servers for clients to exchange metadata to coordinate communication. This is called signaling and tcope with network address translators (NATs) and firewalls.
Kurento Media Server uses both STUN and TURN servers. STUN servers can hold a very large amount of users because they are just coordinating the communication. Clients have direct connectivity, but TURN servers are actually a relay, meaning that all traffic passes through it. For these reasons STUN servers are free and very easy to find, i.e.: https://gist.github.com/zziuni/3741933 , but TURN servers are paid services, and for this reason we installed a private TURN server for NUBOMEDIA. It can be configured on the Kurento Media Server configuration file updating turnURL parameter:

```
turnURL=nubomedia:nub0m3d1a@80.96.122.61:3478
```

4.5 NUBOMEDIA Autonomous Installer

The NUBOMEDIA Autonomous Installer is a deployment tool created in task T3.5.2. Its main purpose is to deploy the NUBOMEDIA platform on top of an IaaS based on OpenStack that has the support of both KVM and nova-docker hypervisors on top of it, and a PaaS platform based on OpenShift Origin v3.1.

Before starting the Autonomous Installer, you must have the following issues prepared:

- OpenStack requirements:
  - Keystone endpoint with admin username and password. The endpoint should usually be something like *http://x.x.x.x:5000/v2.0*;
  - The tenant name on which you are planning to deploy the NUBOMEDIA platform;
  - Glance endpoint for storing the NUBOMEDIA platform images. Its endpoint should be similar to *http://x.x.x.x:9292*;
  - Floating IP pool name, to enable the allocation of public IPs to the NUBOMEDIA instances;
  - You should upload a public key on the OpenStack environment and put the associated private key on the Autonomous Installer root directory;

- If you are planning to run Kurento Media Servers on top of Docker inside the IaaS, next, you should follow the steps described in section 3.1.6.

- OpenShift requirements:
  - The deployed OpenShift should run the Origin version 3.1 and should have its default endpoints accessible from outside;
  - The OpenShift keystore should also be placed on the Autonomous Installer root directory, and should update the variable name for it. In order to create the keystore, you can use Portacle[5];
  - The HAProxy router inside the OpenShift deployment should support L2 load balancing mechanism.

After you have gathered all the variables required by the Autonomous Installer, you should start it by using the following command: *python main.py* .
At this point, the installer will start pulling all the images from the http://repository.nubomedia.eu/images/ and then upload them on Glance. When we have all the images available on the platform, we can start the instances and get all their IP addresses in order to know how to interconnect them in the NUBOMEDIA platform. Each step is monitored in order to determine the time needed for deployment of the NUBOMEDIA platform, and the time needed for each step to be completed.

To provide details on the validation of Objective 1, Metric 1.2 General validity of the admin tools, we run the Autonomous Installer on a new tenant of the testbed. Using the log file that the installer creates, we then extracted the timing for each step that was done using the Autonomous Installer.

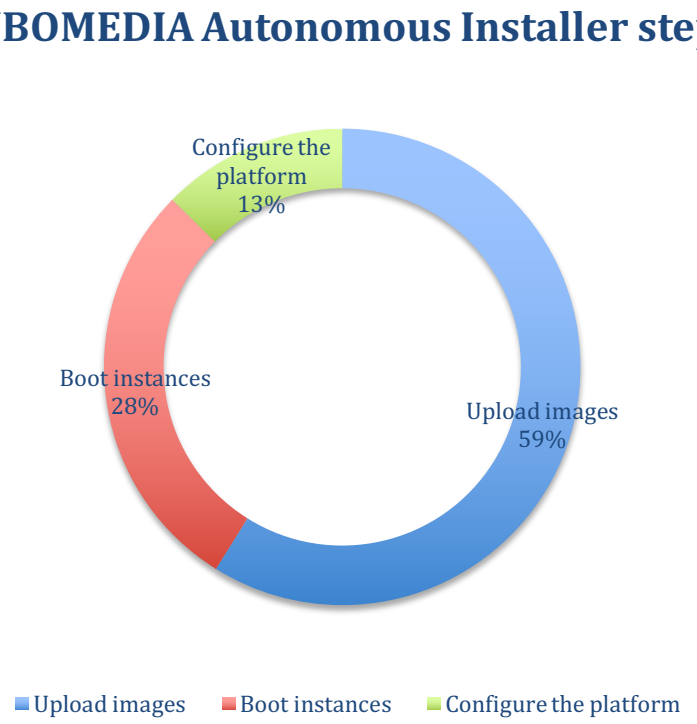

Figure 5 NUBOMEDIA Autonomous Installer steps percentage

[5] http://portecle.sourceforge.net/

Figure 5 shows that most of the installation time is spent on uploading the images from the NUBOMEDIA repository to the Glance registry of the IaaS.

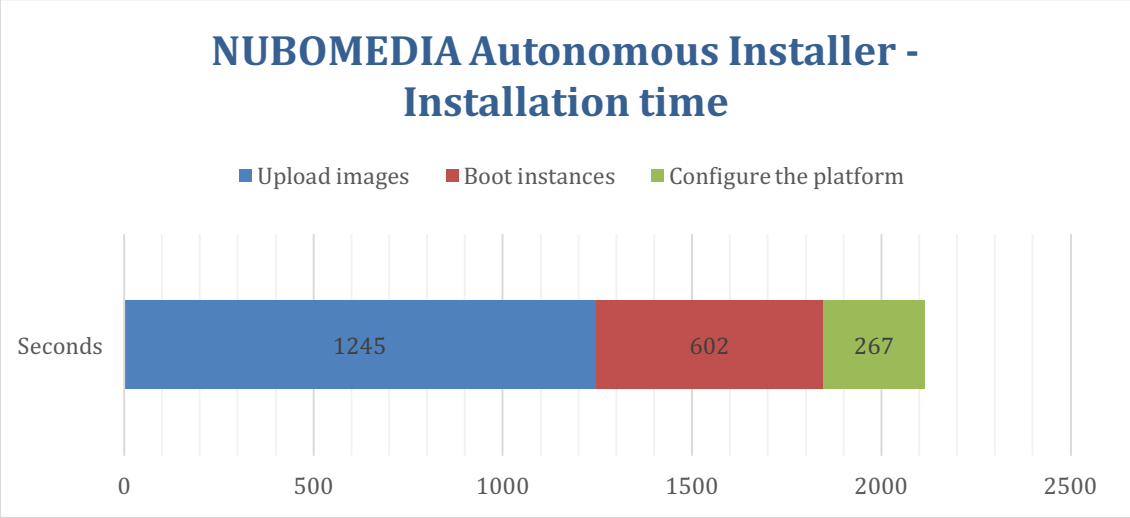

Figure 6 Autonomous Installer - Installation time

As it can be seen in Figure 6, the total time needed to deploy NUBOMEDIA platform using the NUBOMEDIA Autonomous Installer is 2114 seconds, meaning **35 minutes and 14 seconds**. In addition to this, we should also add the time needed for the system expert administrator to prepare the requirements and to adjust the configuration file of the installer. We should also add the time needed for investigating the logs produced by the installer and the time needed to deploy a basic application using the PaaS manager in order to check that everything is working at the expected parameters. All these steps together should allow an expert administrator to deploy NUBOMEDIA in less than 8 hours. Here [AUTONOMOUS INSTALLER LOG] you can find the full log of the deployment that was used to gather the metrics presented above.

We consider that the Autonomous Installer admin tools presented in this D6.1.2 deliverable successfully met the Metric 1.2 General validity of the admin tools of the Validation of Objective 1: An elastic cloud infrastructure for interactive multimedia in DoW NUBOMEDIA project.

5 Continuous Integration Plan

For NUBOMEDIA to be successful, we should use a CI system that should keep the system fully integrated at all times. Integration can happen many times a day, when a partner pushes his code to the Git repository, a Jenkins job has to be triggered to run on the Jenkins server, and check if the code can be compiled and all unit tests run successful. This way each partner will be able to know if there are any issues with his code or if the code is ready to be integrated in NUBOMEDIA.

Each night, there is a cron that triggers a Jenkins job to build the NUBOMEDIA platform and deploy it to the testbed located on the Dev tenant on: http://devconsole.nubomedia.eu . Whenever we want to deploy the current development environment or only some parts of it to production, we just have to run the appropriate Jenkins jobs from http://jenkins.nubomedia.eu/view/Production/ tab from the Continuous Integration system.

In order for USV to be able to build and deploy NUBOMEDIA, each partner should provide details regarding the compilation, installation and configuration for the parts of NUBOMEDIA that they are developing. As a general rule, it is best to provide the integration instructions on a README.md file on the Git repository that hosts the source code, or they can provide instructions using a document similar with the following:
https://docs.google.com/document/d/1eNUhABOyXOCkhi2eZTvwieDgH6Wsob1pfsufajhyikk/edit#heading=h.ij0weqwop9ew

As a git repository for NUBOMEDIA, we suggest to use Github because it is the most well-known free web-based Git repository hosting service for open source projects, which offers all of the distributed revision control and source code management (SCM) functionality of Git. Currently Github has more than 9 million users and over 21.1 repositories, making it the largest code hoster in the world. For joining the https://github.com/nubomedia organization please send an email to nubomediaproject@gmai.com .

5.1 Jenkins

In NUBOMEDIA, we use Jenkins[6] for continuous integration. Jenkins is an open source continuous integration tool written in Java.

On Jenkins, we use a plugin named Docker plugin, that aims to provide Jenkins capability to use a Docker host to dynamically provision a slave, run a single build, then tear-down that slave. We configured a Jenkins slave node that hosts all Docker containers, and we created separate jobs to do nightly build images with Docker for each running environment needed in the CI system. When these jobs are done, fresh images are uploaded to Jenkins Docker machine, and new slave nodes with labels are added to the Jenkins master. The advantage of using this architecture is that Jenkins can run jobs on fresh and isolated Docker containers without installing any packages or changing configurations on a live Jenkins node.

[6] https://jenkins-ci.org

5.2 Packer

Another tool that we use for CI is Packer[7], which creates the virtual machine images from a specific configuration.

We have setup a Jenkins job to run every night and clone the latest configurations for the OpenStack images for the NUBOMEDIA platform, then build fresh images with latest packages for each of them, and upload the newly created images to Glance, and after that delete the old images.

