KASKADA PLATFORM IN CLOUD ENVIRONMENT

HENRYK KRAWCZYK, JERZY PROFICZ
AND BARTŁOMIEJ DACA

Academic Computer Centre in Gdansk
Gdansk University of Technology
Narutowicza 11/12, 80-233 Gdansk, Poland

(received: 7 May 2015; revised: 15 June 2015;
accepted: 23 Juny 2015; published online: 1 October 2015)

Abstract: KASKADA is a computing platform for developing and running services and applications oriented to multimedia processing and data streaming. A solution for serving the platform in the PaaS model is presented. The paper briefly describes the software for creating the private cloud solutions and the components designed and implemented to enable hosting the platform in the cloud environment based on the OpenStack software. The cloud has been deployed to a supercomputer working in C^{2}NIWA at the Gdansk University of Technology. This article assumes the reader's basic knowledge of the KASKADA platform.

Keywords: KASKADA, cloud computing, PaaS, CD NIWA, OpenStack

1. Available software for the computing cloud

In order to enable the KASKADA Platform [1, 2] to be served in the PaaS (Platform-as-a-Service) model [3], the OpenStack software has been used. OpenStack is currently one of the most popular and most rapidly developing environments for creating both public and private clouds. The OpenStack project is supported by such companies as IBM, Intel and HP [4]. There are other open-source products on the market, for instance OpenNebula, Eucalyptus and CloudStack. All of them have many common features, but what distinguishes the OpenStack from the other products is the development speed and the community size. The four aforementioned projects and their monthly contributors count are presented in Figure 1

The Juno version of the OpenStack software has been used for the purposes of this project. As at the moment of writing this article, Juno is the latest version of the software.

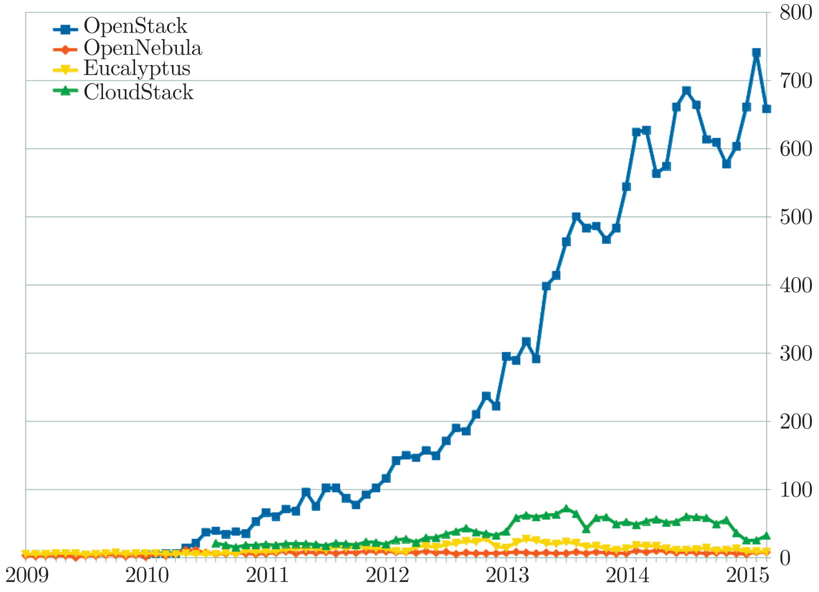


Figure 1. Contributors of the most popular IaaS open source platforms; source: [5]

1.1. The environment configuration

The whole project was carried out on the 16 servers of the Galera+ computing cluster, all of which used the Ubuntu 14.04.1 operating system. The OpenStack software was installed on 14 servers. Additionally, one of the servers – the access server – functioned as a gateway through which the connections (ssh, rtsp, http) were redirected from the global network to the cluster local network. The last server out of the aforementioned 16 was a storage server. Figure 2 presents a server dependency diagram.

3 out of 14 servers on which the OpenStack software was deployed, were used as management nodes:

- *virtual network node* – responsible for the network traffic management within the cloud;
- *storage provider* – responsible for hosting virtual discs to virtual machines;
- *the controller* – managing other services.

Such a division is recommended by the OpenStack developers [6]. On the other 11 nodes (known as the compute nodes), virtual machines are run by means of the KVM hypervisor.

1.2. OpenStack software components

The OpenStack software is divided into particular services. Below, short descriptions of the most important components are provided. Furthermore, Figure 3 depicts a diagram of communication between the components.

The keystone's function is providing the authentication and authorization of users. Moreover, it manages the catalogue of services in which the information

