

| D6.1.3 | |
|---------------|------------|
| Version | 1.4 |
| Author | USV |
| Dissemination | PU |
| Date | 31/01/2016 |
| Status | Final |



D6.1.3: NUBOMEDIA Testbed and simulated load validation 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 2017-01-31 |
| Project type: | STREP |
| Project reference: | 610576 |
| Project web page: | http://www.nubomedia.eu http://www.nubomedia.eu/ |
| Work package | WP6: Demonstrator |
| WP leader | Constantin Filote (USV) |
| Deliverable nature: | Demonstrator |
| Lead editor: | Alin Calinciuc (USV) |
| Planned delivery date | 31/01/2017 |
| Actual delivery date | 31/01/2017 |
| 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



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

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

You are free to:

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

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

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

Under the following terms:

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

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

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

Notices:

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

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

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

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

Contributors:

Alin Calinciuc (USV)
Cristian Spoiala (USV)
Boni García (URJC)

Internal Reviewer(s):

Luis López Fernández (URJC)
Constantin Filote (USV)

Version History

| Version | Date | Authors | Comments |
|------------|------------|------------------------|--|
| 1.0 | 18/10/2016 | Cristian Spoiala (USV) | Added first tests for Metric 1.4 and 1.1 |
| 1.1 | 12/11/2016 | Alin Calinciuc (USV) | Restructured the delivery |
| 1.2 | 23/11/2016 | Cristian Spoiala (USV) | Updated tests for Metric 1.1 and 1.4 |
| 1.3 | 14/12/2016 | Boni Garcia (URJC) | Added Metric 4.1 |
| 1.4 | 28/01/2017 | Alin Calinciuc (USV) | Final review |

Table of contents

| | | |
|--|--|-----------|
| 1 | Executive summary..... | 9 |
| 2 | Hardware infrastructure | 9 |
| 2.1 | Network connectivity..... | 10 |
| 2.2 | Compute machines..... | 11 |
| 2.3 | Management machine..... | 11 |
| 2.4 | Storage system..... | 12 |
| 2.5 | Auxiliary Units..... | 12 |
| 3 | Testbed..... | 13 |
| 3.1 | Testbed procedures | 14 |
| 3.1.1 | <i>Procedures for WP3</i> | <i>14</i> |
| 3.1.2 | <i>Procedures for WP4</i> | <i>15</i> |
| 3.1.3 | <i>Procedures for WP6</i> | <i>15</i> |
| 3.2 | Setup testbed with OpenStack with RDO | 15 |
| 3.2.1 | <i>Software prerequisites:.....</i> | <i>16</i> |
| 3.2.2 | <i>Hardware prerequisites:.....</i> | <i>16</i> |
| 3.2.3 | <i>Operating system preparation for the master node</i> | <i>16</i> |
| 3.2.4 | <i>Setup the networking</i> | <i>19</i> |
| 3.2.5 | <i>Install with Packstack.....</i> | <i>19</i> |
| 3.2.6 | <i>Compute node configurations</i> | <i>20</i> |
| 4 | Software infrastructure | 24 |
| 4.1 | Gitlab..... | 24 |
| 4.1.1 | <i>Features.....</i> | <i>24</i> |
| 4.1.2 | <i>Dashboard</i> | <i>24</i> |
| 4.2 | Jenkins..... | 25 |
| 4.2.1 | <i>Features.....</i> | <i>25</i> |
| 4.3 | Ubuntu repository | 26 |
| 4.4 | TURN server..... | 26 |
| 4.5 | NUBOMEDIA Autonomous Installer | 26 |
| 5 | Continuous Integration Plan | 28 |
| 5.1 | Jenkins..... | 28 |
| 5.2 | Packer..... | 29 |
| 5.3 | Access CI tools..... | 29 |
| 6 | NUBOMEDIA tests | 30 |
| 6.1 | Testing plan..... | 30 |
| 6.2 | NUBOMEDIA integration tests | 30 |
| 6.2.1 | <i>Health tests.....</i> | <i>30</i> |
| 6.2.2 | <i>Benchmark tests.....</i> | <i>34</i> |
| 7 | Validation of NUBOMEDIA objectives | 40 |
| 7.1 | Description of the test for collecting Metric 1.1 and 1.4..... | 40 |
| 7.1.1 | <i>Methodology of the test.....</i> | <i>41</i> |
| 7.1.2 | <i>Test Environment.....</i> | <i>42</i> |
| 7.2 | Measuring degree of achievement of Metric 1.1..... | 42 |
| 7.3 | Measuring degree of achievement of Metric 1.2..... | 44 |
| 7.4 | Measuring degree of achievement of Metric 1.4..... | 45 |
| 7.4.1 | <i>Pipeline latency.....</i> | <i>46</i> |
| 7.4.2 | <i>Jitter.....</i> | <i>46</i> |
| 7.4.3 | <i>Average packet loss probability</i> | <i>47</i> |
| 7.5 | Measuring degree of achievement of Metric 4.1..... | 47 |
| NUBOMEDIA: an elastic PaaS cloud for interactive social multimedia | | 5 |



7.5.1 *Aim and Methodology* 47

7.5.2 *Experimentation and Results*..... 50

8 References..... **52**

9 Annex **53**

List of Figures:

| | |
|---|----|
| Figure 1 USV Cluster for Testbed..... | 10 |
| Figure 2 USV Internet Bandwidth..... | 11 |
| Figure 3 NUBOMEDIA Physical-Virtual infrastructure relationship | 14 |
| Figure 4 Testbed IaaS architecture | 17 |
| Figure 5 GitLab Dashboard | 24 |
| Figure 6 NUBOMEDIA Jenkins Dashboard..... | 29 |
| Figure 7 Flowchart for Health test | 31 |
| Figure 8 Magic mirror WebRTC application | 31 |
| Figure 9 Architecture of the WebRTC Loopback Test | 32 |
| Figure 10 Architecture of the benchmark test | 35 |
| Figure 11 CPU usage for KVM test without filters and 50 fake clients running on 5 KMSs | 36 |
| Figure 12 Latency for KVM test without filters and 50 fake clients running on 5 KMSs | 36 |
| Figure 13 CPU usage for Docker test without filters and 50 fake clients running on 5 KMSs..... | 36 |
| Figure 14 Latency for Docker test without filters and 50 fake clients running on 5 KMSs | 37 |
| Figure 15 CPU usage for KVM test with encoder media filter and 15 fake clients running on 5 KMSs..... | 37 |
| Figure 16 Latency for KVM test with encoder media filter and 15 fake clients running on 5 KMSs..... | 38 |
| Figure 17 CPU Usage for Docker test with encoder media filter and 15 fake clients running on 5 KMSs | 38 |
| Figure 18 Latency for Docker test with encoder media filter and 15 fake clients running on 5 KMSs..... | 38 |
| Figure 19 NUBOMEDIA Autonomous Installer steps percentage | 44 |
| Figure 20 NUBOMEDIA Autonomous Installer Benchmark | 44 |
| Figure 21 Media topology of the nubomedia-network-benchmark application..... | 48 |
| Figure 22 GUI of the nubomedia-network-benchmark application | 49 |
| Figure 23 Total bandwidth (in Mbps) in the Bronze and Gold deployments..... | 50 |
| Figure 24 Packet lost probability in the Bronze and Gold deployments | 51 |
| Figure 25 Packet lost probability ratio between Gold and Bronze..... | 52 |

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 v3 (D6.1.3) 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, as described in the NUBOMEDIA Project DoW, General validity of architecture, USV provisioned a large testbed from 120 cores to 200 cores for simulated load tests. The deliverable also includes the Metric 1.1 and 1.4 related to the auto-scaling, packet loss, jitter and average latency.

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.

In relation to Metric 1.2, as specified in the NUBOMEDIA DoW, we present the Autonomus Installer of NUBOMEDIA into an OpenStack 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 backbone¹;
- (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.

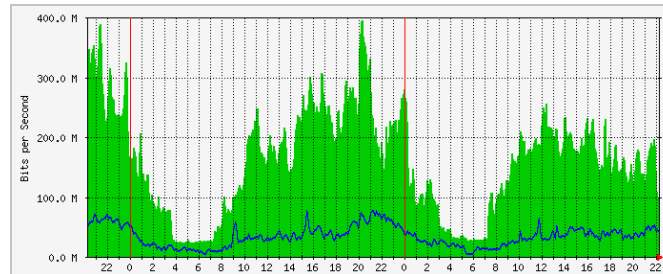


Figure 2 USV Internet Bandwidth

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
NUBOMEDIA: an elastic PaaS cloud for interactive social multimedia

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 Testbed

This section focuses on providing a detailed description of the testbed with all environments and tenants. Additionally we added instructions how to replicate NUBOMEDIA on a different setup. Also were developed test procedures for deploying and validating NUBOMEDIA artifacts. They include cleared instructions for all partners and their responsibilities in running a stable environment.

Also, 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.

On the NUBOMEDIA we have run two different environments.

- A small development testbed running 2 Docker compute nodes and one master with KVM, running on top of three compute nodes with 16Gb of RAM and two quad-core Xeon processors and another compute node running the OpenShift Origin v3. On this testbed we have two tenants:
 - admin tenant, which is used for creating and deploying NUBOMEDIA images
 - development tenant, which is used to develop and test the WP3 tasks before they are shipped into the production testbed.
- The NUBOMEDIA production testbed which is running OpenStack Kilo on 17 compute nodes with KVM and Docker hypervisors and a physical machine for the OpenShift Origin v3. On top of the production IaaS we have three tenants:
 - development testbed, on top of which we executed tests and developed functionalities before we deployed them to production.
 - validation testbed, available at <http://paas-develop.nubomedia.eu:8081> , on top of which we deploy at the end of each month the software that were created on that release and we test everything before we move to production.
 - production testbed, on top of which we deploy the latest stable version of all NUBOMEDIA components. On this tenant we have the PaaS manager, <http://paas-manager.nubomedia.eu:8081>, on top of which all developers should deploy the NUBOMEDIA applications that they are developing.
- The NUBOMEDIA management machine is running the Jenkins server and the NUBOMEDIA private GitHub repository.

