JOINT INSTITUTE FOR NUCLEAR RESEARCH
Laboratory of Information Technologies

FINAL REPORT ON THE SUMMER STUDENT PROGRAM

Development of a storage and visualization system for usage statistics regarding JINR's cloud resources

Supervisor:
Nikolay Kutovskiy

Student:
Ruslan Gainanov, Russia
Saint Petersburg National Research University of Information Technologies, Mechanics and Optics

Participation period:
July 03 – August 27

Dubna, 2016

Contents

Abstract

Cloud technologies are already wide spread among IT industry and start to gain popularity in academic field. There are several fundamental cloud models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). The document describes the system of obtaining and visualization statistics from cloud infrastructure deployed at the Laboratory of Information Technologies of the Joint Institute for Nuclear Research (LIT JINR).

Description of current state

The Joint Institute for Nuclear Research has a cloud infrastructure. It is realized by the fellows of the Laboratory of Information Technologies (LIT). Application of cloud technologies is suitable for the solving of various classes of research tasks. The JINR cloud is based on IaaS model [1]. In that model, a third-party provider (in the JINR hierarchy it is datacenter of LIT) hosts hardware, software, servers, storage and other infrastructure components on behalf of its users. IaaS providers also host user's applications and handle tasks including system maintenance, backup and resiliency planning. IaaS platforms offer highly scalable resources, which can be adjusted on-demand. This makes IaaS well suited for workloads, which are temporary, experimental or change unexpectedly. Other characteristics of IaaS environments include the automation of administrative tasks, dynamic scaling, desktop virtualization and policy-based services.

Data center administrators have a long to-do list when it comes to infrastructure monitoring. From server and equipment monitoring – and in some cases, mainframe monitoring – it is a practice that is often difficult to juggle, especially if you work in a large data center like the LIT JINR. However, monitoring is an essential task. By obtaining the needed data, one can increase security and scalability, efficiently automate and better align resources with capacity needs.

The JINR cloud service is currently based on OpenNebula software version 4.12. The transition to new version 5.0 is planning in the nearest future. There are many changes in the OpenNebula version 5.0 such as fixed critical errors, improvements in user interface and ways of interacting with the equipment [2]. The developers changed inner mechanisms of storing data on the state of physical resources and the problem of incompatibility of the module being used for visualization of the statistics was found.

The used module for visualization of the statistics is implement in the form of the web page in the main window of the management panel of OpenNebula web-inteface called “Sunstone” [3]. The module consists of a set of pie charts displaying the percentage of the total and allocated amount of resources of their total value by clusters (Fig. 1). Moreover, there is the table view of the statistics (Fig. 2).

A pie chart is a circular statistical graphic, which is divided into slices to illustrate numerical proportion. The arc length of each slice in a pie chart is proportional to the quantity it represents.