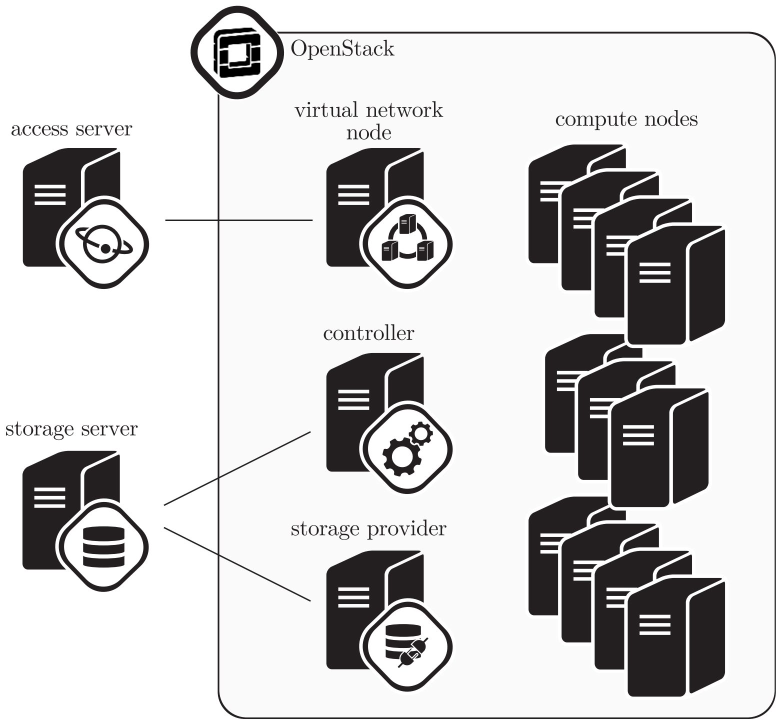


Figure 2. The environment configuration

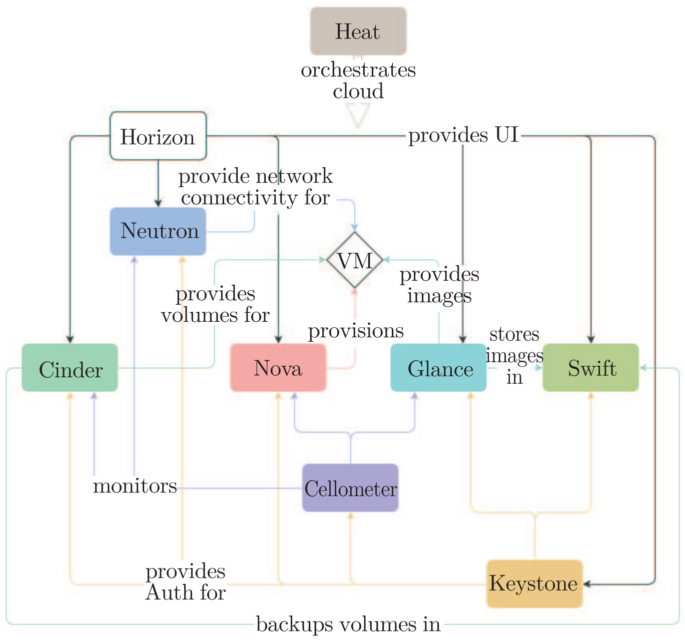


Figure 3. The OpenStack software architecture; source: [6]

concerning the addresses of services is stored. The Glance component stores and manages the virtual machine images. It provides the images to the Nova service. The image is a collection of files which are used to create or rebuild the virtual machine. Images can be stored in the Swift component. Its key function is data storage. The storage is highly fault tolerant and the data is stored in a distributed way. The component is responsible for, *i.a.*, replication of the stored data. This component was not used in our project; it was replaced with the *GlusterFS* distributed file system. The main aim of the Nova component is the management of life cycles of the virtual machines in a computing cloud environment. Nova is also responsible for appropriate distribution of the virtual machines on the servers. The Neutron service manages the virtual networks, routing, ports forwarding, *etc.*, and provides network connectivity for virtual machines. Cinder allows managing disc resources. Furthermore, it enables creating and deleting virtual discs and assigning access to these discs to virtual machines. The Ceilometer component collects the cloud usage statistics. The Horizon component provides a user interface for easy management of all the services. It enables, *i.a.*, virtual machines, networks and storage management, as well as assigning public IPs to virtual machines and checking the cloud usage statistics. The Heat component enables orchestration. In order to do so, it requires an orchestration script in either the HOT format (native for the component) or in the AWS CloudFormation format (developed by Amazon).

2. Extra components implemented

For the purposes of the project, it was necessary to implement a few additional components, as well as to modify the KASKADA platform in a certain way. In this chapter, particular elements and modifications are described.

2.1. KASKADA platform instances database

Figure 4 presents a diagram of the database which stores information on the KASKADA platforms deployed in the cloud. The database consists of three tables:

- *NETWORK* – The table contains a list of virtual networks created for the needs of the KASKADA platform;
- *PLATFORM* – The table stores general information on the deployed platform instance: the platform's name, owner, IP address and port of virtual access server enabling SSH access to the virtual machines and IDs of two virtual networks used by a given instance – the Ethernet network and the InfiniBand network;
- *CONSOLE_VM* – Contains information on KASKADA platform user consoles: addresses of the platforms and the ID of the platform with which a given console is connected.