Source code of packer scripts are on NUBOMEDIA git repository:

- http://git.nubomedia.eu/usv/adm-nubomedia-openstack-repo
- http://git.nubomedia.eu/usv/adm-nubomedia-docker

Jenkins jobs configurations are found on:

- http://git.nubomedia.eu/usv/ci/tree/master/jenkins_jobs/images

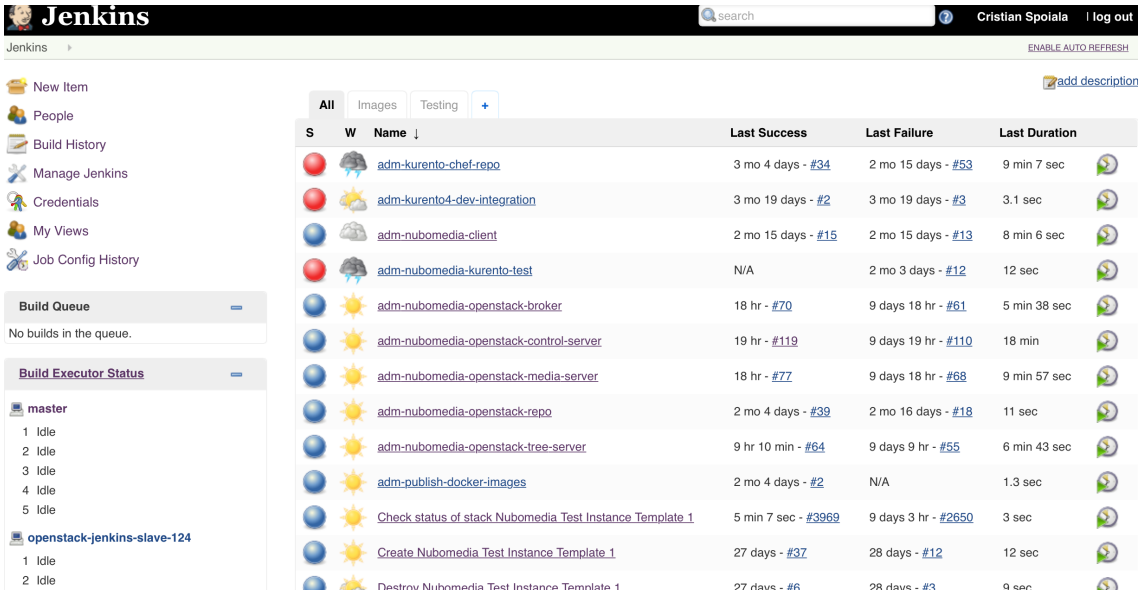

Figure 7 NUBOMEDIA Jenkins Dashboard

5.3 Access CI tools

The NUBOMEDIA implementation of Jenkins can be found at

- http://jenkins.nubomedia.eu

All the installations scripts for CI and all configuration files used on the testbed (OpenStack deployment) are stored on NUBOMEDIA shared git repository on Gitlab:

- http://git.nubomedia.eu

When partners need access to NUBOMEDIA Jenkins or Git repository, they should ask access from alin.calinciuc@usv.ro .

[7] https://www.packer.io

6 NUBOMEDIA tests

6.1 Testing plan

Using Jenkins as the main Continuous Integration tool allowed the design of a series of tests for the software artifacts developed for the NUBOMEDIA platform. The objective of the testing was to make sure that quality is kept at high levels during the development of NUBOMEDIA software. Beside software development process, the tests are also covering the infrastructure and functionality of NUBOMEDIA PaaS.

In order to integrate NUBOMEDIA software artifacts on testbed, we provide nightly build images for the NUBOMEDIA components and Jenkins jobs to test that the newly created images are valid in a NUBOMEDIA architecture.

Images are created automatically every day by a Jenkins job using Packer tool mentioned in Section 5. Following the release of the software artifacts on templates, functionality tests were developed to ensure that the release did not break expected functionality.

6.2 NUBOMEDIA integration tests

NUBOMEDIA integration tests are an important aspect of ensuring functionality of the platform when hardware or software changes are performed. They should alert NUBOMEDIA operators that applications had issues such as performance issues and/or network accessibility issues.

For most of the tests, Jenkins platform (mentioned in Section 5) and open source tools were used.

6.2.1 Health tests

To ensure NUBOMEDIA stability, we developed health tests using Jenkins. They are testing the entire workflow of developers on NUBOMEDIA. Additionally, it checks if the application starts successfully and if it performs a series of tests on the KMSs that have been started by the PaaS to ensure that NUBOMEDIA is running in the expected parameters.

The tests are separated in 3 stages:

- Stage 1: Deploy the app on NUBOMEDIA by using PaaS manager API, the same API that is used by developers to deploy applications on NUBOMEDIA. The Jenkins job is creating an application based on Magic Mirror, and it is passing the application name to the second Jenkins job as parameter.
- Stage 2: Second Jenkins job is checking if application in Stage 1 was successfully created by calling the PaaS manager API. If the application was not created for 1h, it is assumed that the application creation failed.
- Stage 3: If application is created successfully and all its services are running on PaaS, we use the API to get the KMSs started by the PaaS, and we perform a loopback test using java-client test suite.

Any failure at any stage will be reported via email. In the following Figure, we present a flowchart with the workflow of the test.

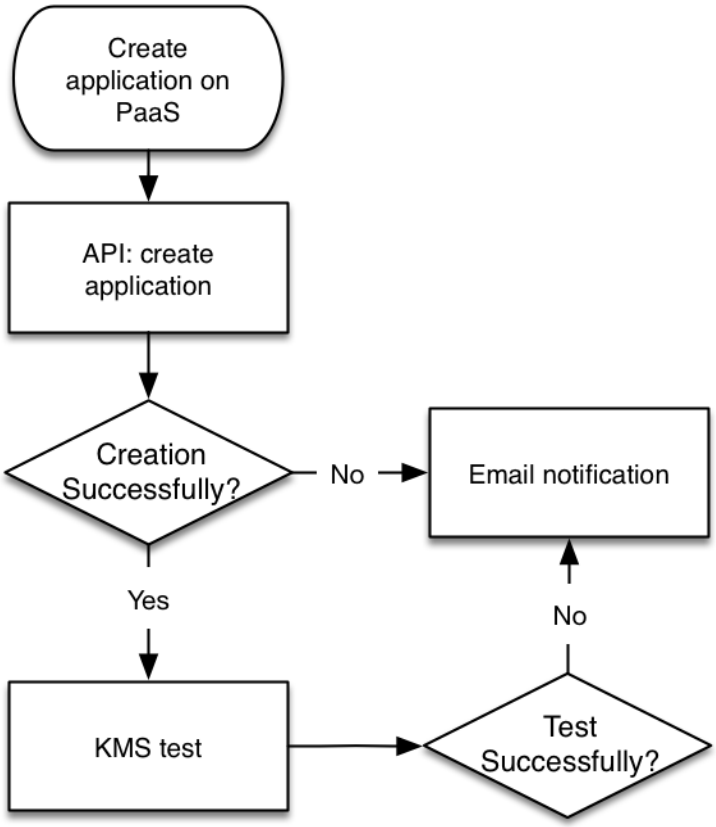

Figure 8 Flowchart for Health test

6.2.1.1 Magic mirror application

This web application shows a WebRtcEndpoint connected to itself (loopback) with a face detector CV filter that detects the face and adds a hat on top of the face.

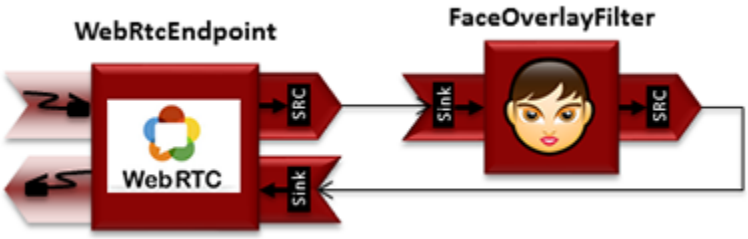

Figure 9 Magic mirror WebRTC application

The application follows client-server architecture. Client-side which is implemented with Javascript starts a WebSocket for custom signals protocol. Another WebSocket based on Kurento Protocol is used for communication between server-side Kurento Java Client and KMS started by the PaaS.

6.2.1.2 KMS tests

KMSs started by the PaaS were tested with Kurento Java Client. We developed a Jenkins script that is starting a Chrome Browser and performs a series for tests for the KMS functionality.

In order to simulate a browser, the job uses the following:

- Selenium Chrome Driver [2]
- Chrome Browser
- Xvfb Display Server [3]

Job is downloading and installs a binary of Chrome browser for Linux from:

- https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb

Other dependencies of the job are:

- Maven for build management;
- Git for source control;
- Java Development Kit (JDK).

NAEVATEC partner developed the Java integration tests and USV fetch the software artifacts from:

- https://github.com/Kurento/kurento-java.git

The following tests are performed:

- WebRtcLoopbackTest - This is a functional test for NUBOMEDIA instance. Media pipeline from instance is composed from a single media element (WebRtcEndpoint) and when it is received on the server, it is sent back to the client.
  The test will pass if the video stream is received back, and play time and colour in the video meet the expectations.

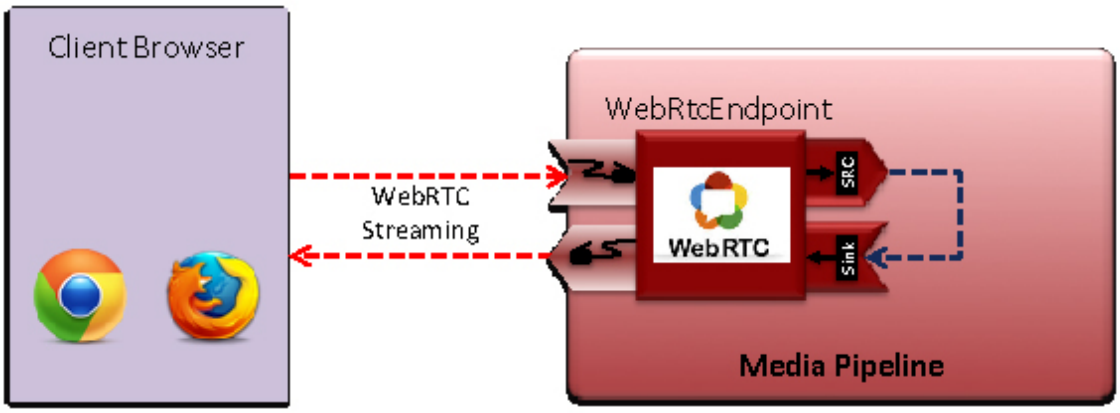

Figure 10 Architecture of the WebRTC Loopback Test

- WebRtcQualityLoopbackTest – Similar with WebRTC Loopback Test, this test is additionally checking the audio quality with PESQ.