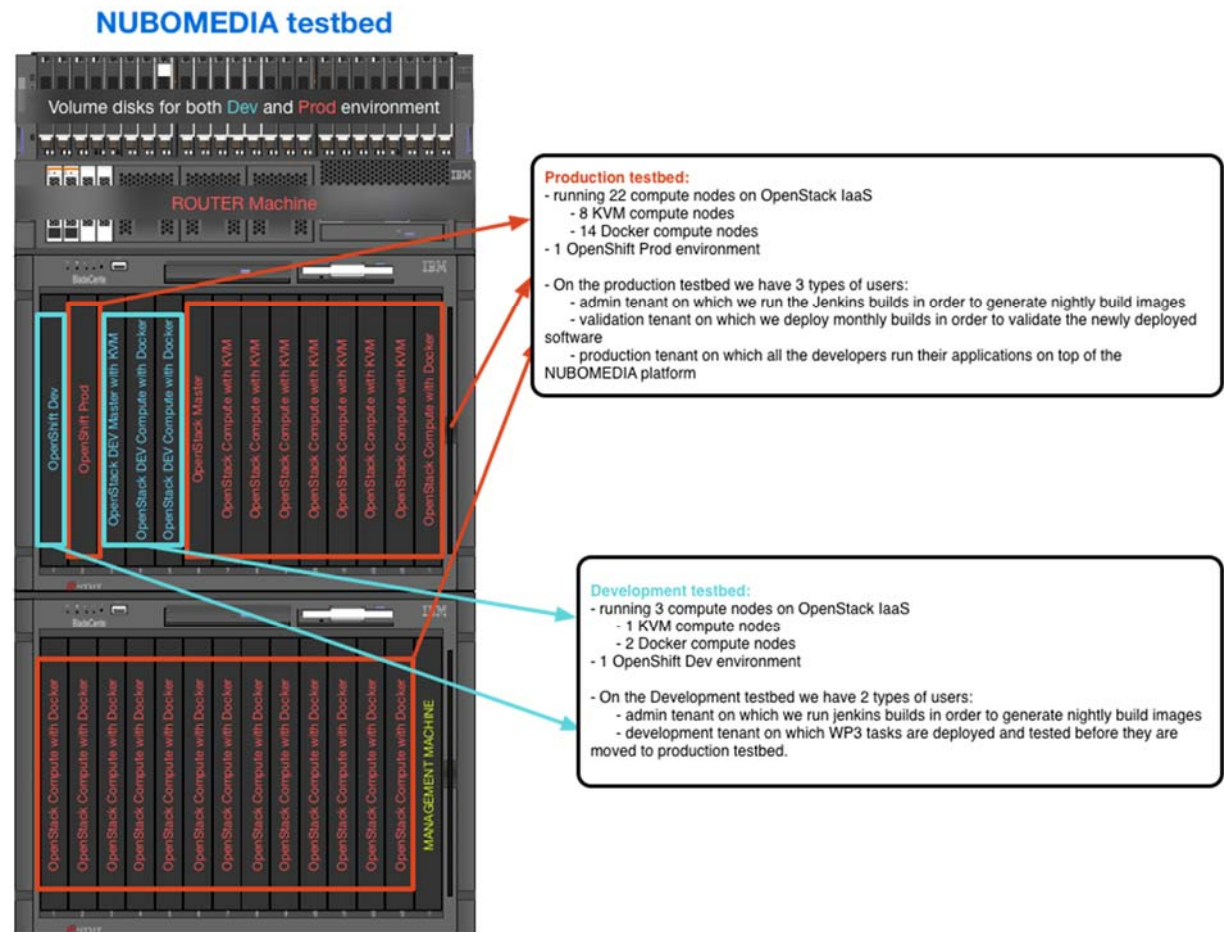


Figure 3 NUBOMEDIA Physical-Virtual infrastructure relationship

3.1 Testbed procedures

For WP6 we developed procedures for deploying and validating NUBOMEDIA artifacts on the testbed.

3.1.1 Procedures for WP3

TUB will be responsible for installing and maintaining the NFVO/EMM on the development testbed.

FOKUS will install and maintain the OpenShift development environment. This should cover also PaaS-Manager components.

USV will maintain OpenStack, in case of required features from new versions of OpenStack, an upgrade of the testbed will be necessary. Every night, USV will push images from the admin tenant (where the images are initially generated) into the development testbed. USV will take care so that it is always available a working version of a Kurento Media Server that is be used to validate the PaaS system.

Integrations tests will be executed by USV every night to validate if new development changes are not breaking existing functionality.

3.1.2 Procedures for WP4

Every month will be generated new releases. On the last week of the month, USV will ask the partners for new KMS artifacts sending a reminder on the mailing list.

For each software artifact, the partners will send the following information:

- Release notes
- Installation instructions
- Minimum KMS version needed
- At least one integration test with instructions on how to install and validate it

USV will deploy the new artifacts into the validation tenant and validate them. After deployment USV will announce the new features that will be validated for 1 week. In the announcement, the installed versions for KMS and the required versions of the API SDKs shall be provided.

After the first week of each month, new artifacts will be moved to production tenant if no issues are reported. An announcement will be made when deployment on production tenant was finalized. Partners may require changing the migration dates in cases of demos or dissemination events.

3.1.3 Procedures for WP6

On the first week of each sprint, after USV deploys the upgraded production tenant, the industrial partners will be deploying their stable version of their demonstrators. Demonstrators will be kept deployed during the month, so that PC can check the evolutions of the developments.

3.2 Setup testbed with 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 page²;
- 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 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].

3.2.1 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.

3.2.2 Hardware prerequisites:

According to the OpenStack architecture example³ 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.

3.2.3 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:

- nova-scheduler service, to allocate VMs to the compute nodes;

² <https://www.rdoproject.org/rdo/projectsinrdo/>

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

- 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.

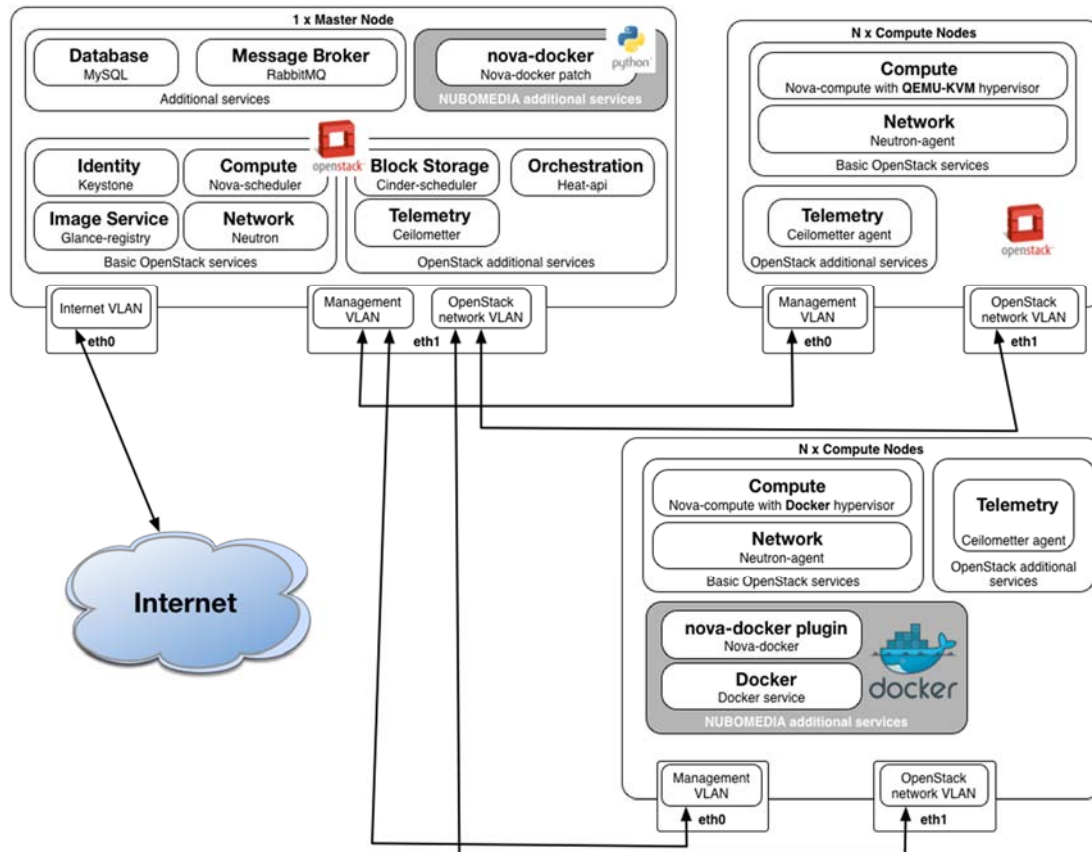


Figure 4 Testbed IaaS architecture

Figure 4 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
```

3.2.4 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.

3.2.5 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](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.
- On the master node, you should update the firewall configuration to accept

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

# 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`

3.2.6 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.

3.2.6.1 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
```

3.2.6.2 Docker image provisioning

- 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>

3.2.6.3 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.

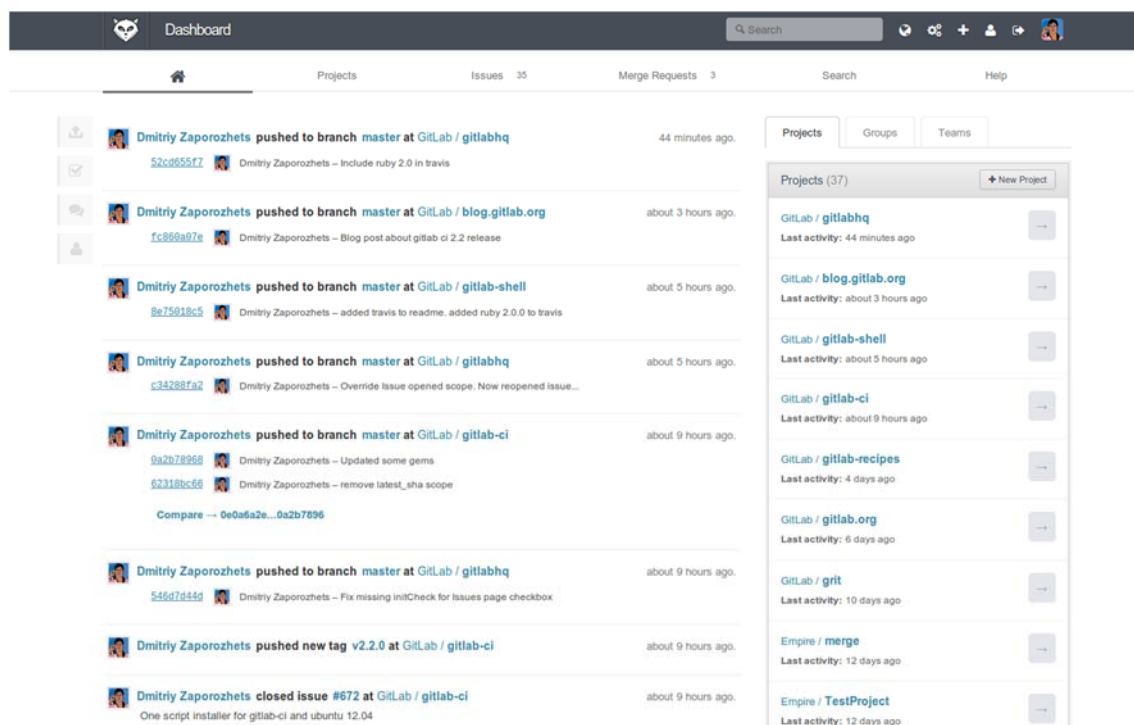


Figure 5 GitLab Dashboard

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 cope 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 NUBOMEDIA project 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 Portacle⁵;
- 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 ask whether you want to install the NUBOMEDIA platform using the automatic mode or using the wizard. We recommend you use the automatic mode in order to avoid any misconfiguration. After starting 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.

⁵ <http://portecle.sourceforge.net/>

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/1eNUhABOyXOCkhi2eZTvwieDgH6Wsoblpfufajhyikk/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@gmail.com.