On the basis of the information stored in the database, virtual networks for the deployed KASKADA platforms are assigned, along with proper redirection of

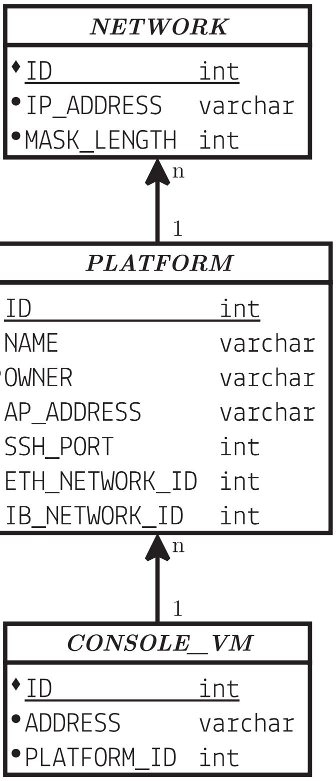


Figure 4. Database diagram

the network traffic to the user's interface of these platforms. In this solution the 5.5 version of MySQL server was used.

2.2. Adaptation of the user console

Cloud environment can be divided into three major categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [7]. This classification is presented in Figure 5. IaaS provides physical or virtual machines, storage, load balancers, images, *etc.* In this project OpenStack software deployed on the Galera+ supercomputer is responsible for the realization of these features. PaaS provides a platform that can be used for developing and/or running client software. The KASKADA platform belongs to this category and hosting it in this model is the main goal of this project. The last category – SaaS – provides access to the software, for example games, office suites or applications created on the KASKADA platform. Accessing any of these categories is possible from devices connected to the Internet. In most cases, access can be realized by applications such as terminal or web browser, but sometimes the client has to install some dedicated software.

The user console of the KASKADA platform allows users, *i.a.*, to create and start new services and to monitor the resource usage of the started services. For

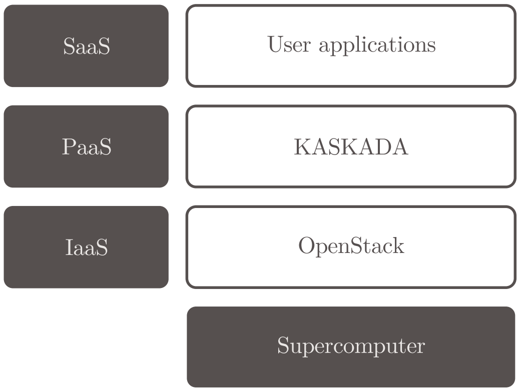
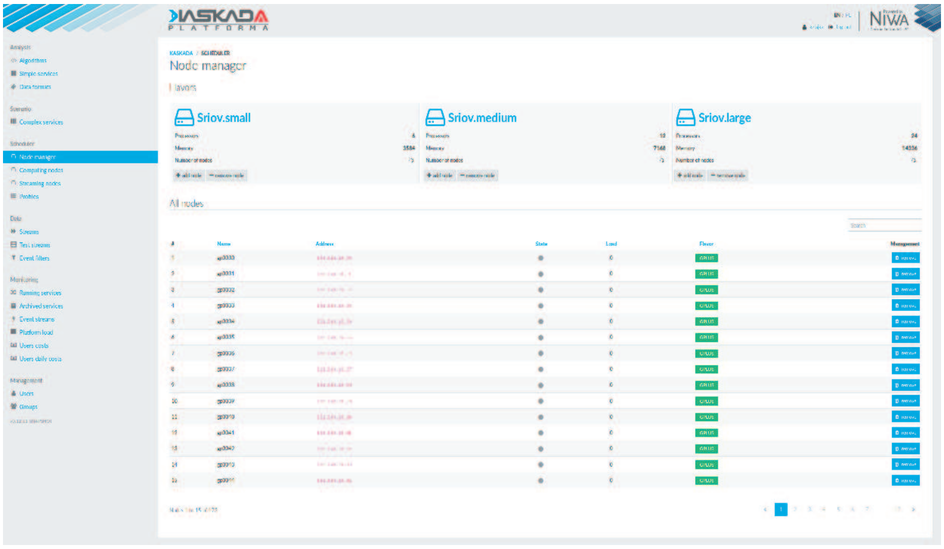


Figure 5. Cloud environment layers

administrators, the user console allows them to monitor, *i.a.*, the resource usage of the whole cluster, the currently running services or costs generated by certain users. However, not all of the administrators' tasks could be performed using the user console, thus, certain adaptations of it were introduced:

- User management from the interface (available for the administrator only) was implemented – hitherto, administrators used the LDAP server;
- Machine management was introduced – it is now possible to run a new machine or to turn off the ones that are no longer needed. Additionally, the size of the starting machine can be chosen (predefined configuration of the CPU number and RAM size). The new computing nodes management panel is shown in Figure 6.



Node manager

Flavors

Sriov.small

Processors	6
Memory	3584
Number of nodes	[illegible]

+ add node − remove node

Sriov.medium

Processors	12
Memory	7168
Number of nodes	[illegible]

+ add node − remove node

Sriov.large

Processors	24
Memory	14336
Number of nodes	[illegible]

+ add node − remove node

All nodes

#	Name	Address	State	Load	Flavor	Management
1	ap0330	[illegible]	●	0	GPLUS	remove
2	ap0331	[illegible]	●	0	GPLUS	remove
3	gp0332	[illegible]	●	0	GPLUS	remove
4	gp0333	[illegible]	●	0	GPLUS	remove
5	ap0334	[illegible]	●	0	GPLUS	remove
6	ap0335	[illegible]	●	0	GPLUS	remove
7	gp0336	[illegible]	●	0	GPLUS	remove
8	gp0337	[illegible]	●	0	GPLUS	remove
9	ap0338	[illegible]	●	0	GPLUS	remove
10	gp0339	[illegible]	●	0	GPLUS	remove
11	gp0340	[illegible]	●	0	GPLUS	remove
12	ap0341	[illegible]	●	0	GPLUS	remove
13	ap0342	[illegible]	●	0	GPLUS	remove
14	gp0343	[illegible]	●	0	GPLUS	remove
15	gp03[illegible]	[illegible]	●	0	GPLUS	remove

Nodes 1 to 15 of 173

Figure 6. New computing node management panel

2.3. New artifacts

In order to use the OpenStack software to deploy the KASKADA platform, it was necessary to create appropriate virtual machine images, as well as an orchestration script which describes the platform's deployment scenario.