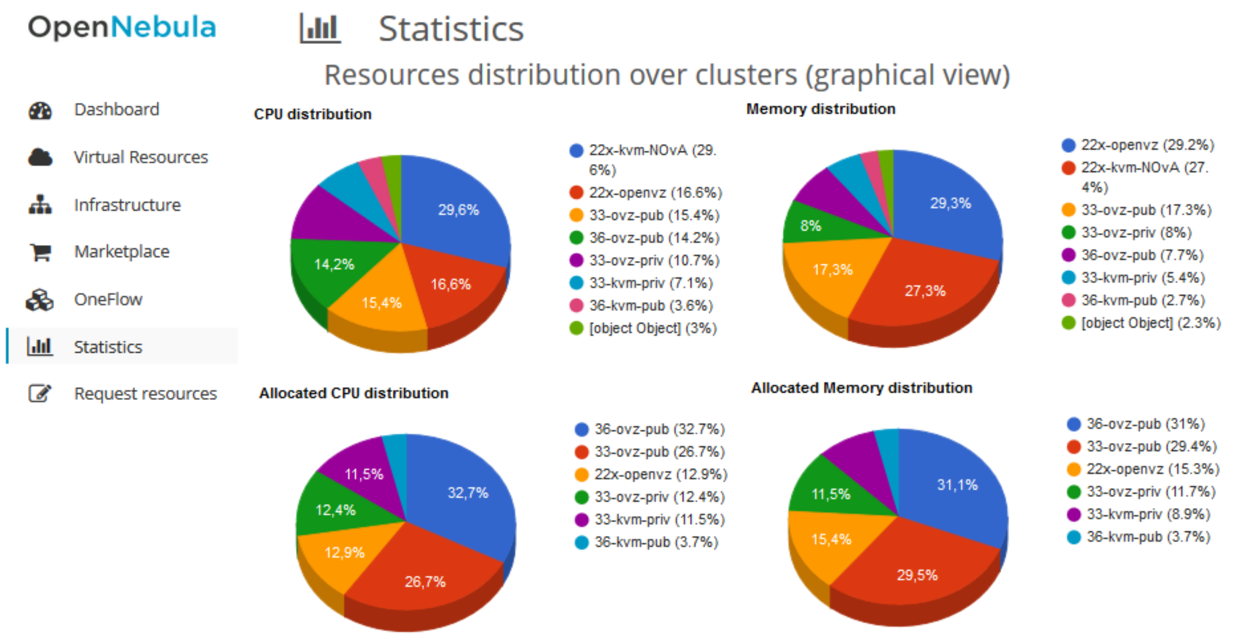
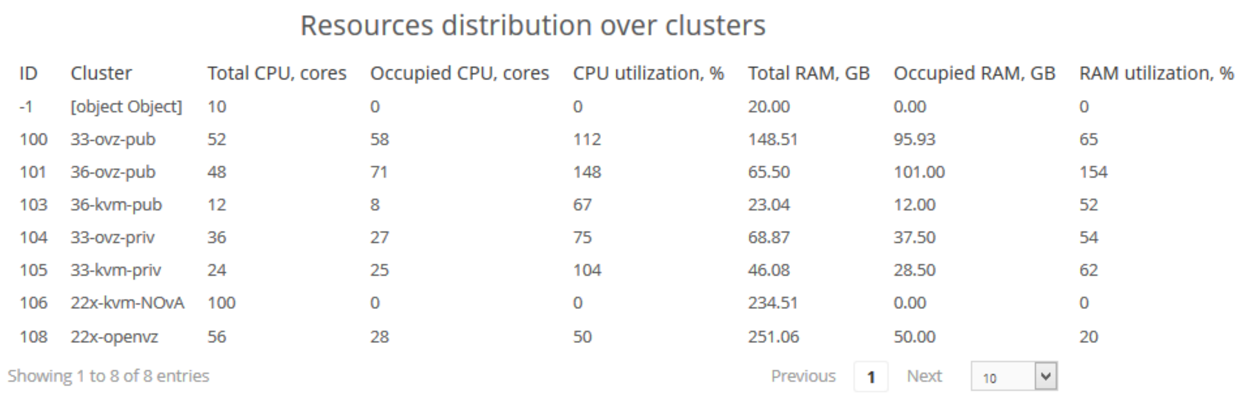


Figure 1 – The current state of the module of statistics of resource distribution over clusters



Resources distribution over clusters

ID	Cluster	Total CPU, cores	Occupied CPU, cores	CPU utilization, %	Total RAM, GB	Occupied RAM, GB	RAM utilization, %
-1	[object Object]	10	0	0	20.00	0.00	0
100	33-ovz-pub	52	58	112	148.51	95.93	65
101	36-ovz-pub	48	71	148	65.50	101.00	154
103	36-kvm-pub	12	8	67	23.04	12.00	52
104	33-ovz-priv	36	27	75	68.87	37.50	54
105	33-kvm-priv	24	25	104	46.08	28.50	62
106	22x-kvm-NOvA	100	0	0	234.51	0.00	0
108	22x-openvz	56	28	50	251.06	50.00	20