5.1 Jenkins

In NUBOMEDIA, we use [Jenkins](https://jenkins-ci.org)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](https://www.packer.io)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

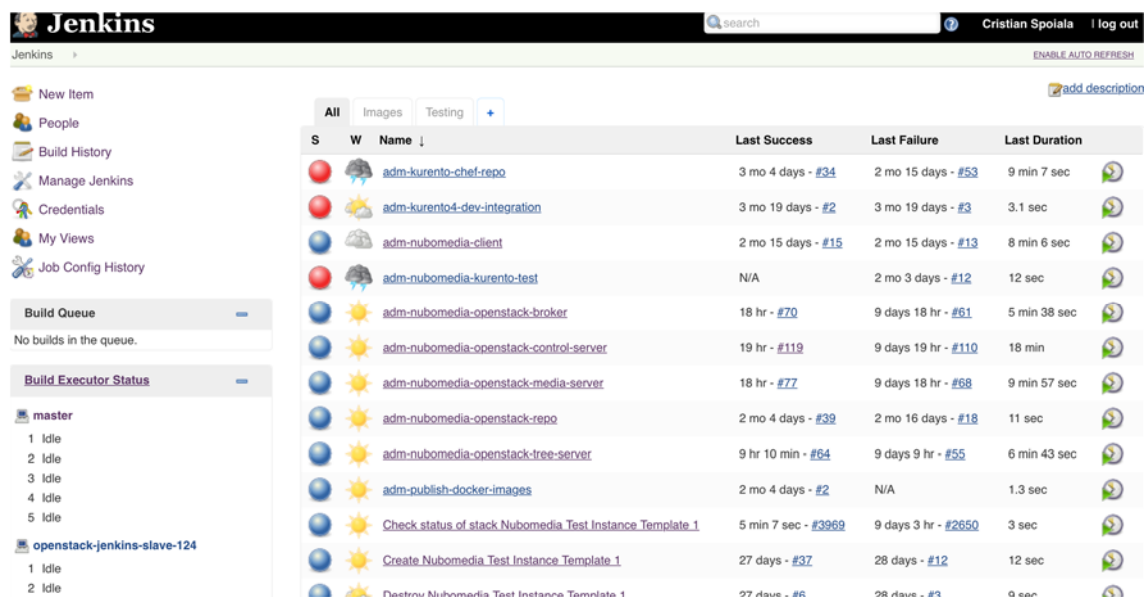


Figure 6 NUBOMEDIA Jenkins Dashboard

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 artefacts 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.

Multiple features were planned to be tested. The whole NUBOMEDIA PaaS is tested by an automatic test described more on section 6.2.1. This test is using NUBOMEDIA PaaS API to start an application, in this way is testing the functionality of the PaaS and the IaaS component. Then the multimedia workers that are running on Docker containers are tested with a specific multimedia test.

Also, 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.

Guides were provided at section 7 for partners to access the testbed and deploy applications with capabilities like VCA, AR to validate them and run on the instances of NUBOMEDIA.

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.

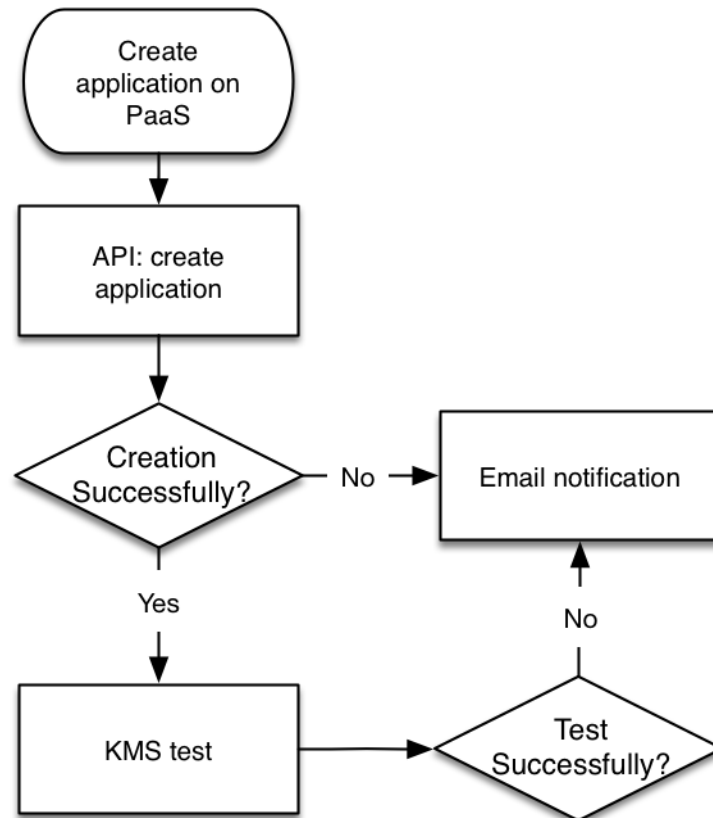


Figure 7 Flowchart for Health test

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.

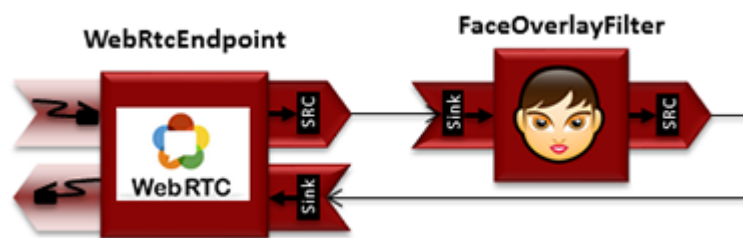


Figure 8 Magic mirror WebRTC application

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 of 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.

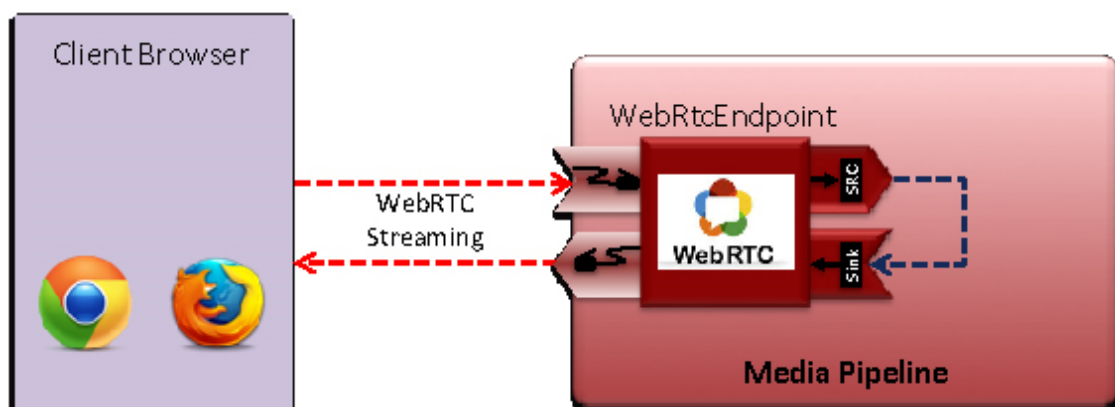


Figure 9 Architecture of the WebRTC Loopback Test

- 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 12):

- 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.

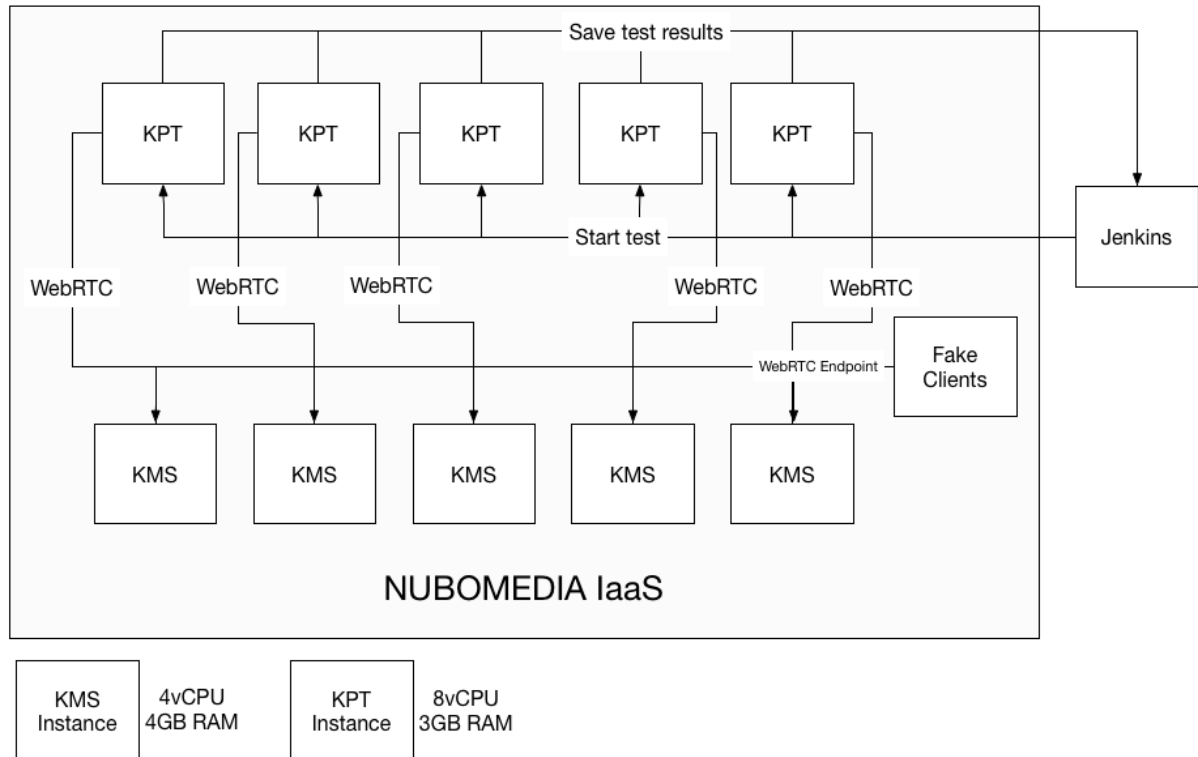


Figure 10 Architecture of the benchmark test

6.2.2.2 Configuration of the test

As shown in Figure 12 each KPT tested one KMS and was configured to use 50 fake clients on the pipeline without any media filter. When was used encoder media filter we used 15 fake clients. These fake clients are implemented using WebRTC endpoints provided by the Fake Clients Machine.

The tests for all 5 KPTs ran in parallel with either KVM option or Docker containers option.

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. For 50 fake clients and a delay of 1000ms means that after 50 seconds all clients were started. As displayed in Figure 13 to Figure 20, the ramp-up period until all fake clients are started is visible and the chart stabilizes after 50 seconds.

6.2.2.3 Test results

CPU usage and latency of the KMS are shown in charts from Figures 13 to 16 for the test without any media processing and 50 fake clients.

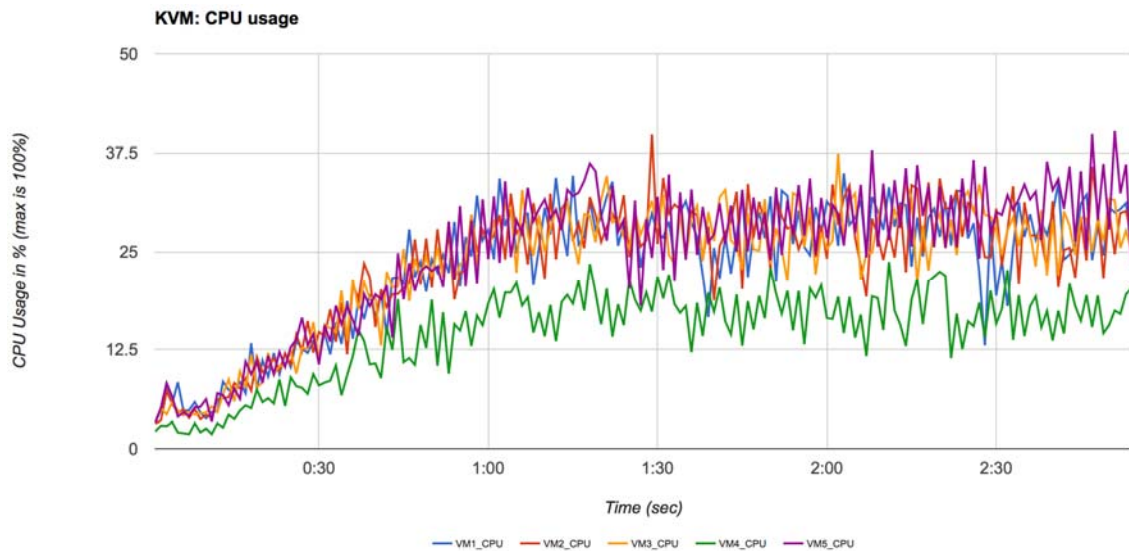


Figure 11 CPU usage for KVM test without filters and 50 fake clients running on 5 KMSs