Virtual machine images had to be created for the KASKADA platform to be deployed in cloud:

- *kaskada-db* – this image contains preliminarily configured servers: LDAP and the database. LDAP is employed by the rest of the machines (Linux user authentication) and by the user's console. The console uses the database as well;
- *kaskada-ev* – an image of the machine with the ActiveMQ server installed;
- *kaskada-uc* – a server on which the user's console is deployed;
- *kaskada-cn* – machines deployed from this image constitute virtual computing and streaming nodes, on which the KASKADA platform runs its tasks;
- *kaskada-ap* – the access server's image. Users of a given platform enter this server using ssh. Furthermore, it serves for compilation of user algorithms.

The images are managed by the Glance component. Glance hosts the images to Nova – the component which deploys virtual machines.

The orchestration script is executed by the Heat component of the OpenStack software. It describes the resources that create the infrastructure for a cloud application. It can be written in the JSON or YAML format, thus it is readable for standard users who do not know any programming language.

The script is implemented in the HOT format. It contains a definition of input parameters. Providing these parameters is necessary to launch the procedure of deploying the KASKADA platform in cloud. The parameters required from the user are:

- the platform's unique name;
- the name and surname of the person deploying the platform;
- the login and password of the administrator of the newly created platform.

Moreover, the OpenStack software administrator's password has to be provided in order to start the procedure of deploying the platform.

The most vital elements of the HOT format employed in the script [8] are shortly described below:

- `OS::Nova::Server` – server specification. One of the parameters of this object is a script which is activated immediately after the machine's launch. As a result, every server can be additionally configured as it is launched. Such a mechanism is extensively employed during the deployment of the KASKADA platform in cloud. It is used, *i.a.*, in order to add the platform's administrator (defined in the input parameters) to the LDAP server;
- `AWS::AutoScaling::AutoScalingGroup` – server group definition. In the definition of this element, it is crucial to provide a template of the server which will be deployed in the group. With this component, the number of servers deployed within the

group can be easily modified. The mechanism is used in the KASKADA platform to manage the number of deployed virtual computing nodes;

- *AWS::CloudFormation::WaitCondition* – a tool designed to enforce the machines deployment order. When a given machine finishes its initialization, it reports the task's realization to this tool and, as a result, a deployment of the next machine is launched;
- elements connected with virtual internet networks:
  - *OS::Neutron::Net/OS::Neutron::Subnet* – objects representing a virtual network or subnetwork;
  - *OS::Neutron::Router* – a virtual router which enables connecting two virtual networks;
  - *OS::Neutron::FloatingIP* – an object representing the machine's external IP address. However, it is not an external address which can be accessed from the internet network. What is meant by "external IP address" here is an address which is accessible from the outside of the cloud (*e.g.* from the computing nodes of the Galera+ cluster).

2.4. Supporting servers

In addition to the modifications of the KASKADA platform and the artifacts for the OpenStack software, it was necessary to implement several servers supporting the cooperation between the platform and OpenStack. In this section, the servers are briefly described.

The proxy server for the RTSP protocol was implemented in JavaScript. Its function is to forward RTSP streams from the KASKADA platform to the internet network. The server establishes two connections: one with a client who wants to receive the RTSP stream, and the other with the server which exposes a given stream. Afterwards, the server forwards the stream from the server to the client. The information required by this server is provided by the following URL address:

rtsp://proxy:port/server_IP/server_port/hash

The elements of the address refer to:

- *proxy* and *port* – the address/domain name of the machine on which the RTSP proxy server has been deployed and the port on which the server listens for connections;
- *server_IP* and *server_port* – the IP address and port of the internal server which exposes the RTSP stream;
- *hash* – the selector of the stream exposed by the internal server.

The StorageHelper server was implemented to enable dynamic adding and removing remote disc resources inside the distributed file system *GlusterFS*. In spite of the fact that the OpenStack software has tools to manage remote disc resources, the KASKADA platform's characteristics do not allow the tools to be used. Unfortunately, the latest version of the OpenStack software does not support assigning the same remote disc resource to numerous virtual machines at the same time.

The server provides two methods:

- `/allocate/{storage_name}` – allocating 100 GB of the storage, identified by the name `storage_name`;
- `/remove/{storage_name}` – removing the previously allocated storage.