Jenkins Job configurations can be found on NUBOMEDIA:

- http://jenkins.nubomedia.eu/view/Testing/job/NUBOMEDIA-create-app/
- http://jenkins.nubomedia.eu/view/Testing/job/testing-NUBOMEDIA-Loopback/

6.2.1.3 KMS image integrity

When on KMS are added, new capabilities automatically create a new KMS image and before being added to the repository, it is tested with a Jenkins job that is passing all functional tests. The tests are similar to those for 6.2.1.2, based on Kurento Java Client and using WebRTC API protocol.

6.2.2 Benchmark tests

We performed benchmark tests that are measuring different types of metrics for validation. We performed simulated load with KPT tool developed by URJC partner.

Using the document provided by URJC that developed a tool to benchmark KMSs, USV created Jenkins jobs to test performance (CPU and latency) metrics on KVM images and Docker images.

We used KMS version 6.1.1.
KVM images are deployed on virtual machines using QEMU for emulating I/O devices.
Docker images are deployed on top of the Docker hypervisor running Docker v1.7.1.

6.2.2.1 Architecture of the test

To perform the test, multiple machines were deployed (Figure 8):

- 5 KPTs - that are running Chrome browsers with Selenium WebDriver;
- Fake client machine - a KMS used to simulate users by using WebRtcEndpoints;
- 5 KMSs - based on a Docker image and a KVM based image.

Configuration of each KMS is:

- 4GB RAM;
- 4vCPU.

Configuration of each KPT is:

- 3GB RAM;
- 8vCPU.

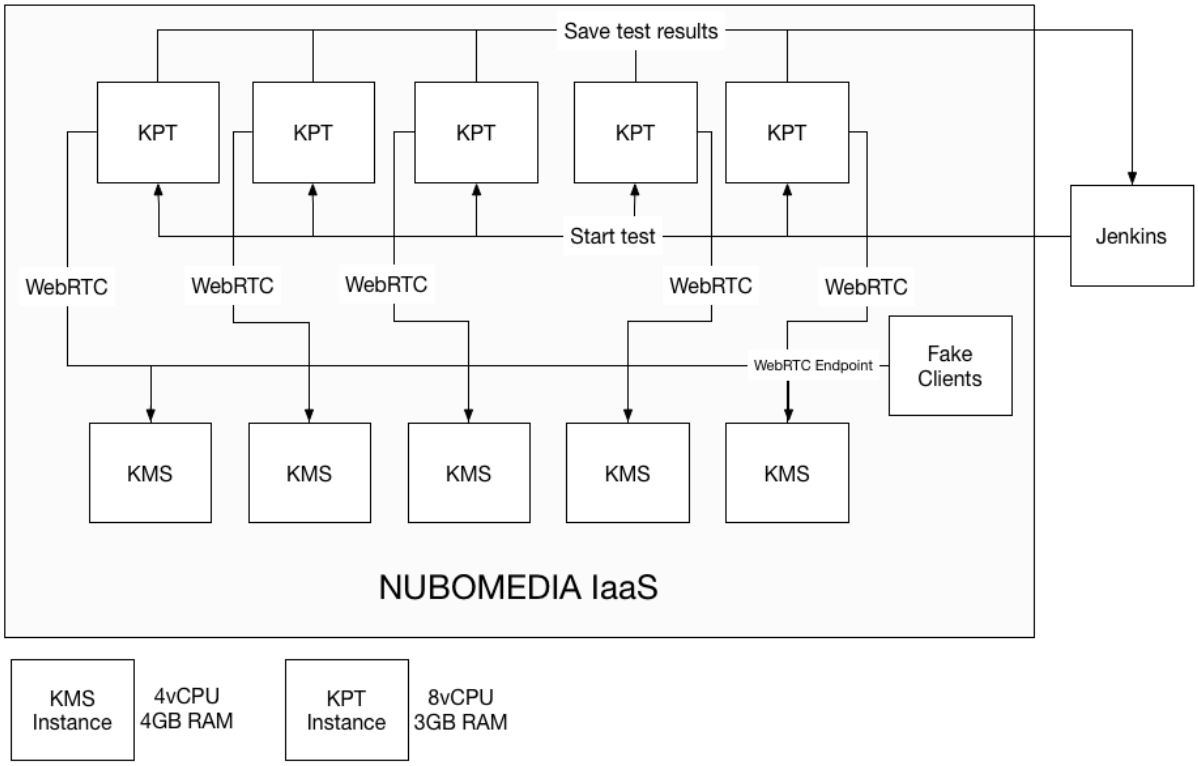

Figure 11 Architecture of the benchmark test

6.2.2.2 Configuration of the test

The test ran in parallel with 5 virtual machines or Docker containers. For tests without any media filter, we used 50 clients. For encoder media filter, we used 15 clients. Memory usage was monitored and the machines did not exceed 80% memory usage.

The test ran for 200 seconds, and the delay between clients was specified to 1000ms.

6.2.2.3 Test results

CPU usage and latency of the KMS are displayed in Figure 9 for test without any media processing.

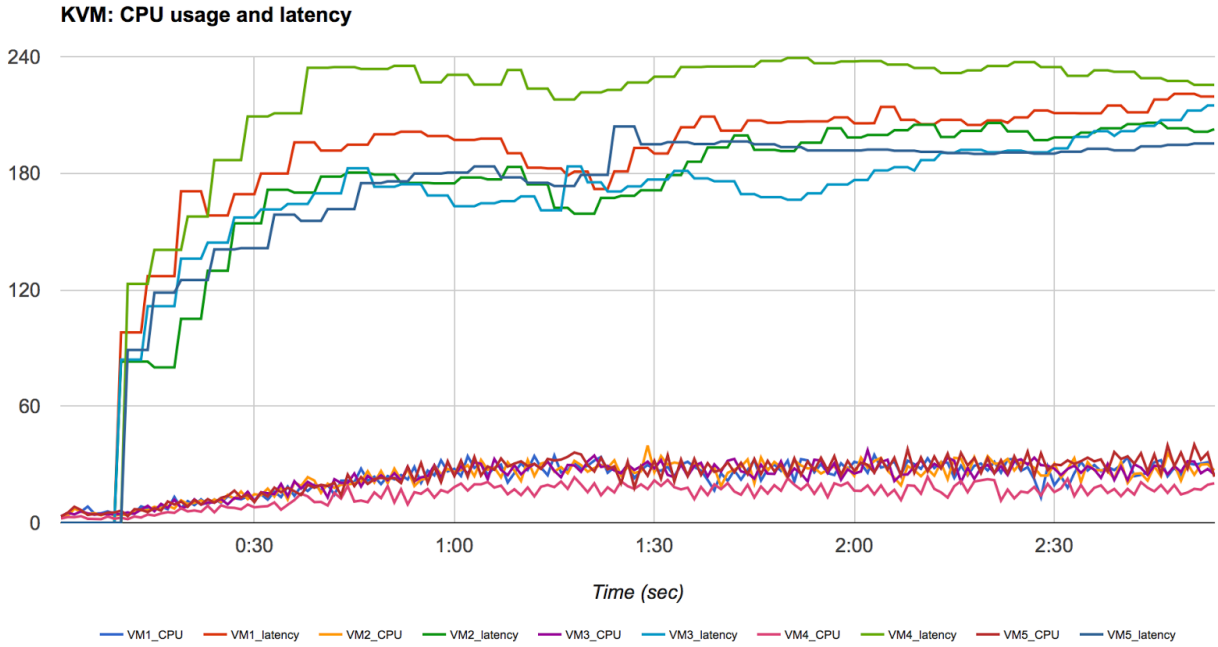

Figure 12 KVM test without filters with 50 clients running on 5 KMS servers

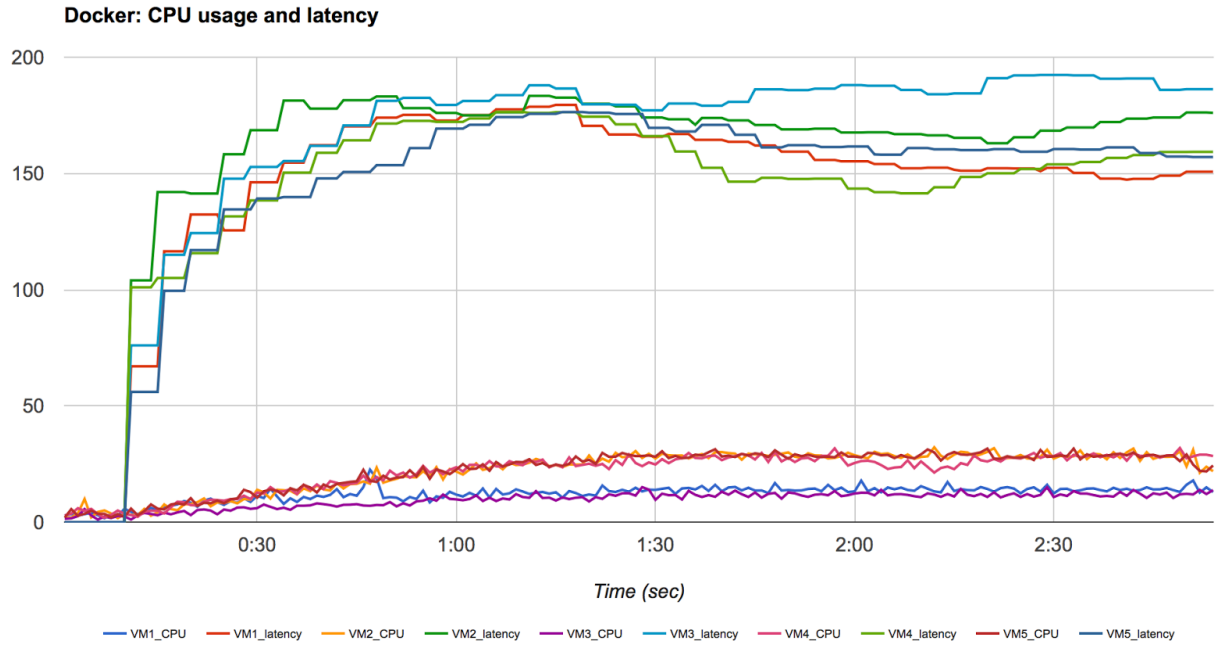

Figure 13 Docker test without filters with 50 clients running on 5 KMS servers

These two charts (Figure 9 and Figure 10) showcase clearly that the CPU usage is 5-10% lower for the Docker instances and that the latency is more stable, and a bit lower.