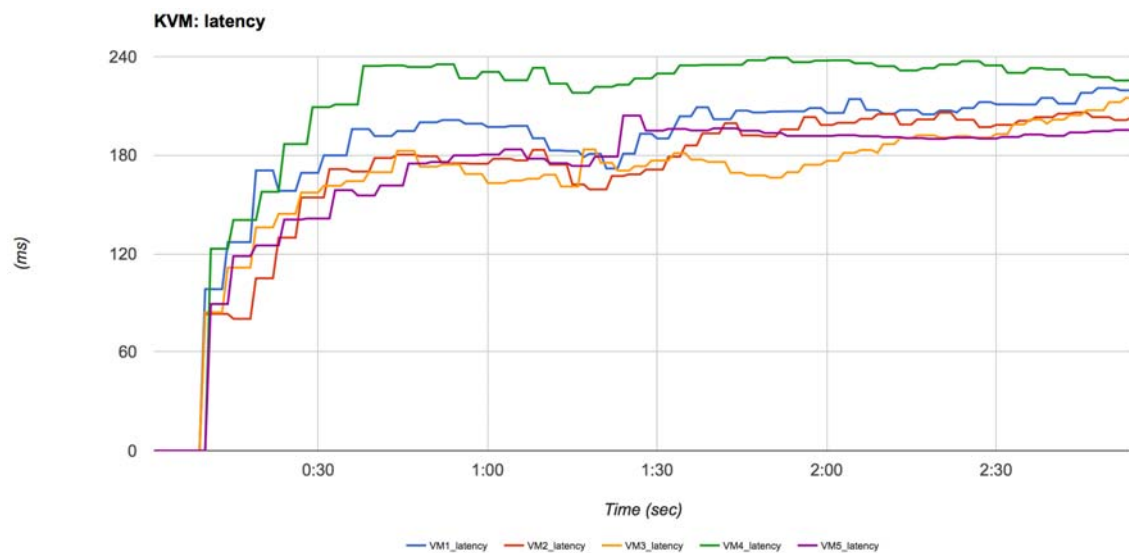


Figure 12 Latency for KVM test without filters and 50 fake clients running on 5 KMSs

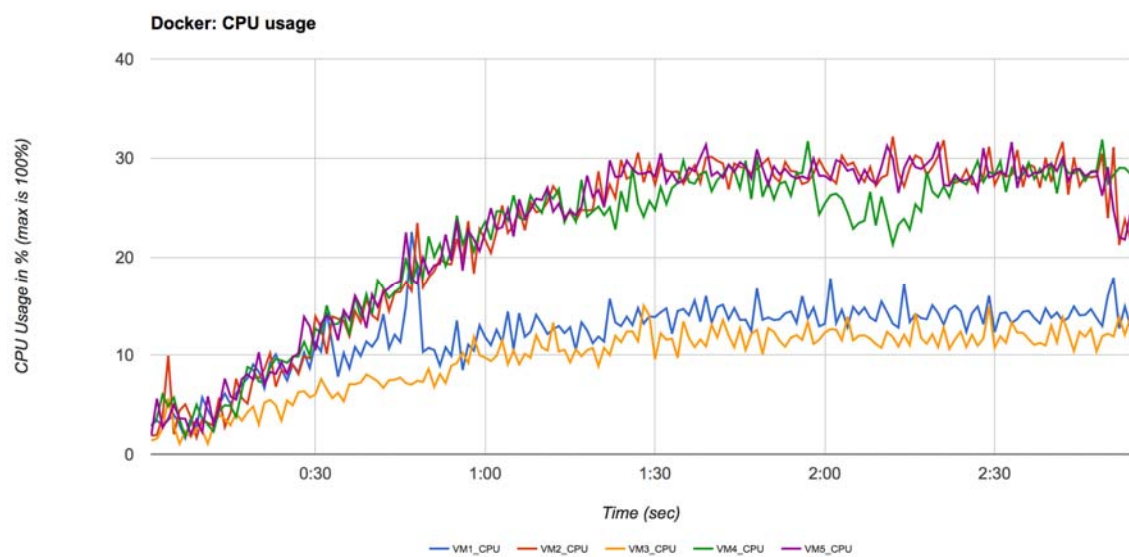


Figure 13 CPU usage for Docker test without filters and 50 fake clients running on 5 KMSs

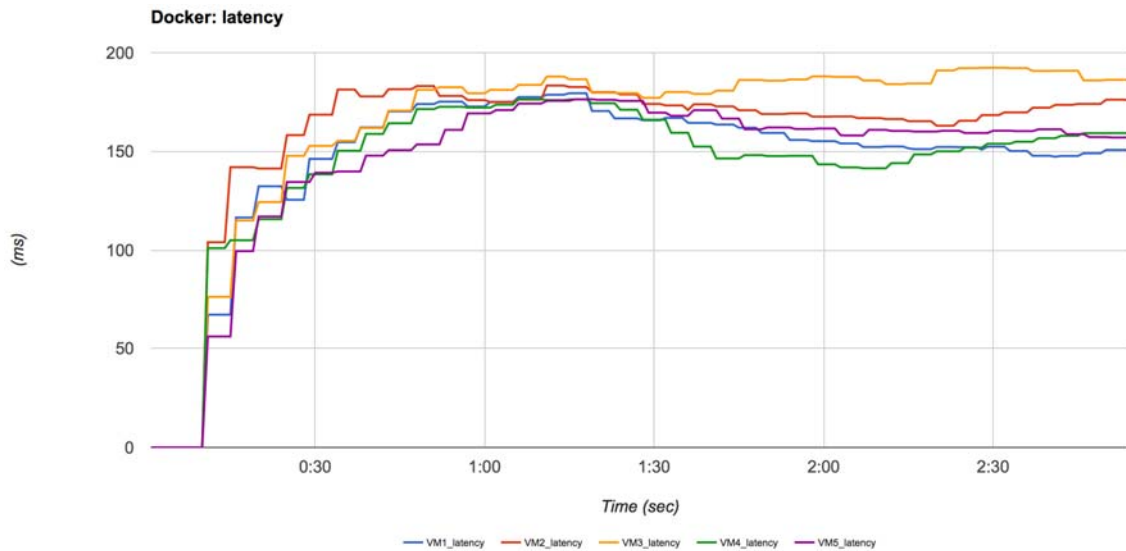


Figure 14 Latency for Docker test without filters and 50 fake clients running on 5 KMSs

Figures 13 to 16 shows clearly that the CPU usage is 5-10% lower for the Docker instances and that the latency is more stable, and a bit lower.

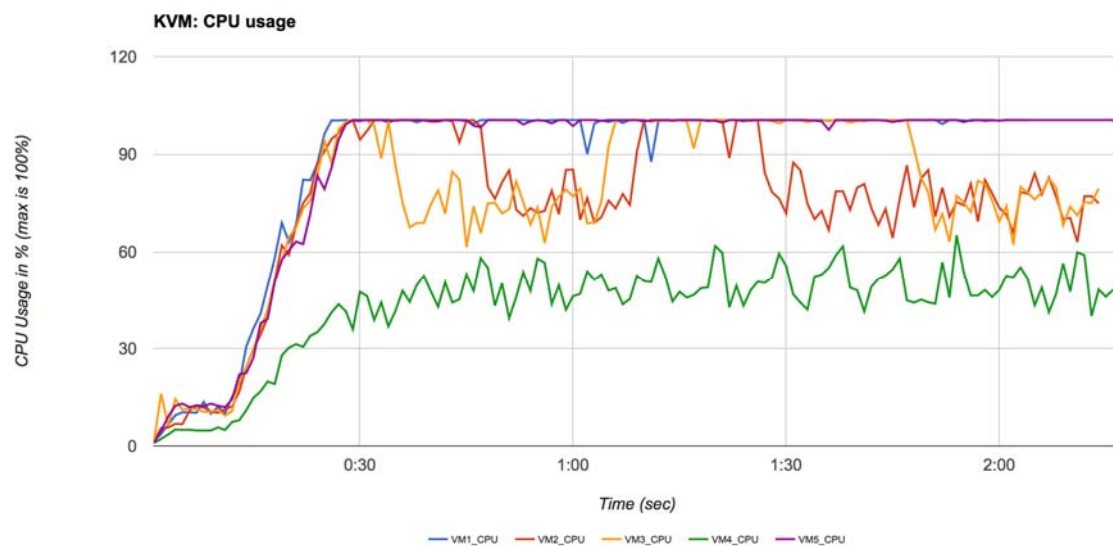


Figure 15 CPU usage for KVM test with encoder media filter and 15 fake clients running on 5 KMSs

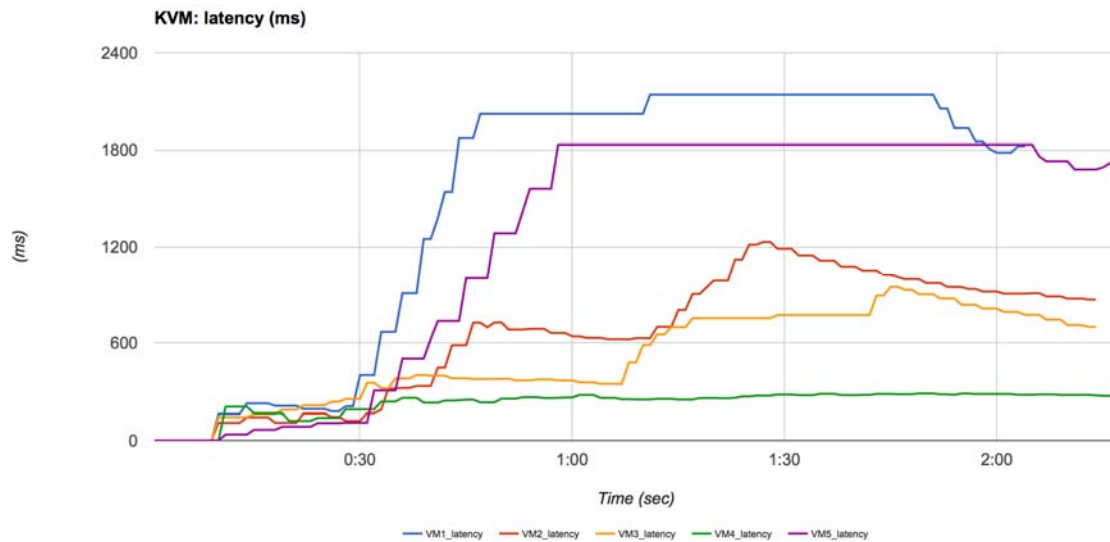


Figure 16 Latency for KVM test with encoder media filter and 15 fake clients running on 5 KMSs