The Apache server is employed to forward the HTTP traffic to the particular KASKADA platforms in cloud. The forwarding is conducted on the basis of the domain name. All the platforms are accessible at the following address: `<platform_name>.cloud.kaskada.task.gda.pl`. The DNS servers forward the traffic connected with the `cloud.kaskada.task.gda.pl` domain to the access server. Next, the Apache server, depending on the `<platform_name>`, forwards the traffic to an appropriate virtual machine on which the platform user's interface has been deployed.

A set of macros in the perl programming language was implemented in order to automate adding new forwards. Each time a configuration of the Apache server is loaded, the macros connect to the database, download the information on the platforms and addresses of the deployed user's interfaces, and then they register the forwards. To enforce the Apache server to reload the configuration, it is necessary to send a `service httpd reload` command.

It was vital to support the cooperation between the KASKADA platform and the OpenStack software, as well as to implement the mechanisms allowing us to circumvent the problem of one external address for the whole computing cloud. These were the reasons for the CloudHelper server to be created. The server provides REST API, the particular methods of which are described below:

- platform management:
  - `GET /platforms/<max_size>` – returns the orchestration script describing the process of deploying the KASKADA platform. In the `max_size` argument, a maximum expected number of virtual machines in the platform is provided. The argument has been introduced to improve the choice of the virtual machine's mask. 3 sizes are supported: 254, 510 and 1022. The `max_size` parameter does not have to be equal to one of the sizes; the server will choose to value closest to the required;
  - `POST /platforms` – this method expects the definition of the platform in the JSON format as a parameter. The definition has to contain information on the name of the deployed platform, its owner and the address of the virtual access node. The information is stored in the database. Moreover, a storage used by the KASKADA platform is allocated. Subsequently, on the node functioning as a gateway, the iptables tool register the redirections of ports, which enables the access of SSH to the access node of the deployed platform;
  - `DELETE /platforms/<platform_name>` – this method results in deleting the `platform_name` platform and all of its elements: virtual machines, storage, virtual networks, redirections/forwardings, *etc.*

- managing a given instance in the cloud:
  - `GET /platforms/<platform_name>/flavors` – the method returns the definitions of the virtual computing nodes accessible for the `platform_name` platform;
  - `POST /platforms/<platform_name>/nodes` – the function of this method is to deploy new virtual computing nodes for the KASKADA platform identified by `platform_name`. The number and type of the new virtual computing nodes is provided in the JSON format, in an enclosed message;
  - `DELETE /platforms/<platform_name>/nodes` – the method evokes turning off and deleting some virtual computing nodes. The list of the chosen nodes should be included in the message in the JSON format;
  - `POST /platforms/<platform_name>/uc` – a method which records the information on the deployed user's interface, stores it in the database and relates it to the `platform_name` platform. Then, it reports the need for reloading the configuration to the Apache server on the node functioning as a gateway.
- file management:
  - `GET /resources/<file>` – due to this method, a server sends a `file` file to the client, provided that the file belongs to the set of the shared files. The method is used for passing the scripts to the virtual machines during the deployment of the KASKADA platform.

3. Architecture and modus operandi

The first part of this section presents the KASKADA platform architecture in the PaaS model. It describes the role of each server and shows which components communicate with each other. The next part describes in detail how the platform is started/stopped and how the administrator can change the number of the computing nodes.

3.1. Cloud architecture

The components described in section 1 have been deployed for servers intended for the realization of this project in a way presented in Figure 7. To ensure a clear view, the OpenStack software components and the connections between the particular components of the KASKADA platform are excluded from Figure 7. Below, a list of servers, components implemented on the servers and the connections between these elements is provided:

- *storage server* – on this machine, the StorageHelper server has been deployed;
- *gateway server* – the server on which the proxy server has been installed for the RTSP protocol, along with the Apache server which forwards the HTTP traffic. The connections are as follows:
  - The RTSP proxy server connects to the stream sources deployed on the *kaskada_cn* virtual machines of the KASKADA platform in cloud;
  - The Apache server, using a set of macros, obtains the information on the HTTP forwardings from the database and, subsequently, realizes the forwardings;