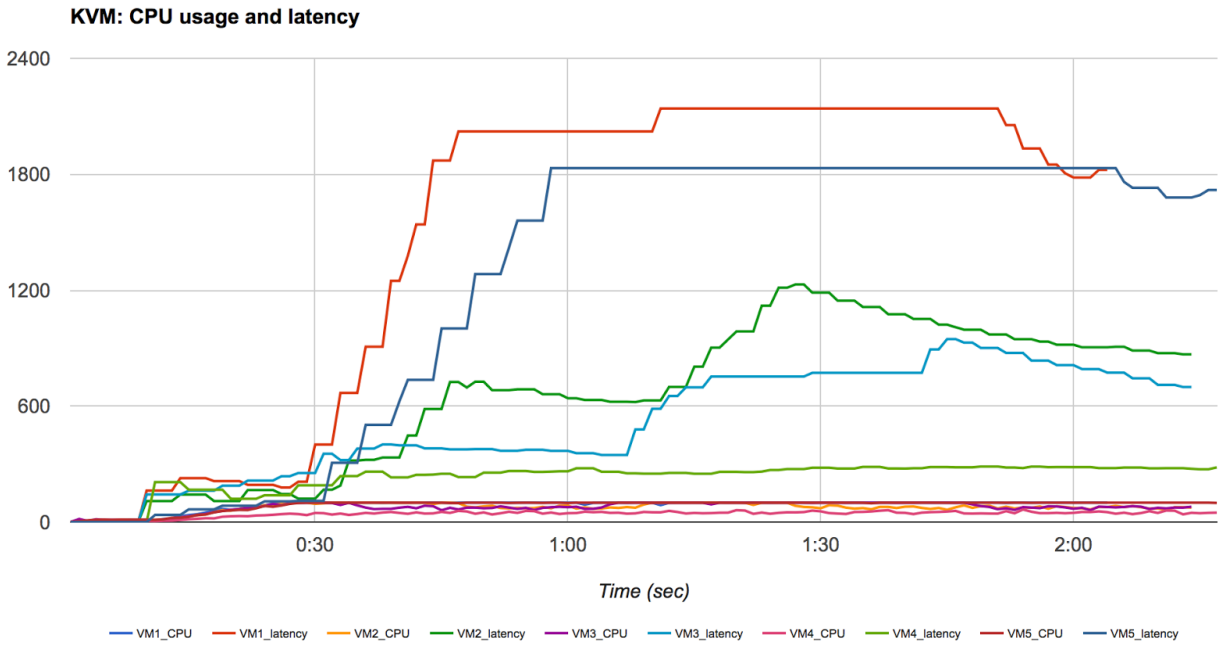

Figure 14 KVM test with encoder media filter on 15 clients running on 5 KMS servers

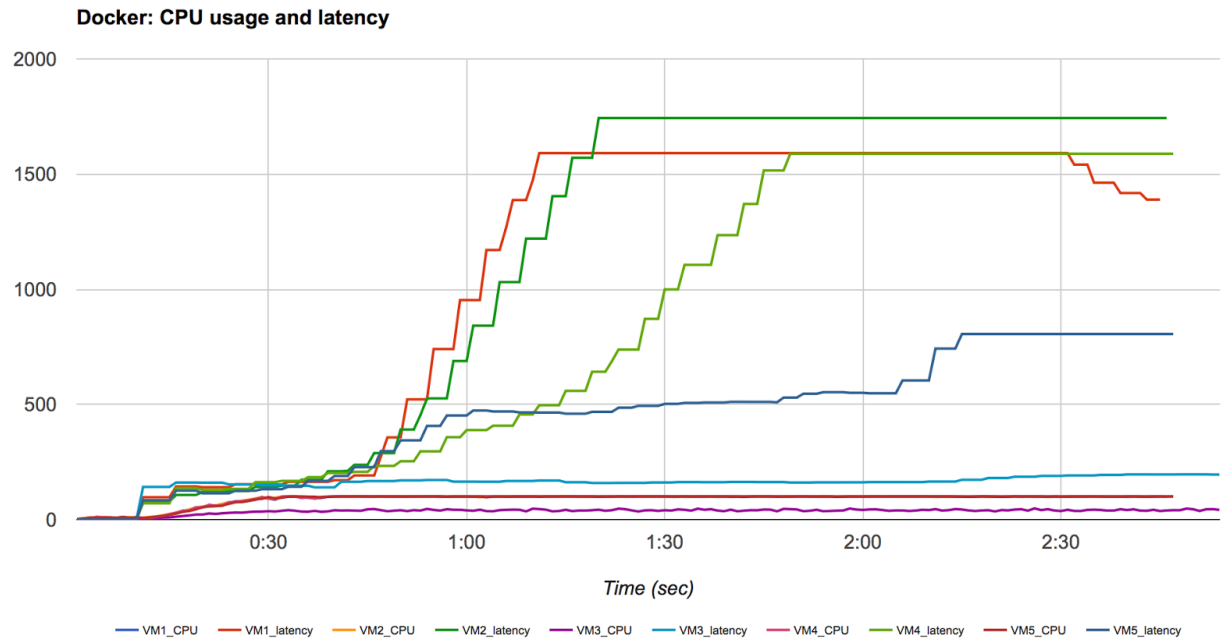

Figure 15 Docker test with encoder media filter on 15 clients running on 5 KMS servers

6.2.2.4 Conclusion

As a result of the benchmark tests, we concluded that Docker performance is better than KVM, especially for latency which is a critical metric for NUBOMEDIA applications. From the performed tests we decided to use Docker as a hypervisor for the deployment of NUBOMEDIA media servers inside the IaaS platform.

7 Access to the testbed

This section provides details on how NUBOMEDIA testbed can be accessed for development. It includes guidelines for developers to deploy applications on NUBOMEDIA PaaS, and also tools for NUBOMEDIA operators to monitor and manage the installation.

7.1 Developers guidelines

This section focuses on developer guidelines for creating, deploying and managing applications on NUBOMEDIA testbed.

7.1.1 Prepare Kurento Media Server

All developers who want to start writing WebRTC applications should first learn the principles of programming with Kurento for Java developers. For doing so, we recommend reading one or more tutorials from this online resource[8].
There are also some prerequisites that you need to prepare before starting:

- Install Kurento Media Server locally using the tutorial to be found here: http://www.kurento.org/docs/current/installation_guide.html
- Check what STUN server is configured on /etc/kurento/kurento.conf.json and if it works well using the following Javascript function:

```
function checkTURNServer(turnConfig, timeout){
return new Promise(function(resolve, reject){

    setTimeout(function(){
        if(promiseResolved) return;
        resolve(false);
        promiseResolved = true;
    }, timeout || 5000);

    var promiseResolved = false
      , myPeerConnection = window.RTCPeerConnection || window.mozRTCPeerConnection
|| window.webkitRTCPeerConnection   //compatibility for firefox and chrome
      , pc = new myPeerConnection({iceServers:[turnConfig]})
      , noop = function(){};
    pc.createDataChannel("");    //create a bogus data channel
    pc.createOffer(function(sdp){
      if(sdp.sdp.indexOf('typ relay') > -1){ // sometimes sdp contains the ice
candidates...
        promiseResolved = true;
        resolve(true);
      }
      pc.setLocalDescription(sdp, noop, noop);
    }, noop);    // create offer and set local description
    pc.onicecandidate = function(ice){  //listen for candidate events
      if(promiseResolved || !ice || !ice.candidate || !ice.candidate.candidate ||
!(ice.candidate.candidate.indexOf('typ relay')>-1))  return;
      promiseResolved = true;
      resolve(true);
    };
  });
}
```

[8] http://www.kurento.org/docs/current/tutorials.html

```
// example usage:

checkTURNServer({
   url: 'turn:127.0.0.1',
   username: 'test',
   credential: 'test'
}).then(function(bool){
   console.log('is TURN server active? ', bool? 'yes':'no');
}).catch(console.error.bind(console));
```

- Install java-jdk and maven using the following command:

```
# apt-get install openjdk-7-jdk maven -y
```

- Add the following profile to your local maven settings file ~/.m2/settings.xml :

```
<profile>
<id>kurento-develop</id>
<activation>
<activeByDefault>true</activeByDefault>
</activation>
<repositories>
        <repository>
            <id>kurento-snapshots</id>
            <name>Kurento Snapshot Repository</name>

<url>http://maven.kurento.org/archiva/repository/snapshots/</url>
            <releases>
                <enabled>false</enabled>
            </releases>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>kurento-releases</id>
            <name>Kurento Repository</name>

<url>http://maven.kurento.org/archiva/repository/internal/</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
</profile>
```

When all the prerequisites are ready on your local machine, you can start developing the first WebRTC application for the NUBOMEDIA platform. We suggest that you start with the Magic Mirror application because this tutorial is based on the Magic Mirror application that you can find here[9].

[9] https://www.kurento.org/docs/5.1.1/tutorials/java/tutorial-2-magicmirror.html

7.1.2 Create your first application

You can start your first application using the tutorials presented in previous section, but if you want to use an already developed Magic Mirror application, you can get it from the Kurento/kurento-tutorial-java git repository:

```
# git clone https://github.com/Kurento/kurento-tutorial-java.git
```

Then, you should go to the kurento-magic-mirror directory inside that repository and add the Kurento Client extended library required for the application to work on the NUBOMEDIA platform.

7.1.2.1 Add Kurento Client extended library

If your application is going to use the Kurento Media Server (KMS), then you need the extended version of the Kurento Media Client (KMC+) as dependency to your project. This version of the KMC consumes the exposed API of the Virtual Network Function Manager (VNFM) responsible for managing the lifecycle of the media components (KMS). KMC+ can be downloaded from the repository, compiled and then included into your project's library as a dependency.