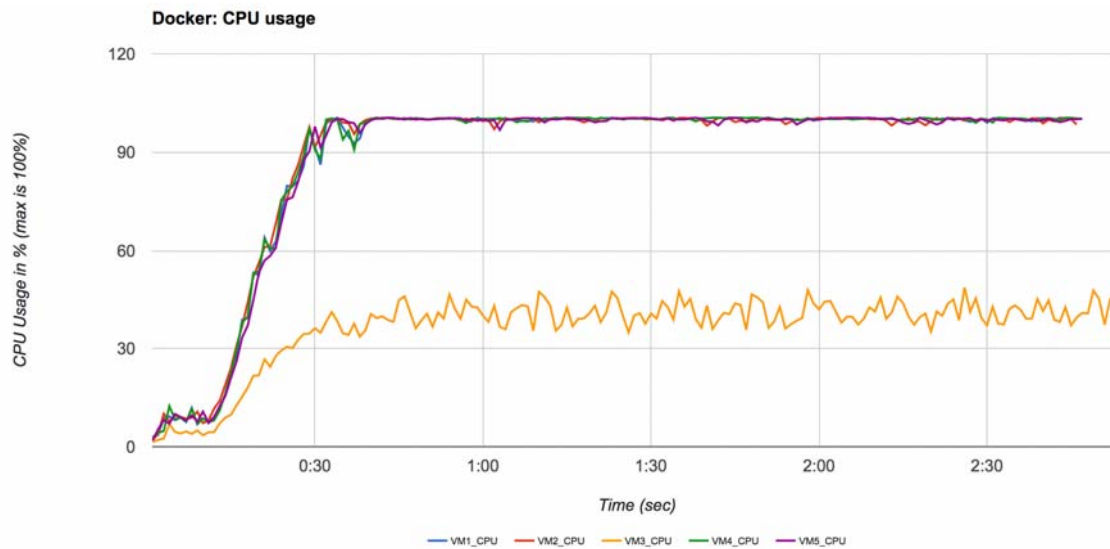


Figure 17 CPU Usage for Docker test with encoder media filter and 15 fake clients running on 5 KMSs

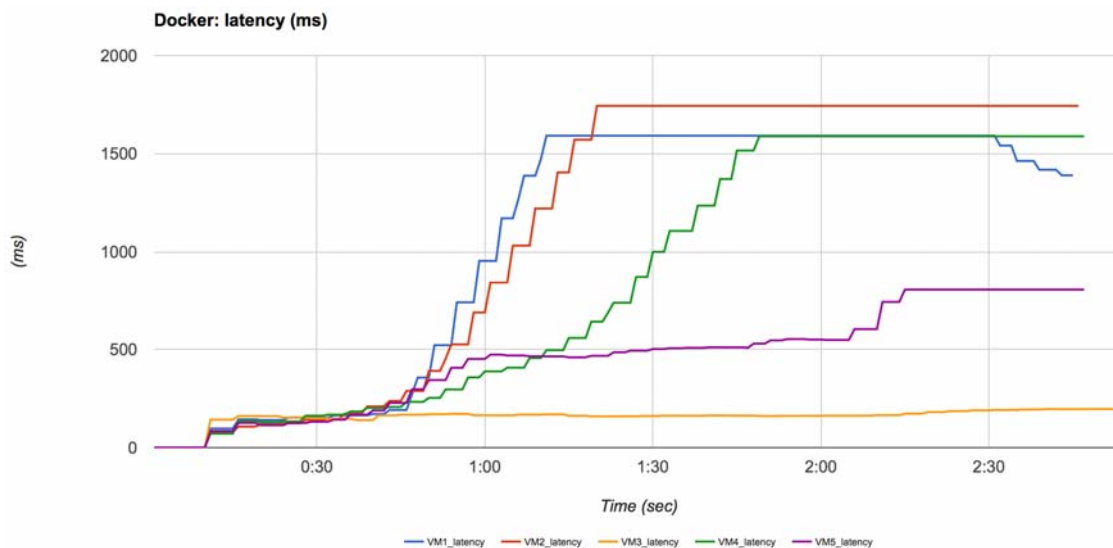


Figure 18 Latency for Docker test with encoder media filter and 15 fake clients running on 5 KMSs

6.2.2.4 Conclusion

As a result of these 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 Validation of NUBOMEDIA objectives

The technical success of NUBOMEDIA is measured through a series of indicators in a real production environment. On production testbed, we deployed all hardware available and we measured multiple indicators with simulated loads.

Objective 1: An elastic cloud infrastructure for interactive multimedia was to confirm that NUBOMEDIA has a stable architecture for an elastic scalable content-aware multimedia cloud.

Attributes to be tested:

| Objective | Attribute |
|------------|--|
| Metric 1.1 | Scalability of NUBOMEDIA |
| Metric 1.2 | Installation time |
| Metric 1.4 | Average packet loss probability (P) |
| Metric 1.4 | Average latency (L) |
| Metric 1.4 | Jitter (J, measured as latency standard deviation) |

7.1 Description of the test for collecting Metric 1.1 and 1.4

To simulate application loads for Metrics 1.1 and 1.4, we used a nubomedia-benchmark tool developed by URJC (<https://github.com/nubomedia/nubomedia-benchmark>). The tool is starting 2 Chrome based browsers that will act as a presenter and a viewer. A number of parameters are configured for the NUBOMEDIA application.

The application nubomedia-benchmark is able to consume many different KMS instances in a tree topology. The WebRTC user media from a browser acting as presenter is sent to a source WebRtcEndpoint, which is able to broadcast to a configurable number of different media pipelines (in different KMS instances) using a set of WebRtcEndpoints. Figure 21 shows this tree topology that was used for the test to gather the results for metrics 1.1 and 1.4.

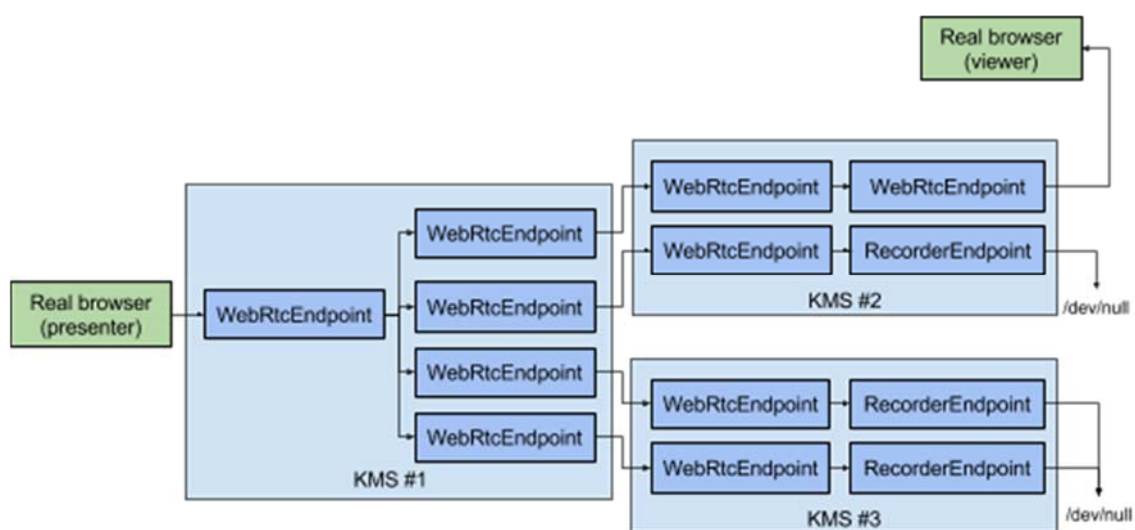


Figure 21 Tree topology of KMS's media pipeline

By using an automatic test, these parameters were automatically passed and the test started. The test was started on a Linux virtual machine running on a separate IBM System x3850 X5 server with 4 x Intel Xeon E7 4860 CPUs and 256GB of RAM, hosted at USV. For each metric, we described the parameters used.

Using specific parameters described on sub-section 7.2, the automatic test generates a detailed CSV file with statistics that are collected during the test.

The number of started KMSs from the PaaS API was collected using the following endpoint:

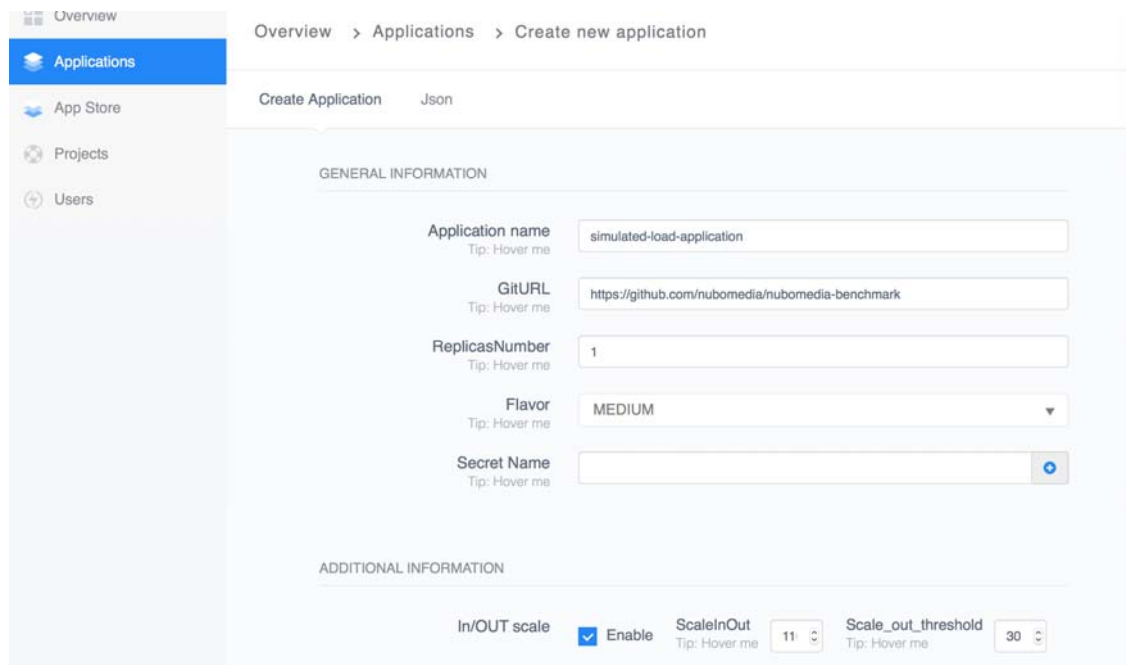
`http://PAAS_URL/vnfr/APPLICATION_ID/media-server/number`

The hardware involved in the test is the USV infrastructure described on Section 2 of this document. The topology of the network was local USV infrastructure based on Gigabit Ethernet.

The connection between the browsers running by automated test on Linux virtual machine and the deployed NUBOMEDIA application was over local network.

7.1.1 Methodology of the test

On production testbed, we configured and started the nubomedia-benchmark tool as a multimedia application with auto-scaling, as shown in the Figure 22. The application was deployed using NUBOMEDIA deployment tool.



The screenshot displays the 'Create new application' form in the NUBOMEDIA PaaS interface. The left sidebar shows navigation options: Overview, Applications (selected), App Store, Projects, and Users. The main content area is titled 'Create Application' and 'Json'. It is divided into two sections: 'GENERAL INFORMATION' and 'ADDITIONAL INFORMATION'.

GENERAL INFORMATION:

- Application name:** simulated-load-application
- GitURL:** https://github.com/nubomedia/nubomedia-benchmark
- ReplicasNumber:** 1
- Flavor:** MEDIUM
- Secret Name:** (empty field with a blue icon)

ADDITIONAL INFORMATION:

- In/OUT scale:** ☒ Enable
- ScaleInOut:** 11
- Scale_out_threshold:** 30

Figure 22 Deployment parameters in NUBOMEDIA PaaS for simulated load test application

By design a NUBOMEDIA application in auto-scaling mode will start by default 2 KMSs but will be connected only to 1 KMS. In this way PaaS ensure that a KMS is started when the load will increase.

Using the automatic test capability of nubomedia-benchmark tool, we started to add simulated load to the test and recorded the results. Furthermore, internally, the tool collected detailed results that were exported in a CSV file stored on local filesystem.

The results for each metric are displayed on next subsections.