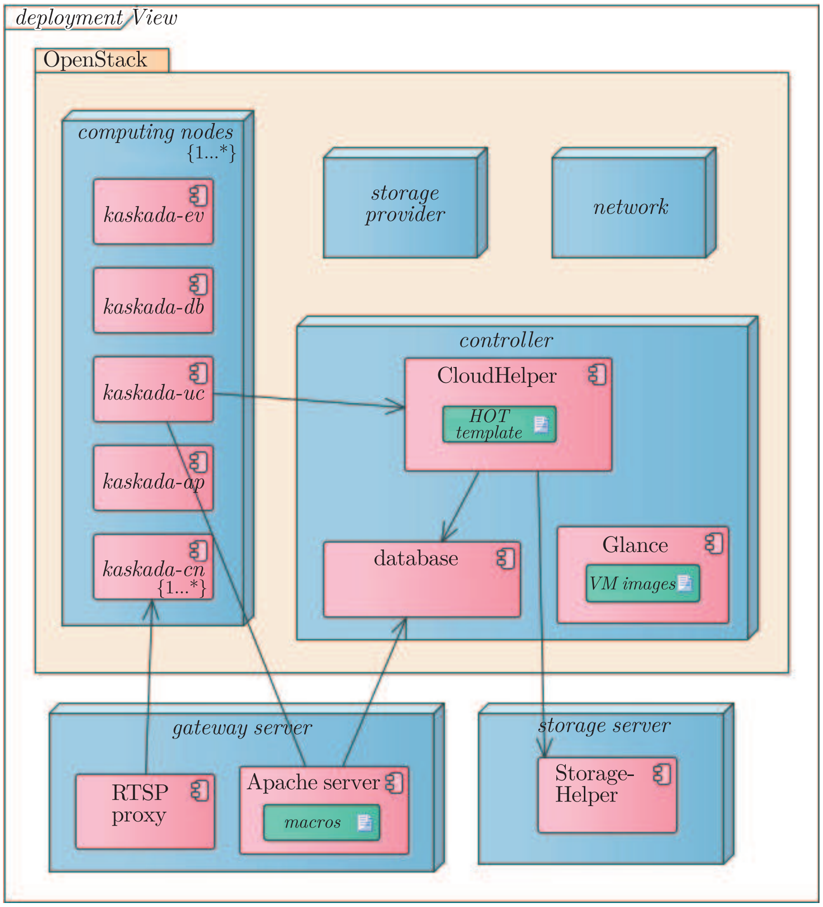


Figure 7. Deployment diagram of cloud architecture

- the OpenStack servers:
  - *controller* – on this machine, apart from the OpenStack services, the CloudHelper server and the database server have been deployed. The connections are:
    The CloudHelper server uses the StorageHelper component (implemented on the storage server) to manage the storage for the KASKADA platform. Furthermore, it stores the information on the deployed platforms in the database;
  - the *computing nodes* – by means of the OpenStack software, the virtual machines belonging to the KASKADA platform are deployed on these nodes. The images of the machines are managed by the Glance component;
  - *storage provider* and *network servers* – servers on which the OpenStack software services have been implemented. The services are indispensable for the proper functioning of the whole environment.

The OpenStack software architecture is extensively described in [9].

3.2. Modus operandi of the KASKADA platform in cloud environment

Launching the KASKADA platform in the PaaS model starts with downloading the orchestration script from the CloudHelper server. After receiving the script's input parameters from the user (the platform's name, the administrator's name and password), the OpenStack software creates a separated virtual network for the platform, and then it assigns the addresses to the particular machines which will be created in the next stages of the process. Afterwards, the OpenStack software creates and deploys the first virtual machine (*kaskada-db*), passing it the parameters provided by the user and the IP address reserved for the *kaskada-ap* machine – the platform's access server. Figure 8 depicts consecutive stages of the platform's deployment process.