If you are using Maven, you should take the following steps:

```
# git clone https://gitlab.fokus.fraunhofer.de/NUBOMEDIA/kurento-client-extended.git
# cd kurento-client-extended
```

At this moment the kurento-client-extended should be available on your local repository. Next, you should add the dependency on you project *pom.xml* file:

```
<dependencies>
...
<dependency>
<groupId>org.kurento</groupId>
      <artifactId>kurento-client</artifactId>
</dependency>
<dependency>
             <groupId>de.fhg.fokus.nubomedia</groupId>
             <artifactId>kurento-client-extended</artifactId>
             <version>1.0-SNAPSHOT</version>
      </dependency>
</dependencies>
```

7.1.2.2 How to instantiate the KMC+

In order to make the application work on the PaaS, you should change the way we instantiate the KMC. Normally, you would instantiate the KMC with the method KurentoClient.create(Properties properties). For example:

```
KurentoClient.create(System.getProperty("kms.ws.uri", DEFAULT_KMS_WS_URI));
```

However, with the extended version, you will need to instantiate the KMC using this method KurentoClient.create(). This way, the URL of the KMS will be discovered using the following procedure:

- If there is a system property with the value "kms.url", its value will be returned.
- If the file "~/.kurento/config.properties" does not exist, the default value "ws://127.0.0.1:8888/kurento" will be returned.
- If the file "~/.kurento/config.properties" exists:
  - If the property "kms.url" exists in the file, its value will be returned. For example, if the file has the following content:

    ```
    kms.url: ws://4.4.4.4:9999/kurento
    ```

  - The value "ws://4.4.4.4:9999/kurento" will be returned. If the property "kms.url.provider" exists in the file, it should contain the name of a class that will be used to obtain the kms url. For example, if the file has the following content:

    ```
    kms.url.provider: com.company.PaasKmsUrlProvider
    ```

The above, explains exactly how this procedure works on the KMC. The extended version KMC+ is the implementation of the "org.kurento.client.internal.KmsProvider" interface. This KMSProvider connects to the VNFM to obtain the KMS URL dynamically.

Next, you need, on your test environment, a configuration file located on your home directory within a folder *.kurento* called *config.properties*. The content of the file should be as follows:

```
kms.url.provider=de.fhg.fokus.nubomedia.kmc.KmsUrlProvider
```

When you finish, you have to compile your application using the following command:

```
# mvn clean install
```

Finally, you should find a zip file inside target directory having the name similar to: kurento-magic-mirror-6.2.2-SNAPSHOT.zip. That is your first NUBOMEDIA application compiled and ready to be added to a Dockerfile required for the PaaS.

7.1.3 Create a Dockerfile

Next step is to create a Dockerfile that will be used by PaaS to install and run your application on it. The zip file has to be located in the same repository as the Dockerfile.

Here is an example of a Dockerfile for our first app:

```
FROM nubomedia/apps-baseimage:v1

MAINTAINER Nubomedia

RUN mkdir /tmp/magic-mirror
ADD kurento-magic-mirror-6.2.2-SNAPSHOT.zip /tmp/magic-mirror/
RUN cd /tmp/magic-mirror/ && unzip kurento-magic-mirror-6.2.2-
SNAPSHOT.zip

EXPOSE 8080

ENTRYPOINT java -Dapp.server.url=https://[URL] -jar /tmp/magic-
mirror/lib/kurento-magic-mirror.jar
```

- **nubomedia/apps-baseimage:v1** is the Nubomedia base image to build and run the application. It is based on alpine linux with java 8 and bash installed and it also has internal packet manager to install other packages. We choose that because ubuntu-based images had build problems and poor performance;
- **Maintainer** is the developer of the application (and the author of the Dockerfile);
- **RUN** is Docker instruction that executes a command given as parameter, thus any misspelling or wrong parameters will result in a build failure;
- **ADD** is a simple cut and paste from base directory to given directory;
- **EXPOSE** is used to declare the ports that will be used from the docker container;
- **ENTRYPOINT** is the command that docker will execute when it starts the container;
- **[URL]** is an external link to resources needed by the application. For Magic Mirror, there is a file named mario-wings.png that should be available in a img directory on your server;
- For other commands and references here[10] are the docker guidelines.

7.1.4 Host the application on a public Git repository

At this stage you should have the Dockerfile and the zip file in the same directory, ready to be published to a public Git repository:

```
# ls -la
Dockerfile
kurento-magic-mirror-6.2.2-SNAPSHOT.zip
# git init
# git add .
# git commit -m “initial commit”
# git remote add origin git@github.com:YourGithubAccount/nubomedia_app.git
# git push origin master
```

7.1.4.1 Host the application on a private Git repository

For private applications that you do not want to be publicly available, you should follow the next instructions:

[10] https://docs.docker.com/v1.8/reference/builder/

- Create a secret using an HTTP POST on http://80.96.122.73:8081/api/v1/nubomedia/paas/secret with this JSON object as body (**IMPORTANT**: private key value must preserve newlines, use **n** to preserve it):


```
{
      "projectName":"nubomedia",
      "privateKey":"-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQEAxBSEyhfttjfqesNc8/NNgYp93hDSetbWWi3/v48FhZebU6P/\nwx2C
5XtrOo5Co8R98loqCh+Dw3OoH6HhPBqSq7JQYqEU3MakaZ4Sh/19rziPPA1g\nmKU9GEXhog
u415vQz1sD/heXDQFNZdUokcYFNrbfGpfmsBEdkcVzPLBN4GJP1wCu\nM7oWzwOhLbqh98ib
8WadqDBiRqztal5h2Xa7wWsgulDkaCvtOPlIm7W/TICyUXOd\nNH811dJ4EhqmbJQZlOzsuA
2js6Vz58HqV8ytYiVdkQnsP5jLzPNM0nkSh3XWeube\n0zhuEJmh0wS9BmsipKa9VIXNH2/m
nufq28wMoQIDAQABAoIBABUn3ZfscwJpEAyE\nzZ+ojaE/bwspp3wHeAMs2V4ysTbTv7eLh0
nnAjt+UHh15uzCg5BFeCm1csMA1I/t\nKF8SwuZxi8jIdnbHm++lVXyEti3UnWeuTdDKa0gW
Kh0QxLXGowXsXQbqRqrpjA9D\nq2fnBKL9oh69au9uOVGEC0XuA8kEwg7aJ39Dcr543UJj/Q
feNoaJtw1LA99dag1/\nG4is9dpBRPU4uIQo37CBLf1TbpYTQNL+46WYHhz72p/BIAoIv+kS
dgsrRvyV/JeP\ncTcazMUJrXSxDZzWmfr32yqtQRu1nU+uckisDwpqgUuhfnoKkoU4BH1uKx
W+tzK6\nXktqJlECgYEA601pR2HGspgcZcNv7zeWRGxWbMngfzC/W33sCsTlqvLcP+gq6bJX
\n4PB7ulLCDII2MEnF05+Fh8pM+egUgr7Y+B3AFrOJDg2tIlXu8Ro3Uwoxd28oO61Z\nawdv
kuCZ9Qeue9mPYVGIjQoY84UbJlbXjfiF+Huylx24VeThOjCYZm0CgYEA1VPk\nk42r7I/61A
r+SscHyaT8lUMssMjENa2y7Wh4WvH/ZKirgo2E1q/uPdtrP3YfP154\ngf0bbsrl6zm1oPB/
y0UOGC0XWc7F6nn23C9ax27ICavo62g06+t/qqW4BcAfp8VN\nIxmyNNtvo56CuoI1+PcyMZ
aVbYwPgecwQHGMboUCgYEApR8PsB33N8DyvJ7nX/Gc\nK6vzAiiwt9DXmDbHe88sdEg1M0uT
Qaf7b0iTKu+EaQ6/RCehAZ7CL8ZROlYYfp+6\n1nLaJ5QZq5kBVEUFhoAlLsrKZ8vDag194F
O5glLG92JKmXLU4TA8KO1bERjpMoBi\nh6hNK1ByxQUAJJaXTyRm7gkCgYBrTbCK+9b/vgh4
AjOY33YuWnvmhIyFO+dd7Mo0\nmrj3XgSN2D21BIRODN50ZNsUZ9Ed6eIJ2Iuk9hAieru+gV
p2n3yQcpXtSZHJ+KFQ\nbc1mxXV/T+ZwCtGb3bAw4Pyof9QsapT7U+CMr9f+4Ct3rymA2q53
vPvax3nBaM2f\njL4LlQKBgE/VkGclp53bBagi6A+AiM6U2oYqgFSyyNdLYI6N+5zghbERd2
zvF9Vu\nfjCSbyGgpvyJw2cPTF+MwiYvfuRBGW6JNfk2QhdoFHWIZ0Zis2cGs3o/wHCW5K6/
\nXSYNlaIoCsgWHXHPUaVgWui2B51nfnu31B5tUYAmjXZc42Miz+ml\n-----END RSA
PRIVATE KEY-----"
}
```


- The GUI (or the HTP request) will get in return the secret name that has to be used when you want to deploy the application

7.1.5 Deploy application on NUBOMEDIA PaaS

There are two ways of deploying a NUBOMEDIA app on the PaaS: by using the GUI or by doing a HTTP POST request.

For HTTP POST Request method you should do a call with the following JSON as body on the following url: http://80.96.122.73:8081/api/v1/nubomedia/paas/app .

```
{"gitURL":"https://github.com/YourGithubAccount/nubomedia_app.git”,
  "appName":"my-app-name",
  "projectName":"nubomedia",
  “ports”:[
       {
          “targetPort”: 8080,
          “port”:8088,
          “protocol”: “TCP”
       }
  ],
  "flavor":"d1.medium",
  "replicasNumber":1,
  “secretName”:”previously-defined-secret-name”
}
```

For a more detailed description on how the PaaS API should be used, you can check the Deliverable D3.2 section 3.4.1. API.

If you prefer to use the GUI, you can login on : http://80.96.122.73:8081/#/login using your credentials.

After you input the correct credentials, you will get to the Elastic Media Manager dashboard.

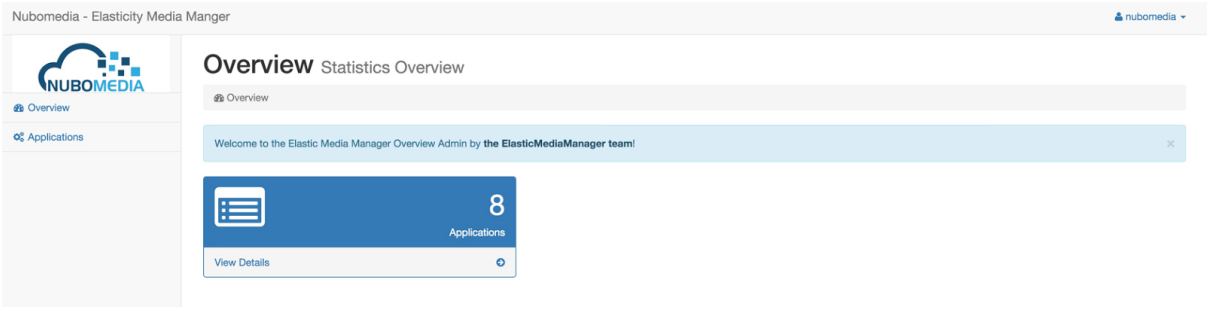

Figure 16 EMM Dashboard

If you click on Applications or the card displayed, you will get to the applications list.