Showing 1 to 8 of 8 entries Previous 1 Next 10

Figure 2 – Table view of the module of statistics of the resource distribution over clusters

In addition, for resource distribution over clusters the managers need the statistics of department resource usage. Such statistics is represented in the pie charts and the table view (Fig. 3, 4)

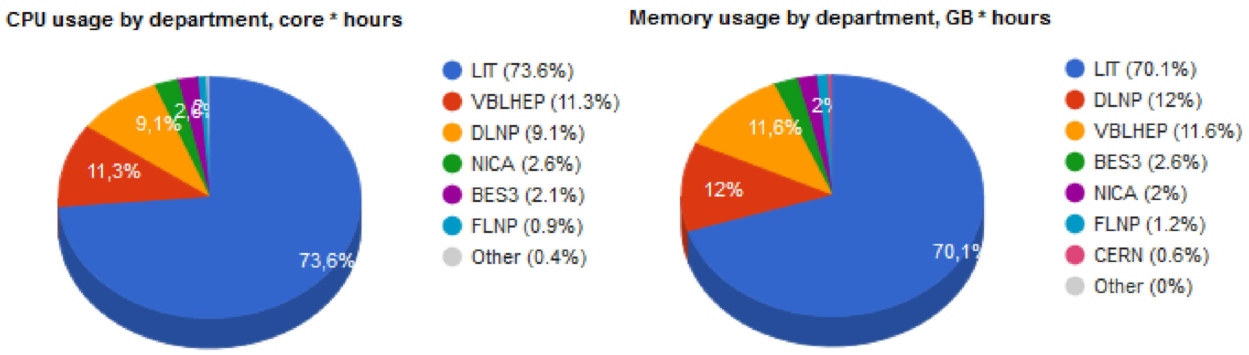
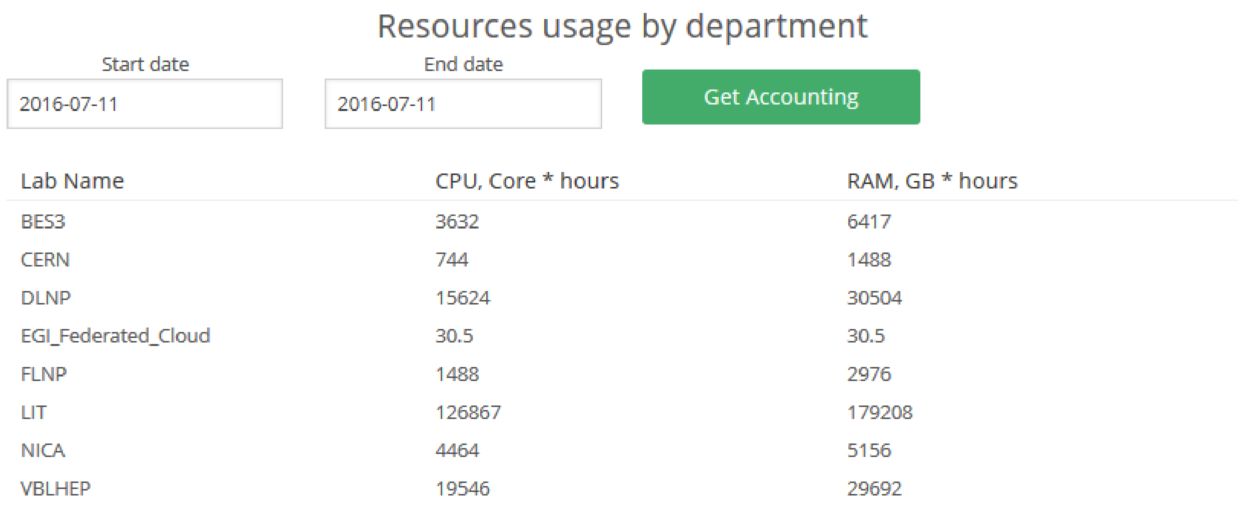