7.1.2 Test Environment

Specifications for Linux client test machine:

| Specification | Value |
|---------------|----------------|
| OS | Ubuntu 14.04.4 |
| RAM | 10 GB |
| CPU | 12 vCPU |

Specifications for the KMS instances:

| Specification | Value |
|----------------|--------------|
| OS | Ubuntu 14.04 |
| KMS version | 6.5 |
| Docker version | 1.7 |
| CPU | 2 vCPU |
| RAM | 2 GB |
| DISK | 5 GB |

NUBOMEDIA specifications:

| Specification | Value |
|----------------|------------|
| Release | 8.4 |
| OpenShift | V3 1.1.1.1 |
| Docker version | 1.7 |
| Java | 1.8 |

7.2 Measuring degree of achievement of Metric 1.1

Objectives for metric 1.1 was for NUBOMEDIA PaaS to scale by using all available resources. The PaaS has to scale up and down based on the load from multimedia applications.

Tested scenario was to simulate load and measure how NUBOMEDIA scaled and if technological requirements are fulfilled. The following metric was measured for this scenario:

| Metric | Fully successful | Reasonably successful | Unsuccessful | Achieved value |
|--------------------------|------------------|-----------------------|-----------------|----------------|
| Scalability of NUBOMEDIA | Over 200 Cores | Over 100 cores | Under 100 cores | 188 cores |

As shown in the table above, for metric to be reasonably successful, it should be over 100 CPU cores. To be fully successful, the scalability of NUBOMEDIA should be over 200 CPU cores.

The automatic test was configured with following parameters:

| Description | Parameter | Value |
|-----------------------------|----------------------|---|
| URL to NUBOMEDIA app | app.url | https://h32120d44.apps.nubomedia-paas.eu |
| Number of KMSs for the tree | extra.kms | 120 |
| Rate to add new KMSs | kms.rate | 120 |
| Initial session number | init.session.number | 1 |
| Gather internal latency | kms.internal.latency | true |
| Use KMS tree topology | kms.tree | true |

Full command used for automatic test:

```
mvn test -Dapp.url=$URL -Dinit.session.number=1 -Dkms.tree=true -Dextra.kms=120
-Dkms.rate=120 -Dkms.internal.latency=true
```

As displayed in the Figure 23, using simulated load we achieved 94 virtual machines with 2 vCPU each, resulting a usage of 188 CPU cores.

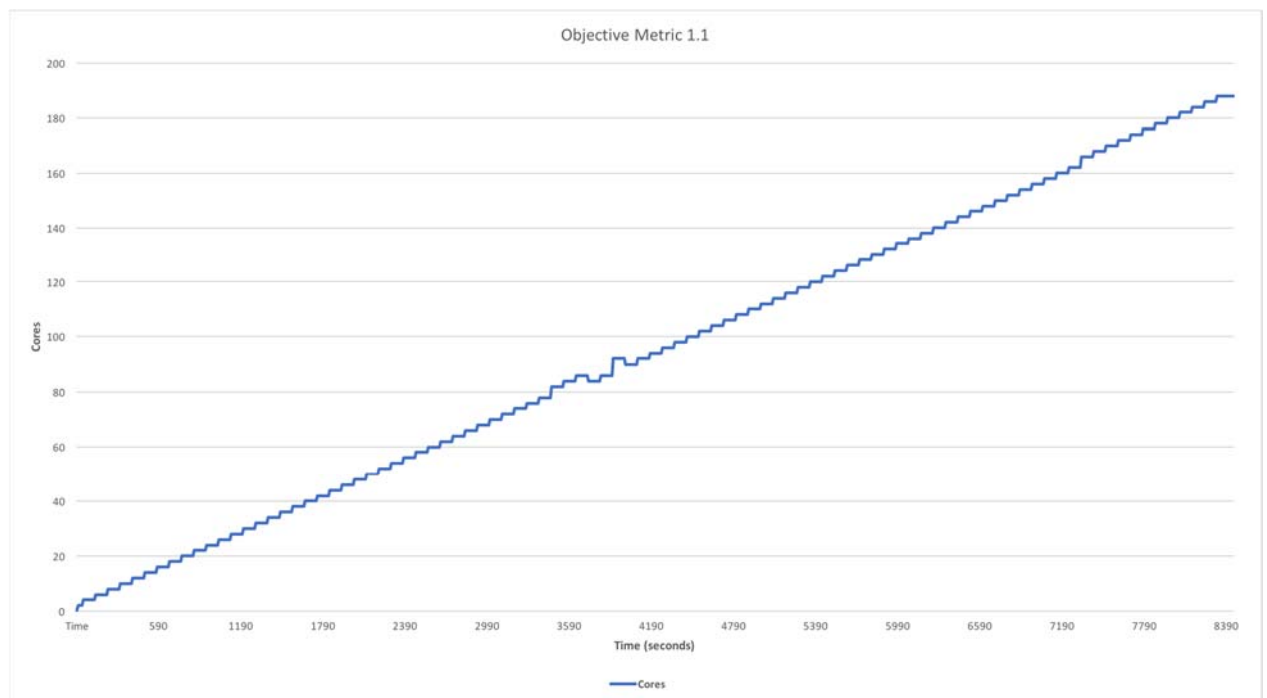


Figure 23 Auto-scaling result of simulated load test

7.3 Measuring degree of achievement of Metric 1.2

Metric 1.2 was already presented in deliverable D6.1.2 last year. This year we updated the Autonomous Installer with the latest software developed inside the consortium and re-run the test in order to measure the time needed to deploy the NUBOMEDIA platform. For this we run the Autonomous Installer on a new environment, and, like last year we used the log file that the installer provides to extract the time needed for each step of the Autonomous Installer.

NUBOMEDIA Autonomous Installer steps

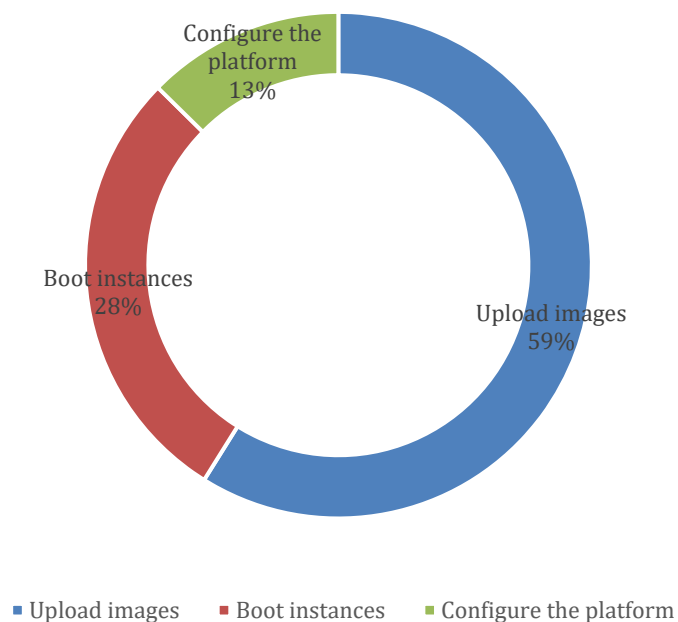


Figure 19 NUBOMEDIA Autonomous Installer steps percentage

Figure 19 shows that most of the installation time is spent on uploading the images from the NUBOMEDIA repository to the Glance registry of the IaaS.

This year we found that, with the latest updates, the installation of the NUBOMEDIA platform is now done in 2572 seconds, meaning **42 minutes and 52 seconds**, as can be seen in Figure 20.

NUBOMEDIA Autonomous Installer Benchmark

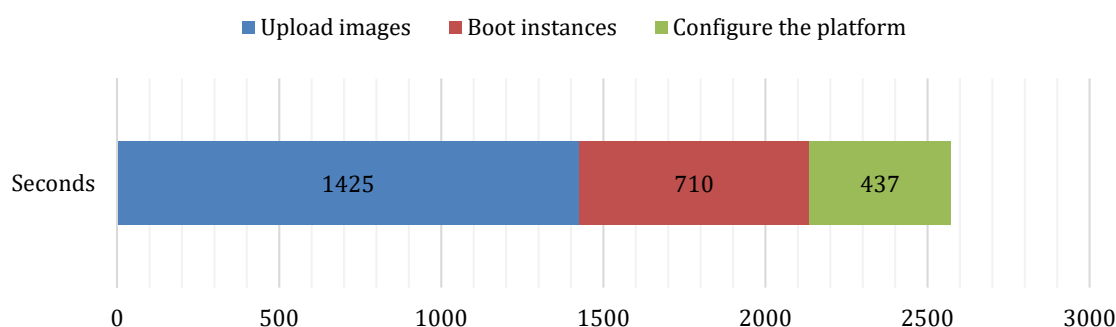


Figure 20 NUBOMEDIA Autonomous Installer Benchmark

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. Gathering the IaaS and PaaS requirements can take up to 2 hours because system administrators might need to setup a different tennant for testing the NUBOMEDIA platform and to create the OpenShift Keystore.

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.

Next, deploying a basic NUBOMEDIA application on top of the PaaS should require less than 1 hour, because the app should be configured in an appropriate way in order to be able to run it on top of the NUBOMEDIA platform. The expert administrator should follow the developer guidelines described here <http://nubomedia.readthedocs.io/en/latest/>. If the deployment of the app is not successful then the administrator should proceed in investigating the logs looking for deployment problems, and after fixing them he should try do a new deployment.

All these steps together should allow an expert administrator to deploy NUBOMEDIA in less than 5 hours. If the expert system administrator doing the NUBOMEDIA deployment is stuck at any of the steps in deploying the platform or in preparing the prerequisites needed for installing it, then, he should ask for help in the NUBOMEDIA Group forum locate here: <https://groups.google.com/forum/#!forum/nubomedia-dev>.

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 D6.1.3 deliverable successfully meet 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.

7.4 Measuring degree of achievement of Metric 1.4

Objectives for Metric 1.4 are related to the Quality of Service (QoS) and Quality of Experience (QoE) and to the performance of NUBOMEDIA infrastructure with relevant indicators like network performance and quality.

The most appropriate metrics for measuring QoS are: average packet loss probability (P), average latency (L), and jitter (J). Given its complexity, we only consider P, L, and J along the pipeline, which means only along the path the media flow traverses within a NUBOMEDIA instance.

For collecting the data, we used the detailed CSV file generated by the nubomedia-benchmark tool.

The following metrics and success criteria associate according to DoW were collected for Metric 1.4 Quality of Service (QoS) and Quality of Experience (QoE):

| Metric | Fully successful | Reasonably successful | Unsuccessful | Achieved value |
|--------|------------------|-----------------------|--------------|----------------|
|--------|------------------|-----------------------|--------------|----------------|

| Average packet loss probability (P) | Under 0.05 | Between 0.1 and 0.05 | Over 0.1 | 0 – No loss |
|-------------------------------------|------------|------------------------|------------|-------------|
| Average latency (L) | Under 50ms | Between 50ms and 250ms | Over 250ms | 0.3ms |
| Jitter (J) | Under 50ms | Between 50ms and 250ms | Over 250ms | 35ms |

7.4.1 Pipeline latency

Mean value for latency measured for metric L was 0.3ms, and according to the Table above, to be successful was supposed to be under 50ms. A reasonable successful result was supposed to be between 50ms and 250ms. Standard deviation value for latency measured was 0.08ms.

Figure 24 is displaying the average latency during the test, within the media pipeline.