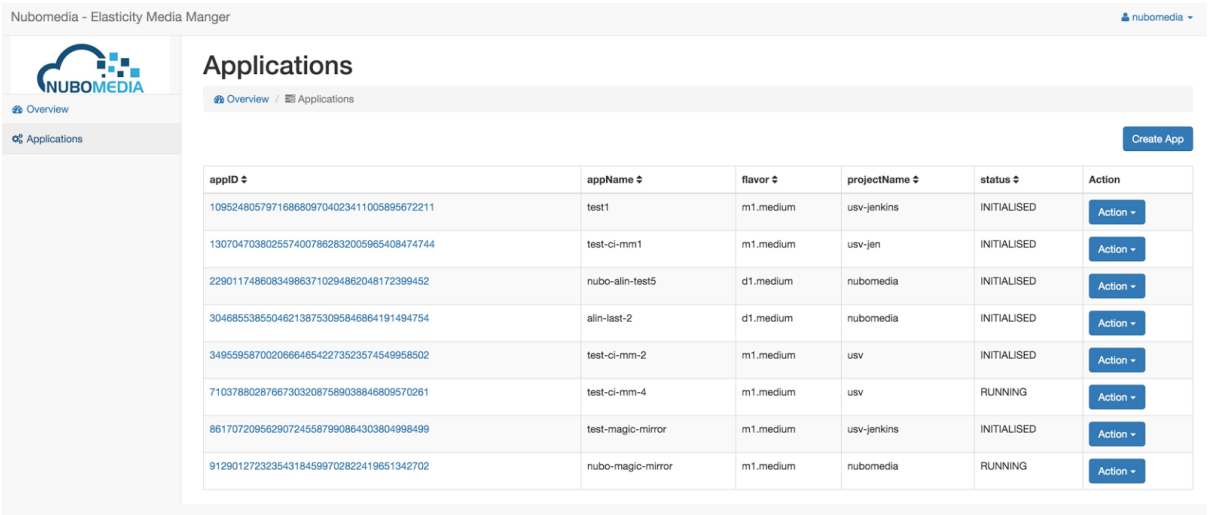

Figure 17 EMM: Applications list

From here, you can click on the “Create App” button and create your own application.

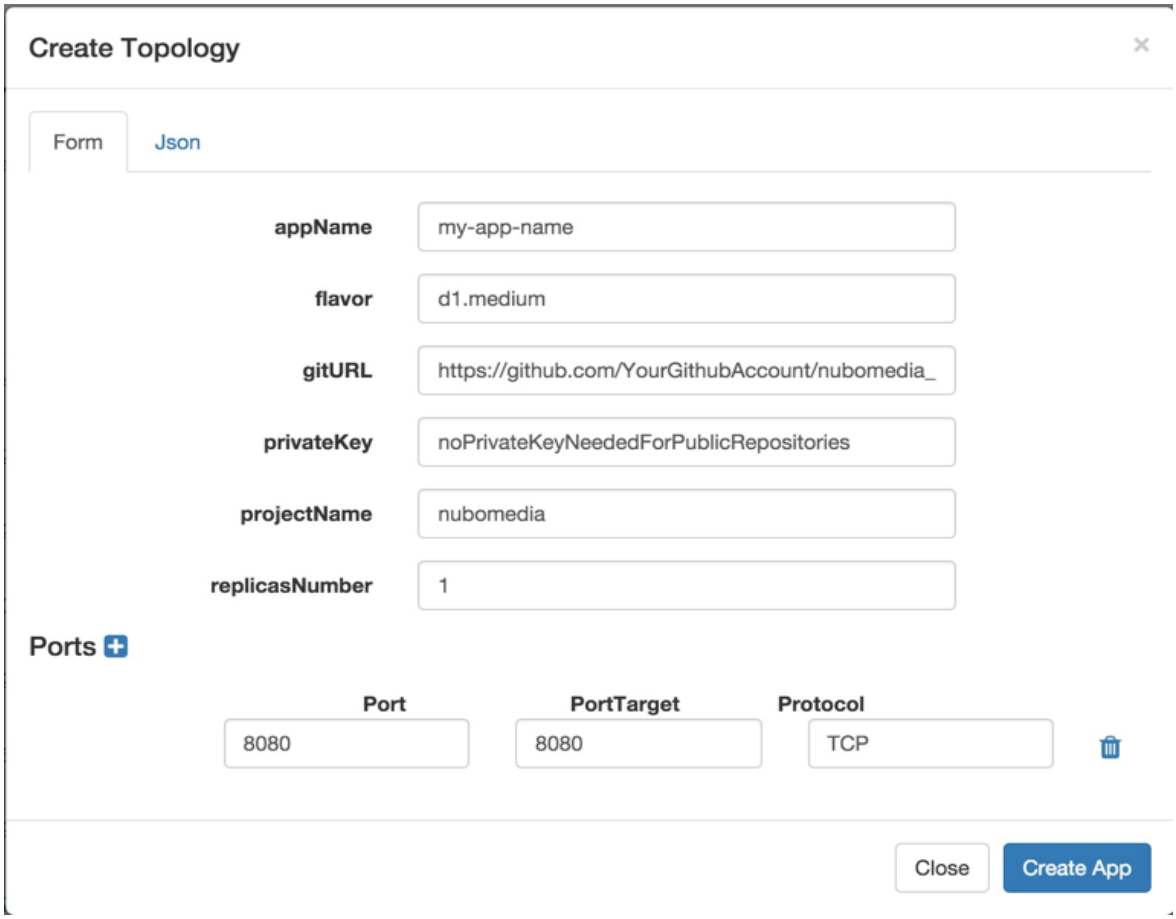

Figure 18 EMM: PaaS deployment of an application

PaaS parameters needed for creating an application:

- **gitURL**: git repository, where the jar and Dockerfile (and even other files that are necessary for the application) are committed;
- **appName**: the application name that will be used to create the DNS entry to use your application;
- **projectName**: your project name. Currently we are using the nubomedia project for all applications;
- **ports**: an object that maps the ports that are used from container to the ports that have to be exposed to outside, with relative protocol;
- **flavor**: the correspondent media server flavor that OpenBaton will instantiate. Currently, we use docker containers and you should choose from the following types of flavors: d1.small, d1.medium and d1.larg. For general purposes, we suggest to use d1.medium flavor;
- **replicasNumber**: generally, the number of containers that must be created by the PaaS after the building phase, should be 1;
- **secretName** (optional): the name of the secret that has to be used only if your application is on a private git repository.

At this point, we should have the app deployed on the OpenShift PaaS.

7.1.5.1 Check status of the application

Status of deployment can be checked from EMM application list showed previously or from OpenShift management console: https://paas.nubomedia.eu:8443/console

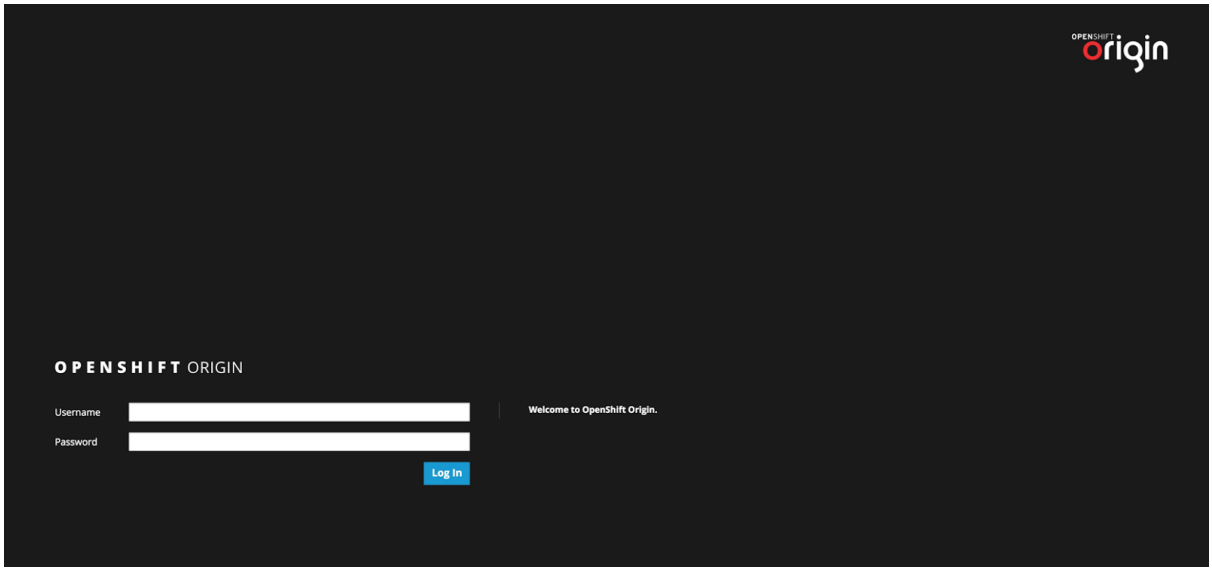

Figure 19 OpenShift PaaS console

After login, you have a screen where you should choose your project, in our case NUBOMEDIA, and in the end, you get on the dashboard where you can check if your applications have been successfully deployed.