Figure 3 – The current state of the module of statistics of resource distribution over departments



Resources usage by department

Start date	End date	
2016-07-11	2016-07-11	Get Accounting

Lab Name	CPU, Core * hours	RAM, GB * hours
BES3	3632	6417
CERN	744	1488
DLNP	15624	30504
EGI_Federated_Cloud	30.5	30.5
FLNP	1488	2976
LIT	126867	179208
NICA	4464	5156
VBLHEP	19546	29692

Figure 4 – Table view of the module of statistics of the resource distribution over departments

Aforementioned transition to the new version of OpenNebula makes problem of usage the module of statistics designed in JINR. In conjunction with this it was necessary to modernize the current statistics module and transfer it to an external service with the aim of independence from internal mechanisms and to guarantee successful upgrades OpenNebula platform in the future. It was suggested to store the analyzed indicators in a database, which allows to analyze their changes over time and to obtain the dynamics for the selected period.

The choice of the new solution

To visualize the same statistics on the distribution of resources of the JINR cloud the *Grafana* system was chosen. It provides a user-friendly interface through a web browser displaying various kinds of statistical metrics in real-time, gives flexible and functional ways to customize the layout of charts and graphs [4].

Grafana is an open source feature rich metrics dashboard and graph editor for metrics databases. It provides a powerful and elegant way to create, share, and explore data and dashboards from disparate metric databases. It supports a wide variety of graphing options for ultimate flexibility. Grafana supports authenticated login and a basic role based access control implementation. Grafana is deployed as a single software installation. It includes a web-server and presentation logic. It is written in Go and Javascript.

Grafana does not store or collect data. For such purpose one can use such databases as Graphite, Elasticsearch, Prometheus, InfluxDB, OpenTSDB and KairosDB. Among them, the majority is time-series databases (TSDB). According to «DB Engines», the most popular database of this type is *InfluxDB* [5].

InfluxDB is an open source database written in Go specifically to handle time series data with high availability and high performance requirements. InfluxDB is meant to be used as a backend storage for many use cases involving large amounts of timestamped data, including DevOps monitoring, application metrics, IoT sensor data, and real-time analytics. It has a simple, high performing write and query HTTP(S) APIs. Custom high performance datastore is written specifically for time series data. The TSM engine allows a high ingest speed and data compression. Expressive SQL-like query language is tailored to query aggregated data easily. All of the listed features that InfluxDB currently supports make it a great choice for working with time series data.

InfluxDB has a simple and functional libraries for data transfer – influxdb-ruby [6]. It is used for development in the new statistics module.

OpenNebula was designed to be easily adapted to any infrastructure and easily extended with new components. The result is a modular system that can implement a variety of cloud architectures and can interface with multiple services (Fig. 5).