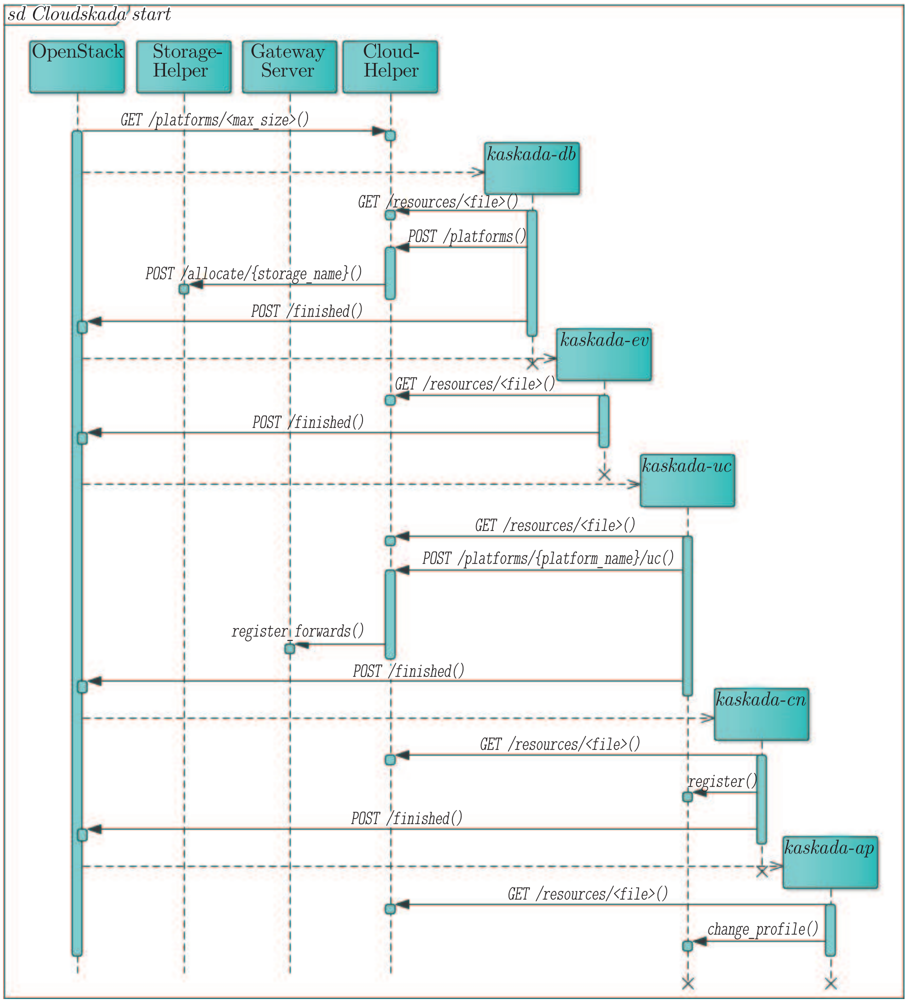


Figure 8. Deploying KASKADA platform instances

The *kaskada-db* server initializes its activity with downloading bash scripts from the CloudHelper server (`GET /resources/<file>`). The scripts prepare the database and the LDAP server for the KASKADA platform. Moreover, they register the platform on the CloudHelper server (`POST /platforms`), passing the information on the platform to the server. The information includes the platform's name, its owner and IP address of the access server. The CloudHelper server records the obtained information in the database and instructs the StorageHelper server to allocate shared storage for the platform. After the CloudHelper and Storage-Helper servers finish their activity, the *kaskada-db* machine informs the OpenStack software of the end of the initialization process.

Another virtual machine deployed by the OpenStack software is *kaskada-ev* with an ActiveMQ server implemented on it. After the start, the server downloads bash scripts (`GET /resources/<file>`) which then mount shared the storage dedicated for the platform. When the scripts execution is completed, a method notifying the OpenStack software of the end of the *kaskada-ev* server deployment is activated.

The third server created is *kaskada-uc*. On this server, the user console will be started. In the first step bash scripts from CloudHelper server are downloaded (`GET /resources/<file>`). This scripts are used to connect remote storage to the server and to post-configure the Tomcat server (*i.a.* set the database and the LDAP user and the password, set the name of the platform *etc.*). Next, the user console is registered in the CloudHelper server (`POST /platforms/<platform_name>/uc`). The CloudHelper server stores the IP address of the user console machine in database and delegate registering HTTP traffic forward and SSH tunneling to the GatewayServer. From this moment, the user console can be accessed by users via the HTTP address `<cloud_name>.cloud.kaskada.task.gda.pl`. When the *kaskada-uc* server is configured and registered, it informs the OpenStack services that the next servers can now be started.

On the next step the OpenStack software starts two machines: virtual computing node (*kaskada-cn*) and virtual streaming node (*kaskada-sn*). After downloading scripts from the CloudHelper server (`GET /resources/<file>`), the remote storage is mounted, servers connect themselves to the InfiniBand network and each server registers itself in the user console. The *kaskada-sn* server launching process ends with informing the OpenStack software about the end of the initialization.

Kaskada-ap is the last server that is launched by the OpenStack software. As all the servers, it starts with downloading the bash scripts from the CloudHelper server (`GET /resources/<file>`). The scripts are responsible for mounting the shared storage. The last step of the initialization process is changing the active profile in the KASKADA platform, which enables the starting of services on the platform.

The adding/deleting of the nodes to the KASKADA platform is shown in Figure 9.

In order to add or remove nodes, the administrator of the KASKADA platform must choose a proper option from the user console – Node Manager. In the first step, the user console gets the available server sizes – predefined sets

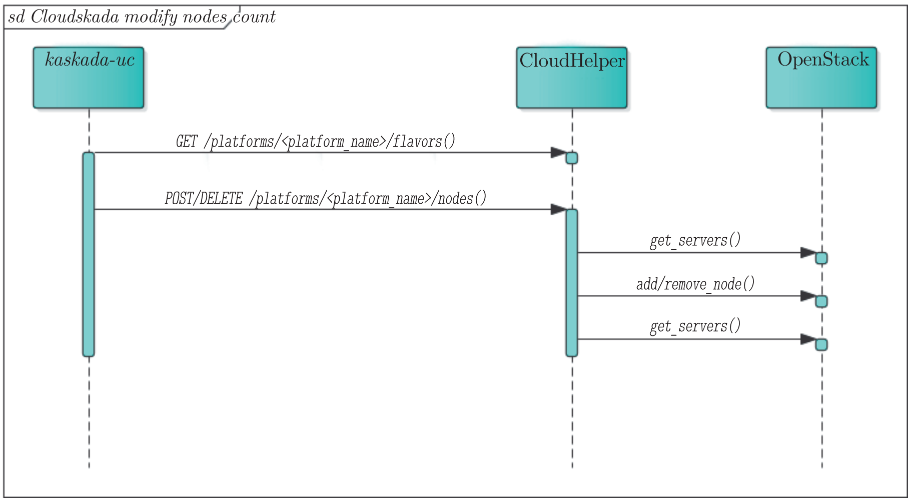

sd Cloudskada modify nodes count
kaskada-uc
CloudHelper
OpenStack
GET /platforms/<platform_name>/flavors()
POST/DELETE /platforms/<platform_name>/nodes()
get_servers()
add/remove_node()
get_servers()

Figure 9. Nodes count modification

of memory sizes and the core count – from the CloudHelper server (*GET /platforms/<platform_name>/flavors*). Then, the administrator chooses proper server size and specifies modifier of nodes count. The modifier should be less than 0 if nodes should be deleted, or greater than 0 if nodes should be added. The user console component sends a proper message to the CloudHelper server: *POST /platforms/<platform_name>/nodes* and a message in the JSON format containing the selected size and modifier when adding nodes, or *DELETE /platforms/<platform_name>/nodes* with list of chosen servers to delete. List of servers is designated automatically by user console.