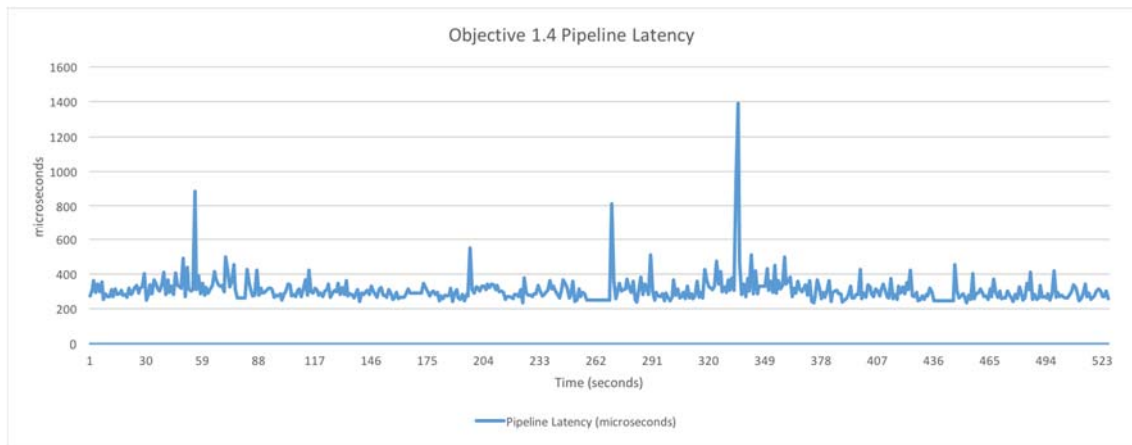


Figure 24 Average latency during simulated load test

The results obtained are very successfully and confirms the stability and scalability of NUBOMEDIA platform.

7.4.2 Jitter

Another metric to be analysed J was fully successful with a mean value of 35.4ms. To be fully successful was supposed to be under 50ms. Standard deviation value for Jitter was 20ms.

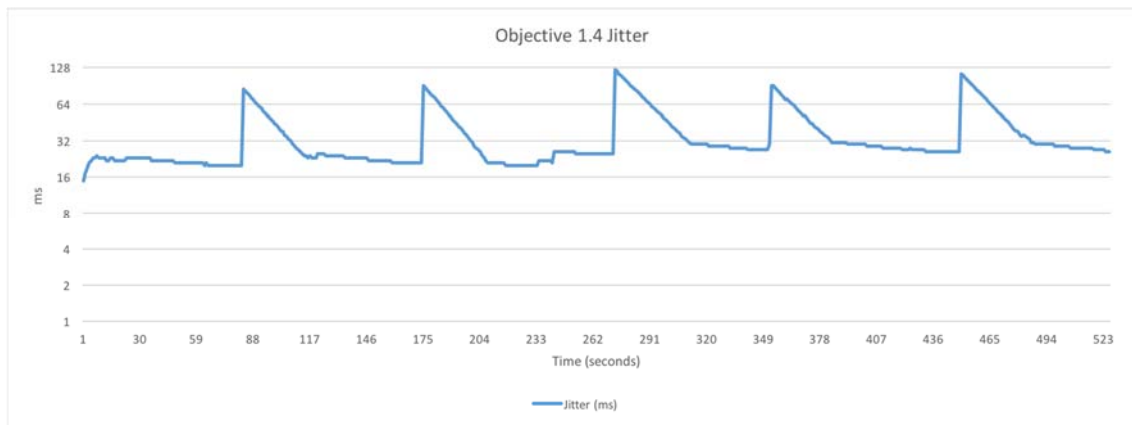


Figure 25 Jitter parameter during simulated load test

As shown in the Figure 25, Jitter doesn't seem to grow. It is not in the same order of magnitude compared to latency because is a network jitter and depends on the infrastructure network. Infrastructure jitter is standard deviation of the infrastructure latency.

7.4.3 Average packet loss probability

As stated on table above for the metric P to be successful indicator should be under 0.05. We achieved the value of 0 for P so the metric is fully successful.

For P we cannot draw any figure as the was not lost any packet during the test.

We consider that the P, L, and J metrics for measuring QoS presented in this D6.1.3 deliverable successfully met the Metric 1.4 Quality of Service (QoS) and Quality of Experience (OoE) of the Validation of Objective 1: An elastic cloud infrastructure for interactive multimedia in DoW NUBOMEDIA project.

7.5 Measuring degree of achievement of Metric 4.1

7.5.1 Aim and Methodology

Metric 4.1 requires the information gathered from the system to improve the QoE provided to users. In order to gather meaningful data, we have created a NUBOMEDIA application specifically designed to simulate the high load condition need to provide an indicator showing the relative improvement on QoS obtained by the Service-Level Agreement (SLA) layer. This application is called nubomedia-network-benchmark, and it can be deployed in the NUBOMEDIA PaaS as any other application. The GitHub repository of this app is inside the NUBOMEDIA organization:

<https://github.com/nubomedia/nubomedia-network-benchmark>

In this application, the WebRTC user media from a browser feeds a media pipeline inside a media server (KMS) within NUBOMEDIA. This media pipeline has a single input WebRtcEndpoint connected to N WebRtcEndpoints. This number N can be configured from the application GUI. Each WebRtcEndpoint-x is connected to another WebRtcEndpoint-y in another media pipeline of another KMS. In this second media pipeline, N RecorderEndpoints are connected to each WebRtcEndpoint. This is done to drive media out of the pipeline. This topology is depicted in the following diagram:

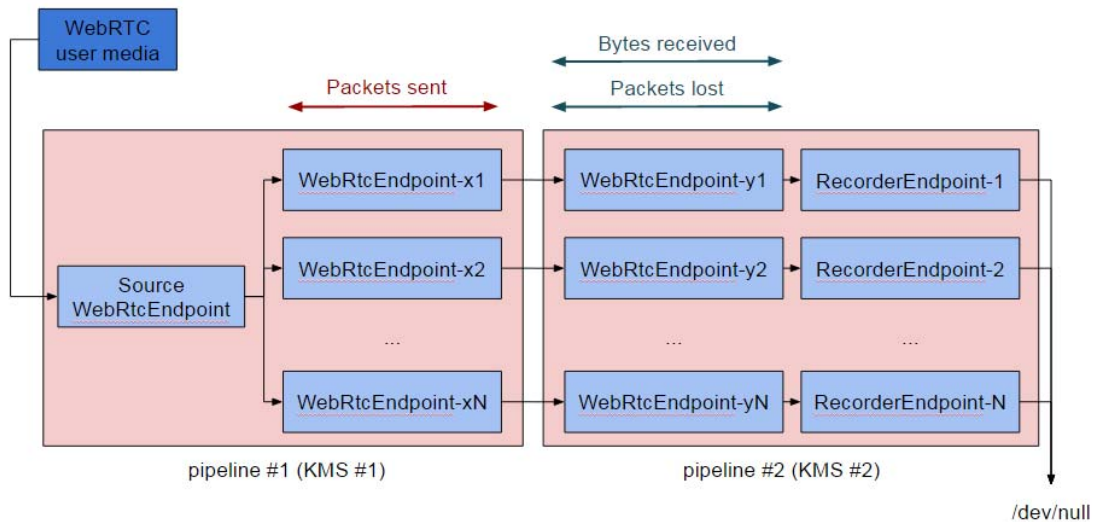


Figure 21 Media topology of the nubomedia-network-benchmark application.

The aim of this kind of topology is to create a high load of media traffic between both instances of KMSs. In addition, this value of that bandwidth can be tuned simply changing the number of WebRtcEndpoints (N) and also the bandwidth of each individual media element. The total bandwidth (in Mbps) between the two media servers will be the aggregation of the data transmitted between each pair of connected WebRtcEndpoints (i.e. WebRtcEndpoint-x1 → WebRtcEndpoint-y1, WebRtcEndpoint-x2 → WebRtcEndpoint-y2, and so on).

The application gathers a set of WebRTC statistics including packets sent, packet lost, and bytes received (see diagram before), which are written by the application as a CSV file. A CSV file is a comma separated values file, and it can be opened using a spreadsheet processor, for instance Microsoft Excel or LibreOffice Calc. The gathering rate can be configured using the application GUI (see picture below).

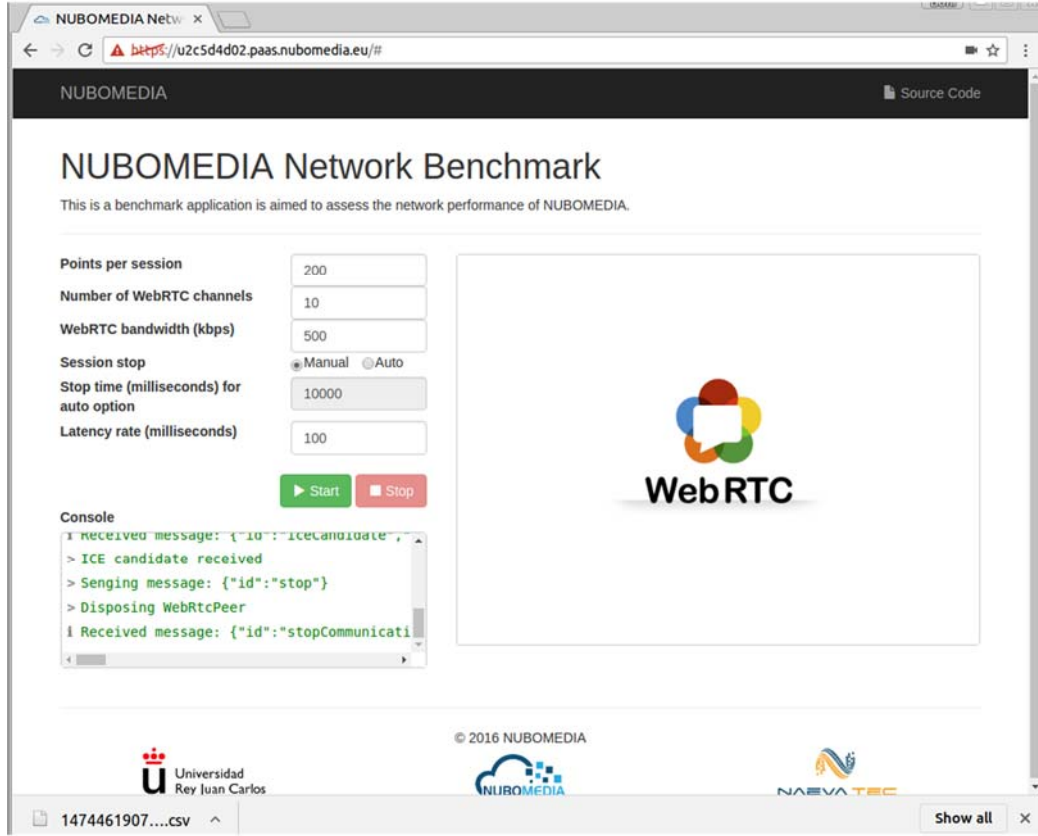


Figure 22 GUI of the nubomedia-network-benchmark application

In order to calculate the value for metric 4.1, this application will be deployed twice in the NUBOMEDIA PaaS. On the one hand, the application will be deployed using an SLA of **bronze**, and in the other hand we will select the best SLA, i.e. **gold**. For each deployment, the probability of losing packets is calculated. For each pair $[WebRtcEndpoint_{xi}, WebRtcEndpoint_{yi}]$, this figure is calculated as follows:

$$PacketLostProb_i = 1 - \frac{PacketLostWebRtcEndpoint_{yi}}{PacketSentWebRtcEndpoint_{xi}}$$

Therefore, the total packet lost probability is the aggregation of each individual contribution. This is calculated as follows:

$$PacketLostProb = 1 - \frac{\sum_{i=1}^N PacketLostWebRtcEndpoint_{yi}}{\sum_{i=1}^N PacketSentWebRtcEndpoint_{xi}}$$

This value is calculated both for bronze and for gold deployment. The final value of the metric 4.1 is calculated as the average of each measurement as follows:

$$Metric4.1 = \frac{Average(PacketLostProb_{Gold})}{Average(PacketLostProb_{Bronze})}$$

As defined in NUBOMEDIA DoW, the success criteria for this metric is the following:

- Metric is fully successful: ratio is 0.8 or lower
- Metric is reasonably successful: ratio is 0.9 or lower
- Metric is unsuccessful: ratio over 0.9

7.5.2 Experimentation and Results

As defined before, the application has been deployed using the bronze first, and then in gold SLA. In order to validate the experiment, the value of the total bandwidth between media servers must be controlled. The evolution of this indicator is depicted in the following charts:

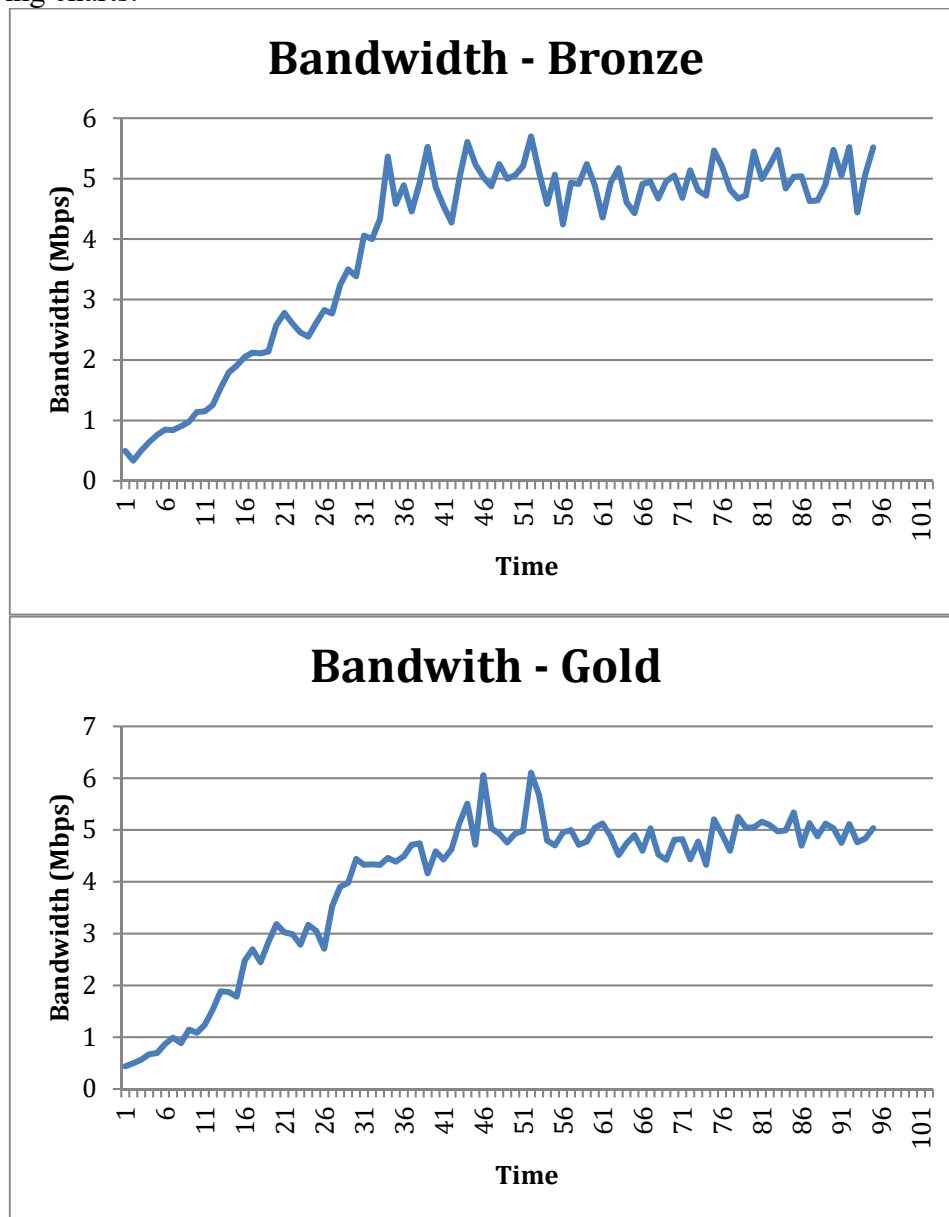


Figure 23 Total bandwidth (in Mbps) in the Bronze and Gold deployments

As can be seen, both bandwidth trends are similar. Both for bronze (3.946 Mbps average; 1.575 Mbps standard deviation) and Gold (3.996 Mbps average; 1.474 Mbps standard deviation) there is almost the same media traffic between the media servers.

Due to the fact that the traffic is equivalent in both experiments, we are able to compare the packet lost probability. The observed difference in this ratio is going to be motivated by the underlying network, which is directly linked to one or other SLA (bronze or gold). The evolution of the packet lost probability during the time of the experiment is shown in the following charts:

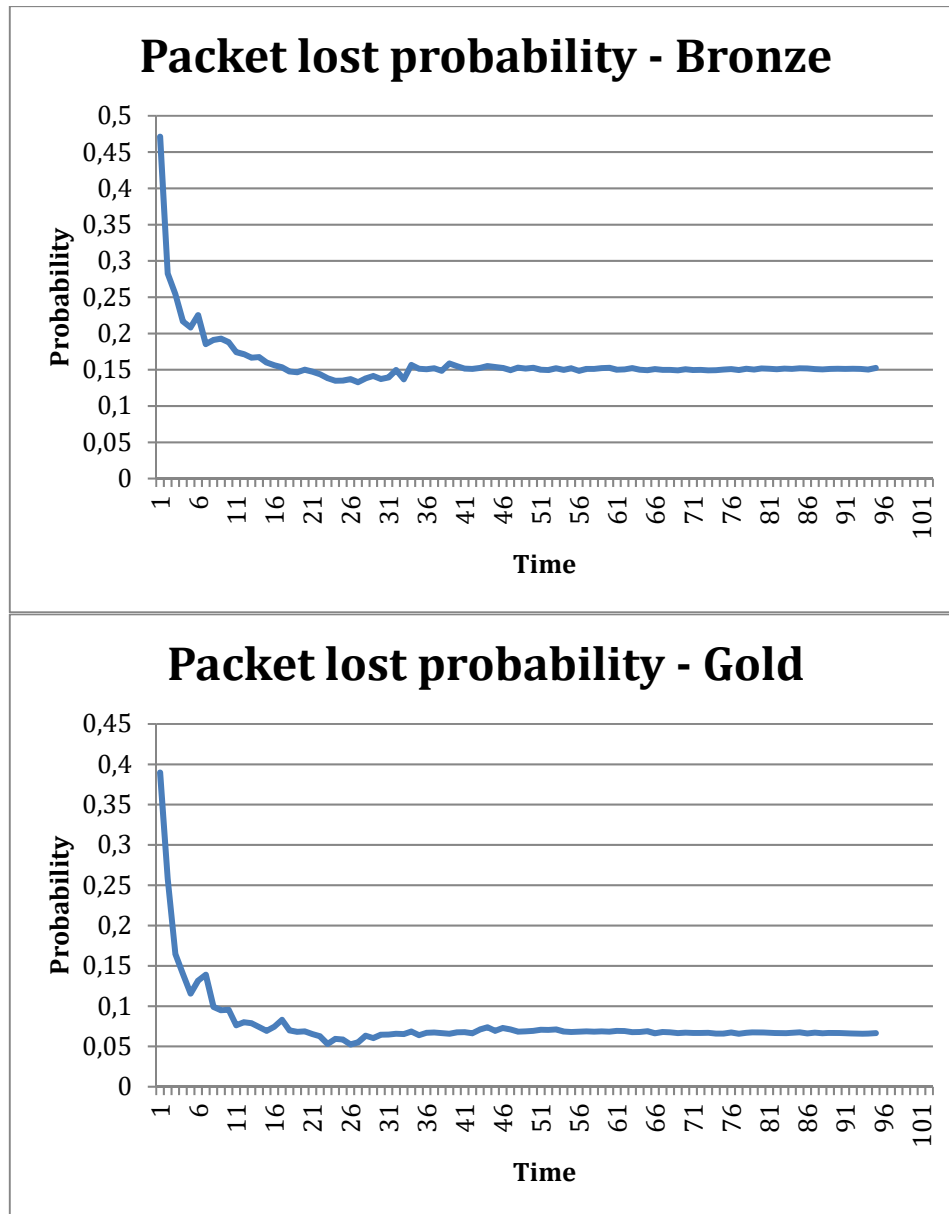


Figure 24 Packet lost probability in the Bronze and Gold deployments

Both charts show the same behavior on the evolution of this indicator in time. Nevertheless, looking closer into the average values we can find slight but significant differences:

$$Average(PacketLostProb_{Bronze}) = 0.160$$

$$Average(PacketLostProb_{Gold}) = 0.077$$

As we can see, the average probability of losing packages in bronze is higher than the probability of losing package in gold. This makes sense, due to the fact that the underlying network capabilities in gold SLA is better (maximum bandwidth 100 Mbps) than in bronze (maximum bandwidth of 5 Mbps). All in all, the final value for metric 4.1 is the following:

$$Metric4.1 = \frac{0.077}{0.160} = \mathbf{0.484}$$

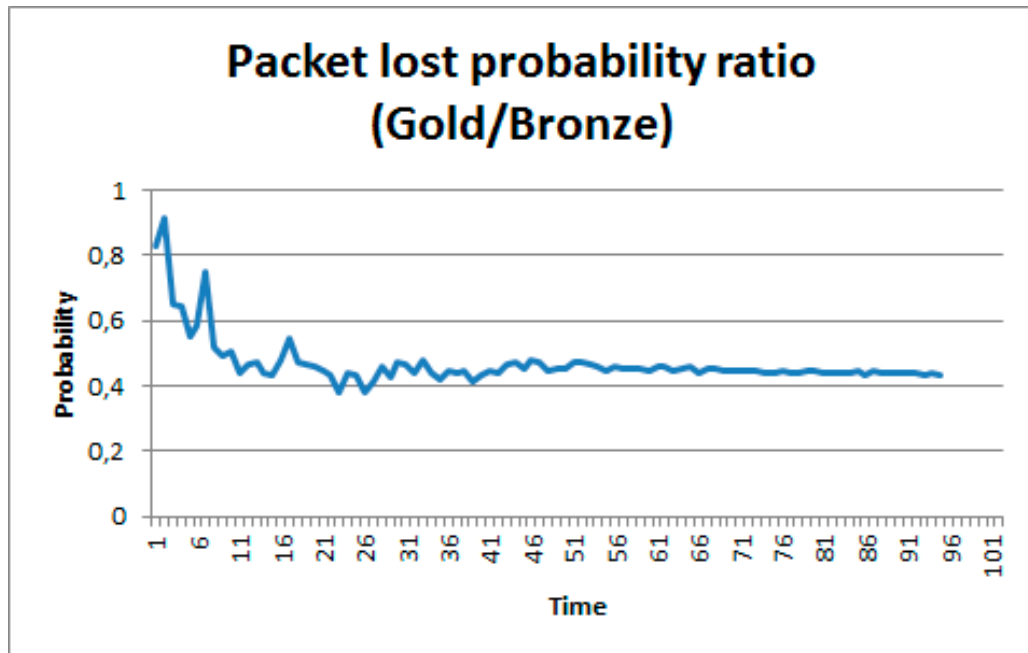


Figure 25 Packet lost probability ratio between Gold and Bronze

All in all, we can conclude that metric is fully successful since this ratio is lower than 0.8.

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/1tuHYtd6srWZSl2LMqALiY7jucvmordwtsLqE_rT54HI

[AUTONOMOUS INSTALLER LOG] <https://drive.google.com/file/d/0B6xn-f43p84tQVNVTEg2UW56OWs/view>