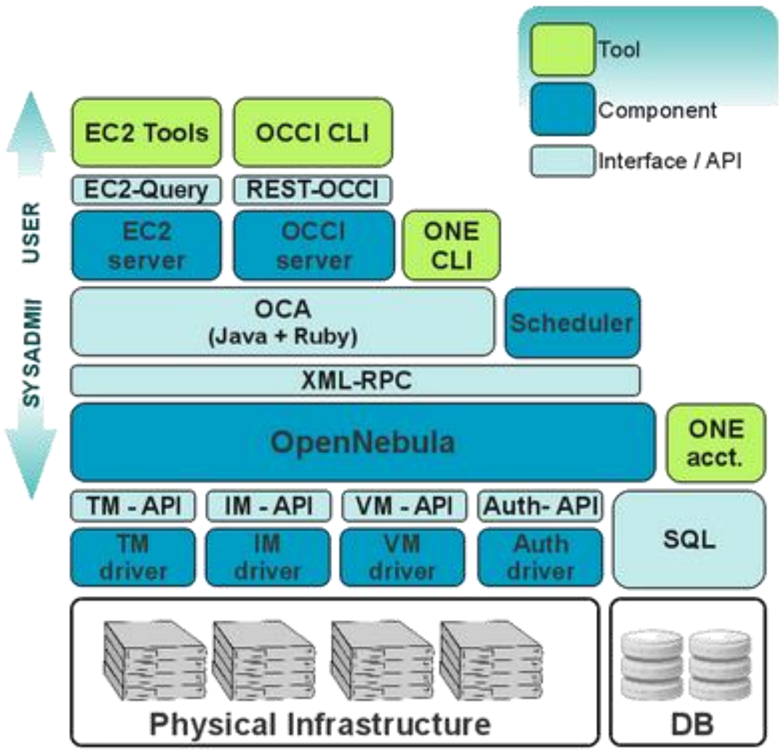


Figure 5. The cloud architectures and the system interfaces of OpenNebula

System interfaces expose the full functionality of OpenNebula and are used to adapt and tune the behavior of cloud to the target infrastructure. The XML-RPC interface is the primary interface, exposing all the functionality for interaction with OpenNebula daemon. Any resource, including VMs, virtual networks, images, users, hosts and clusters is controlled and managed through the XML-RPC interface.

XML-RPC works by sending an HTTP request to a server supporting that protocol. The client in that case is typically a software which needs to call a single method on the remote system. Multiple input parameters can be passed to the remote method, one value is returned. The parameter types allow nesting of parameters into maps and lists, thus larger structures can be transported. Therefore, XML-RPC can be used to transport objects or structures both as input and as output parameters.

The response that contains the current data about cloud resources is standardized and it was not changed with the change of version of OpenNebula.

It was decided to use the Ruby programming language for the storage of data. That is why a requested XML-document with statistics will be carried out with of Ruby-module XML-RPC::Client [7], and document processing – with help of the Nokogiri library [8].

The architecture of component interaction

According to selected technologies the following architecture for collection, storage and visualization was designed (Appendix A, Fig. A.1). The *User* can pass to the server #1 (*Server #1*) and #2 (*Server #2*) on their URL-address through a *Browser*. Following the Server #1, he can interact with the Grafana (view graphs, charts and tables obtained on the basis of data from InfluxDB). If the User typed the address of Server #1, then he will be able to manage cloud resources and virtual machines.

Ruby scripts for collecting statistics were developed. A cron job runs that scripts within certain period of time. These scripts collect statistics from OpenNebula through XML-RPC, convert the response into the structured format and write to InfluxDB via HTTP API.

Hence the following tasks should be resolved:

1) to develop a program for collecting, converting and storing statistical data;

2) to design and configure database;

3) to design and develop a module for statistical data visualization.

Description of the developed program

The main logics of the program is implemented in the Ruby-scripts. Following the standards of object-oriented programming, all functions are presented in the following classes:

1. The class for collect the data about resource usage from OpenNebula.
2. The class for write the prepared data to InfluxDB.
3. Auxiliary classes.

The main ones are considered below.

Collect the data about resource usage from OpenNebula

The class **OneParser** collects and prepares data from OpenNebula (List. 1). The full listing are shown in Appendix B (List. B.1).

Listing 1. Description of class **OneParser**

```
class OneParser
  def initialize(hostname)
  def get_system_version
  def get_clusterpool_info
  def get_host_info host
  def get_userpool_info
  def get_user_info user
end
```

The creation of object of this class is in a file **main.rb** which consistently invokes methods to obtain statistics on the clusters, departments and users (List. 2). The processing of the XML-document receiving from OpenNebula occurs in these methods.

Listing 2. Working with class **OneParser**

```
one = OneParser.new(ONE_HOST)
clusters = one.get_clusterpool_info
clusters.each do |c|
  c.hosts.each do |h|
    one.get_host_info h
  end
end
departments, users = one.get_userpool_info
users.each do |u|
  one.get_user_info u
  if (!(u.lab.to_s.empty?)) &&
(departments.include?(u.lab.to_s))
    departments[u.lab.to_s].add_user u
  end
end
```

Writing data to InfluxDB

The class **Influx** is responsible for recording data (List. 3). The methods, which are invoked from the main file (List. 4), make sending the read data to InfluxDB. The full listing is showed in Appendix B (List B.2).