When deleting nodes, the only thing that the CloudHelper must do is to forward the list of the servers to delete to the OpenStack software. When adding new nodes, CloudHelper server starts with getting the list of already started servers of the chosen size (*get_servers()*). Then it commissions the OpenStack software to start the requested number of servers. When the OpenStack software ends its work, the CloudHelper server gets the list of all started servers once again, compares the received list with the one received at the beginning of this process and determines the list of newly started servers. Then, it tests the state of all the newly created servers. If all servers have started successfully (state *RUNNING*), it returns the 200 code and the list of the newly created servers to the KASKADA user console. If some of them have a *FAILED* state, it returns the 206 status and the list of the servers which have started successfully. When no server has started properly, it returns the code 400 and an empty list.

After adding/deletion of the nodes, the user console of the KASKADA platform adds/removes servers to/from the list of the computing servers. After that, the process of the node count modification ends.

The last process described in this section is platform removing. After the KASKADA platform removal request (*DELETE /platforms/<platform_name>*) the Cloud-

Helper server orders the GatewayServer to deregister HTTP forwarding and SSH tunneling connected to the requested platform (*deregister_forwards()*). Then it asks the OpenStack software to remove all the elements (servers, virtual networks, *etc.*) from the KASKADA platform. After that, the CloudHelper server commissions the StorageHelper server to remove the shared storage and all the data used by this platform's users. At the end it removes all the information connected with this platform from the database and the procedure ends. This process is presented in Figure 10.

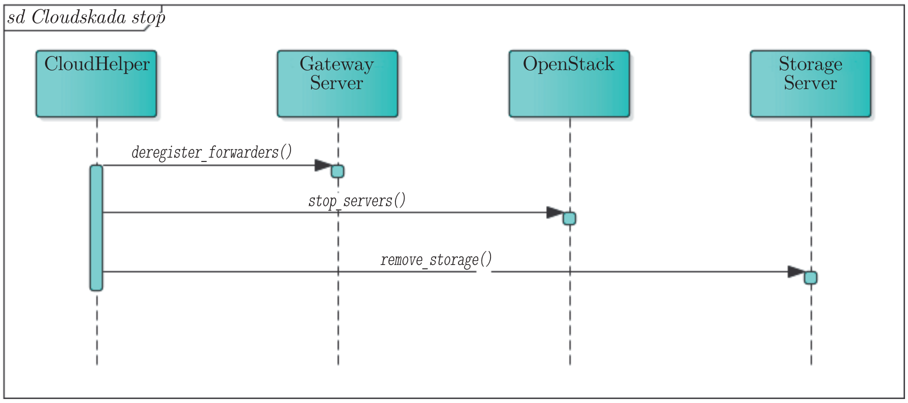


Figure 10. Removing the platform

4. Further works

The KASKADA platform in the PaaS model was fully tested by a dedicated team of testers. They executed over 500 use cases to ensure that the platform was fully functional. Presently it still lacks a few features, *i.a.*:

- automatic scaling of the compute node count based on the actual load of servers;
- adding additional user console servers and load balancing;
- group management panel in the user console;
- automatic compiling and deploying of the user's algorithms directly from the SVN/GIT repository, with no need for manual copying of the source code and compiling it on virtual nodes.

Adding these features will ease the administration of the KASKADA platform in cloud environment. The *automatic compiling and deploying* feature will allow removing the SSH access to the virtual machines. Moreover, this solution has been deployed to the Galera+ cluster and in short time it should be moved to the new Tryton supercomputer [10].

References

[1] Krawczyk H and Proficz J 2010 *KASKADA – Multimedia processing platform architecture*, Proceedings of the 2010 International Conference on Signal Processing and Multimedia Applications (SIGMAP) 26

[2] Krawczyk H and Proficz J 2012 *Real-Time Multimedia Stream Data Processing in a Supercomputer Environment*, Interactive Multimedia, Ioannis Deliyannis (Ed.), InTech.

[3] Chang W Y, Abu-Amara H and Sanford J F 2010 *Transforming Enterprise Cloud Services*, Springer 55

[4] Companies supporting The OpenStack Foundation , http://www.openstack.org/foundation/companies/ (accessed on 08/07/2015)

[5] Qingye J 2015 *CY15-Q1 Community Analysis – OpenStack vs OpenNebula vs Eucalyptus vs CloudStack*, http://www.qyjohn.net/?p=3801

[6] Openstack architecture , http://docs.openstack.org/juno/install-guide/install/apt/content/ch_overview.html (accessed on 08/07/2015)

[7] Buyya R, Broberg J and Goscinski A 2011 *Cloud Computing: Principles and Paradigms*, Wiley Press 1

[8] OpenStack Heat Resource Types , http://docs.openstack.org/developer/heat/template_guide/openstack.html (accessed on 08/07/2015)

[9] Fifield T, Fleming D, Gentle A, Hochstein L, Proulx J, Toews E and Topjian J 2014 *OpenStack Operations Guide*, OpenStack Foundation, O'Reilly Media

[10] Krawczyk H, Nykiel M and Proficz J *Tryton supercomputer capabilities for analysis of massive data streams*, Polish Maritime Research (to be published)