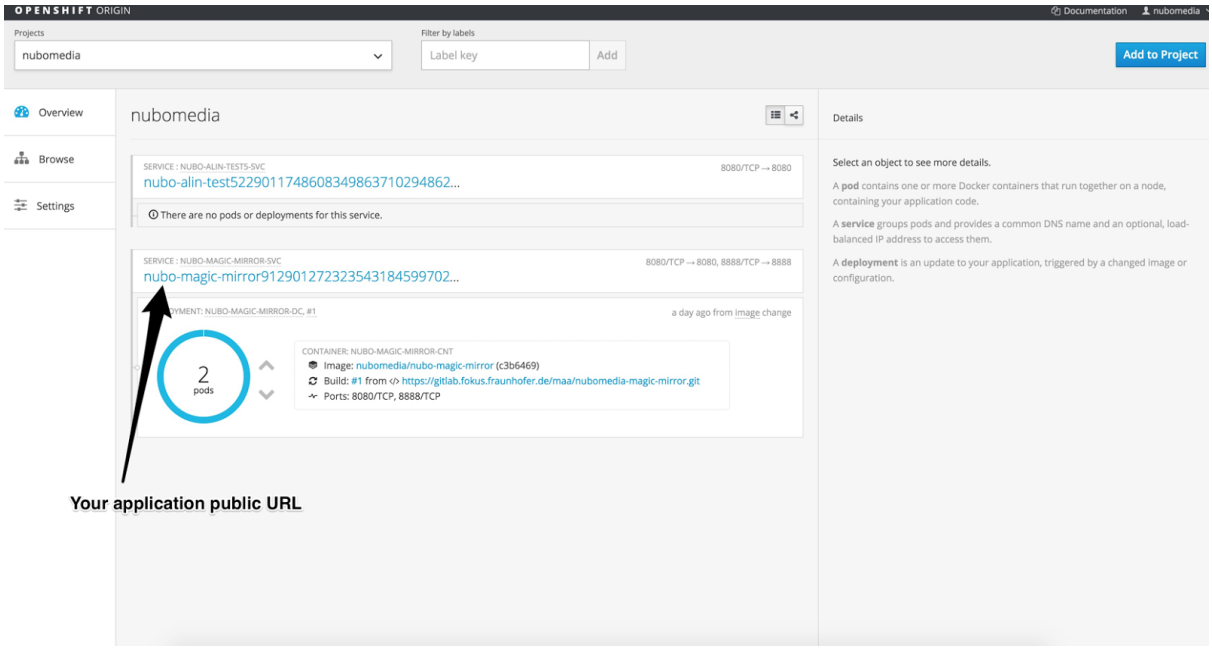

Figure 20 OpenShift application details

If you click on your application public URL, it will open the app you just deployed on a new tab.

The container having the status “Succeeded” and 0 running containers is the one used to build the docker image from the Dockerfile you provided on the Git repository.

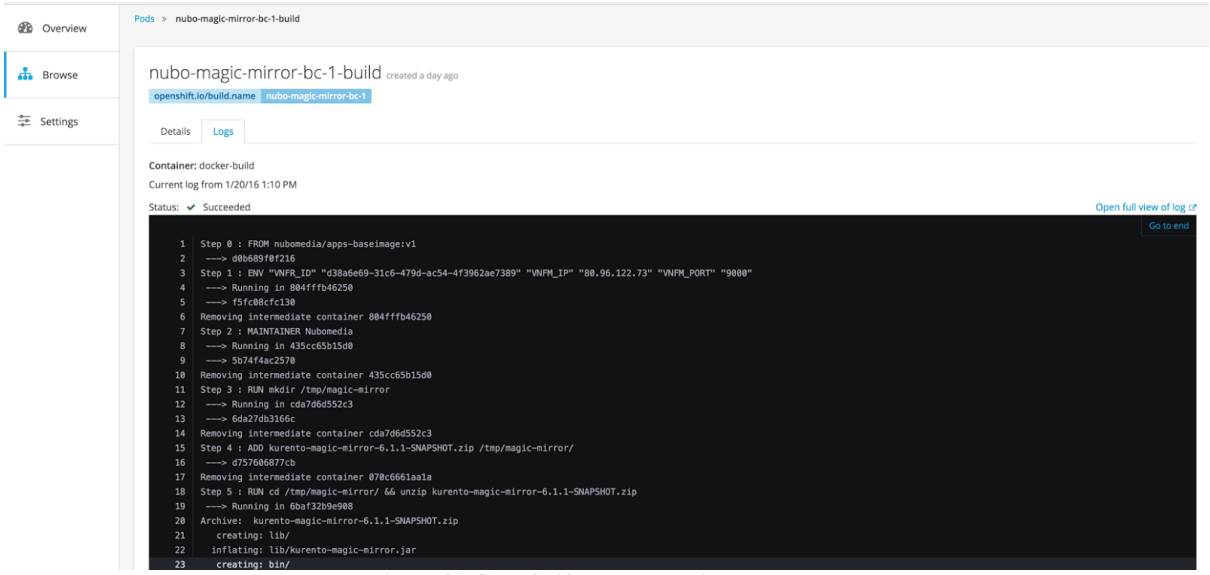

Figure 21 OpenShift console: build logs

For the application logs, you will have to go to the other Pods that are running and click on the "Logs" tab. You can also access the Docker container running the application you deployed using the "Terminal" tab.

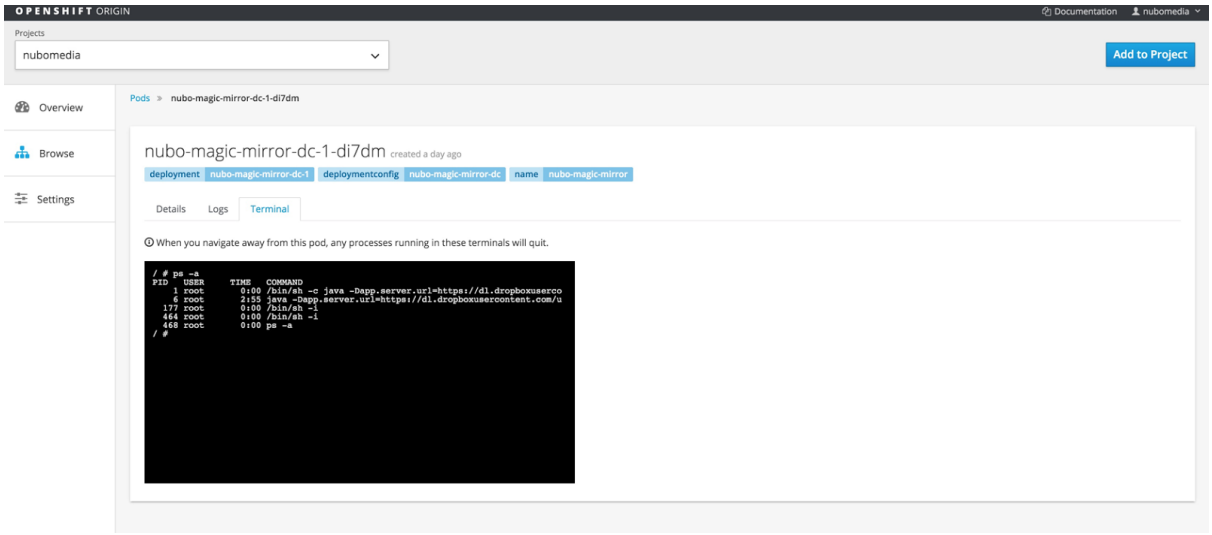

Figure 22 OpenShift console: application logs

7.2 OpenStack Horizon (Web Interface)

7.2.1 Access Horizon

You need to go to the login link: http://devconsole.nubomedia.eu and input the needed credentials.

Credentials to access the testbed are found on the wiki:

- https://www.nubomedia.eu/redmine/projects/nubomedia/wiki/Accessing_NUBOMEDIA_testbed

When you are logged in, you will see all running instances.

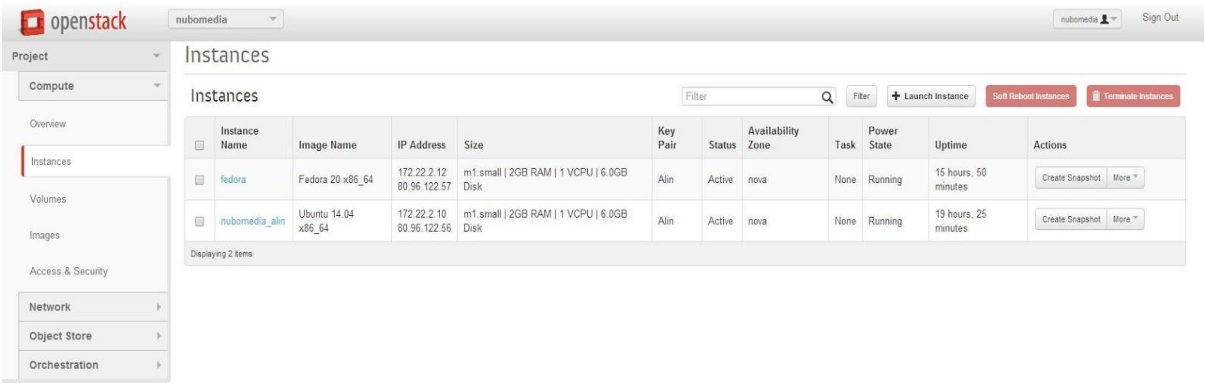

Figure 23 Horizon VM Management

7.2.2 Create an instance

When you are logged in, you can Launch Instance from the upper right.

7.2.2.1 On the first tab you will need to input the:

- **Instance Name**, which can be whatever name you want
- **Flavor**. A flavor is a virtual hardware template that is defining the size for RAM, disk, number of cores, and so on. You can see the Flavor Details after you have defined a flavor in the right side.
- **Instance Count**. The number of instances that will be lunched using this template.
- **Instance Boot Source.** Here you can choose to boot from image, boot from a volume or boot from a snapshot. If you want to start a new clean instance you should chose *Boot from image*.
- **Image name.** At this point, you choose the Linux distribution you want to run. Currently there are two types: Fedora 20 x86_64 and Ubuntu 14.04 x86_64.

7.2.2.2 On the second tab:

You will need to define your public Key Pair that you will use when you connect to the instance, and the Security Group. For default, the security group will permit connections on all ports of the instance.

When all these fields are completed, you can start to click on Lunch in the right down of the popup.

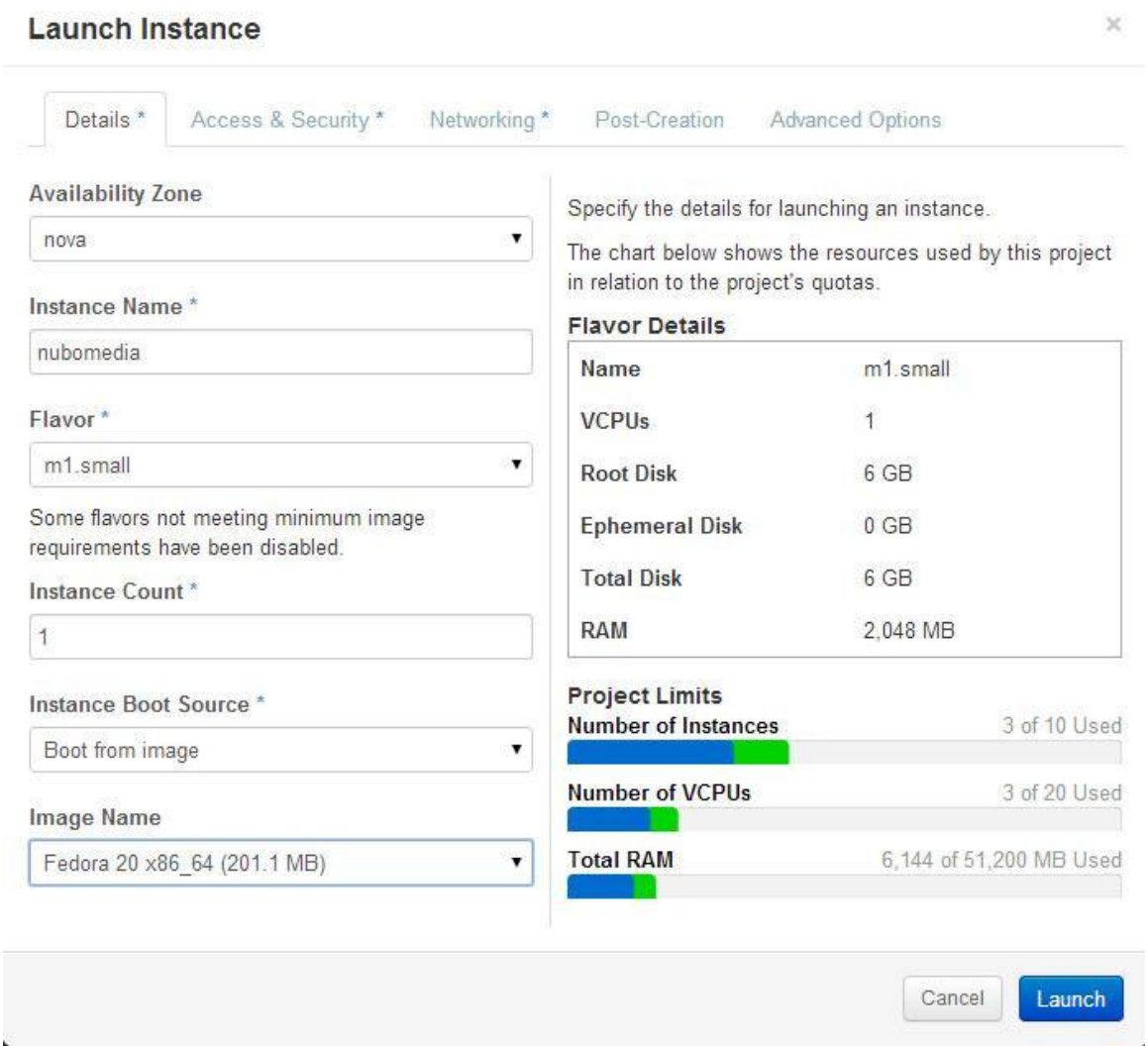

Figure 24 Horizon VM Management Launch Instance

7.2.3 Associate a floating IP

Once the instance in Power State Running, you can associate to it a Floating IP. This means you can add a public IP address to it, because instances initially have only OpenStack internal IP addresses.

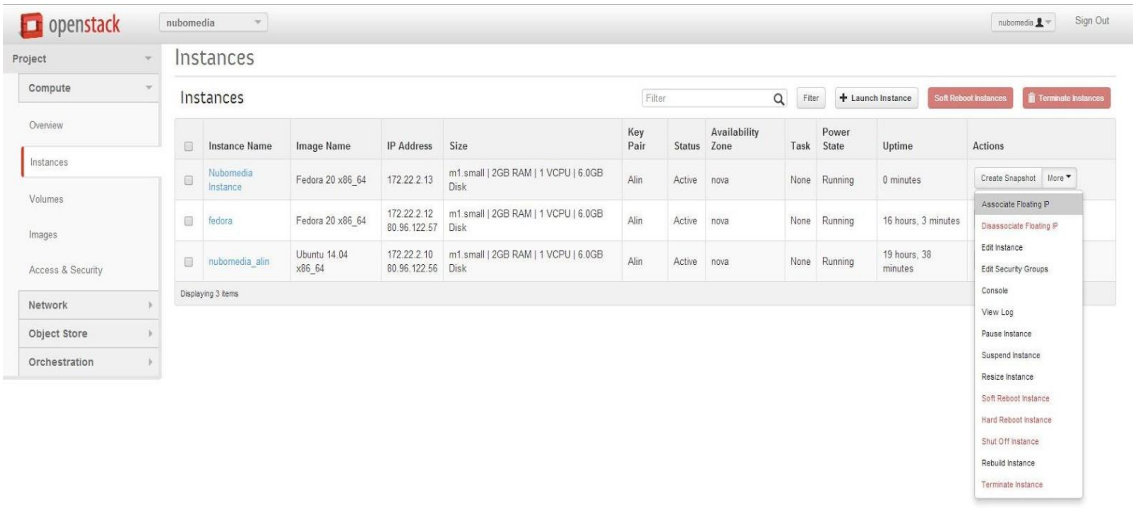

Figure 25 Horizon VM Management Associate IP

Then, if No IP addresses are available, you should click on + and, then, from the Pool dropdown, you should choose external and Allocate IP.

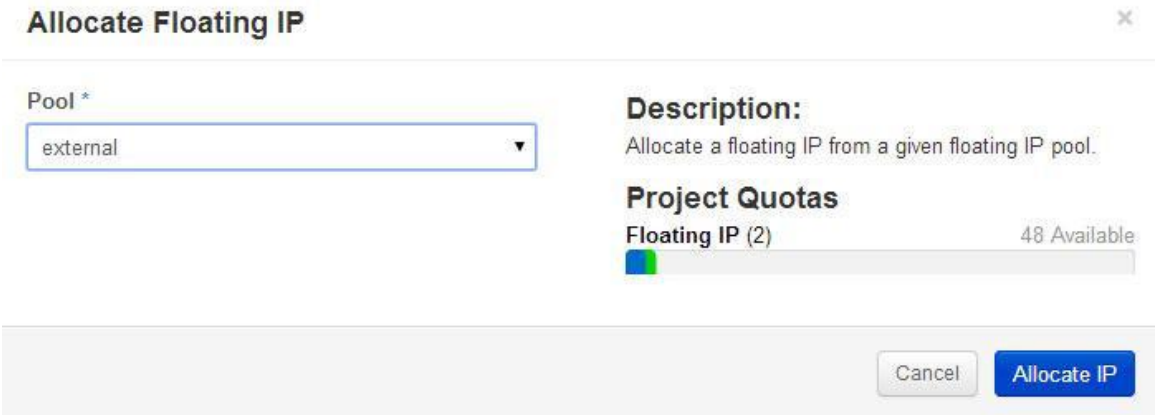

Figure 26 Horizon VM Management Allocate IP

Finally, when the new IP address is allocated to you, click on Associate to associate it to your instance.

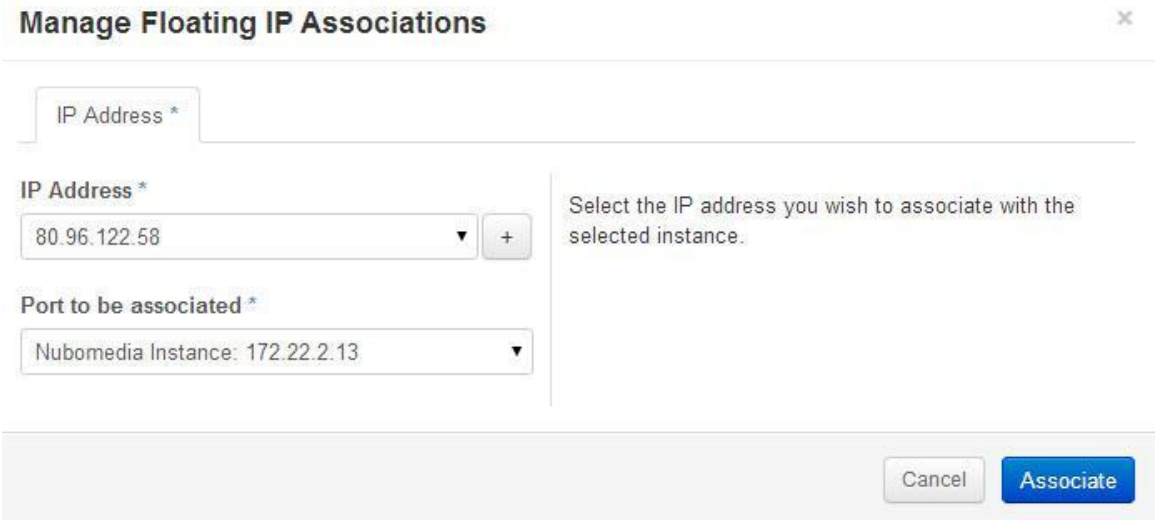

Figure 27 Horizon VM Management Confirm Associate IP

7.2.4 Connect to your instance

Now you can **connect using SSH** and the *private key* associated with the public key you have added to the public IP address associated to your instance.

7.2.5 Delete an instance

If you want to **delete a machine** or **hard reboot** it, you can use the dropdown More on the Actions tab.

7.3 OpenStack API

To interact with testbed, OpenStack API available in WP3 can be used. This method of connecting is described in detailed in the WP3 D3.1.2 Virtual Infrastructures document.

8 References

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

[2] SeleniumChromeDriver: https://code.google.com/p/selenium/wiki/ChromeDriver

[3] Xvfb is a Display Server that operates in memory without showing any output: http://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml

[4] https://github.com/openstack/nova-docker - Nova-docker repository

[5] https://github.com/usv-public/nubomedia-nova-docker

[6] http://nova-docker-pach-for-pulling-docker-containers-on-compute-nodes.readthedocs.org/en/latest/.

[7] http://creativecommons.org/licenses/by/3.0/

[8] https://software.mirantis.com/reference-documentation-on-fuel-folsom-2-1/reference-topologies-provided-by-fuel/

[9] http://wiki.li3huo.com/Gitlab

[10] https://wiki.jenkins-ci.org

9 Annex

Continue integration plan: https://docs.google.com/document/d/1oZ-wGAHkDY1qf2g1Z1PHpY4nnLg3CCuuua4rO0F-2Rc/edit#
Kurento testing infrastructure: https://docs.google.com/document/d/1fVeXSBUHuJxbv1uIpHyyUTbnM65OqVVdJFnN3Mc0FrQ/edit#heading=h.gjdgxs
Performance comparison of KMS of Docker and KVM images: https://docs.google.com/document/d/1tuHYtd6srWZSl2LMqALlY7jucvmordwtsLqE_rT54HI
[AUTONOMOUS INSTALLER LOG] https://drive.google.com/file/d/0B6xn-f43p84tQVNVTEg2UW56OWs/view