Listing 3. Description of class **Influx**
```
class Influx
  def initialize(hostname)
  def write_clusters_info clusters
  def write_departments_info departments
end
```

Listing 4. Working with class **Influx**
```
inf = Influx.new(INFLUXDB_HOST)
inf.write_clusters_info(clusters)
inf.write_departments_info(departments.values)
```

Auxiliary classes

To explain that classes one needs to briefly describe a concept of virtualization and some terms in that fields.

Each department has employees. They need some virtual resources. The employee has an user account on the cloud service. User creates virtual machines (VMs) which runs on the host (physical computer that located at a datacenter). To run VMs some physical resources of host (number of CPU cores, amount of RAM and disk space) are allocated. The VM consumes them in active state. The set of hosts are grouped into so called cluster which can has a centralize control.

In this way, there is a need for the following classes:

- Class **Host** (List. 5) is for a physical computer;
- Class **Cluster** (List. 6) is for a group of hosts;

- Class **User** (List. 7) is for a user account;
- Class **Department** (List. 8) is for a department that includes employees.

Listing 5. Description of class **Host**
```
class Host
  attr_accessor :id,         :name,
   :cpu_total,  :cpu_real, :cpu_allocated,
   :ram_total,  :ram_real, :ram_allocated,
   :disk_total, :disk_real

  def initialize id
end
```

Listing 6. Description of class **Cluster**
```
class Cluster
  attr_accessor  :id,         :name,
    :cpu_total,  :cpu_real,  :cpu_allocated,
    :ram_total,  :ram_real,  :ram_allocated,
    :disk_total, :disk_real, :hosts

  def initialize id
  def add_host h
end
```

Listing 7. Description of class **User**
```
class User
  attr_accessor :id, :name, :gname, :lab,
   :cpu_real,  :cpu_allocated,
   :ram_real,  :ram_allocated,
   :disk_real, :disk_allocated,
   :vms_real,  :vms_allocated
  def initialize id
end
```

Listing 8. Description of class **Department**
```
class Department
  attr_accessor :name,
   :cpu_real,  :cpu_allocated,
   :ram_real,  :ram_allocated,
   :disk_real, :disk_allocated,
   :vms_real,  :vms_allocated
  def initialize name
  def add_user u
end
```

The classes **Host** (List. 5) and **Cluster** (List. 6) make the necessary calculations of the following parameters:

- total numbers of physical processors (CPU) (attribute *cpu_total*);

- number of CPU allocated to users (attribute *cpu_allocated*);
- real usage of CPU (attribute *cpu_real*);
- total amount of RAM (attribute *ram_total*);
- the amount of RAM allocated to users (attribute *ram_allocated*);
- real usage of RAM (attribute *ram_real*);
- total amount of disk space (attribute *disk_total*);
- real usage of disk space (attribute *disk_real*);

Unlike the classes Host and Cluster, classes **User** (List. 7) and **Department** (List. 8) have additional parameters as:

- total numbers of created virtual machines (attribute *vms_real*);
- quantity of the allocated places under the created virtual machines (attribute *vms_allocated*)

Description of the database

Aforementioned InfluxDB is used for the long-term storage. It is intended for storing *time-series* data which is nothing more than a sequence of data points, typically consisting of sequential measurements made from the same source over a time interval [9].

The measurement is a part of InfluxDB's structure that describes data stored in the associated fields. Measurements are strings. Measurement can have the tag value or a field value. Tag values are strings and metadata is stored then. Tag values are indexed and so increases query performance.

Description of the database scheme

In accordance with the described requirements the next tables are designed (Fig. 6):

- **rd_cluster** – for storage of measurements of resources used by clusters;
- **rd_department** – for storage of measurements of resources used by departments;

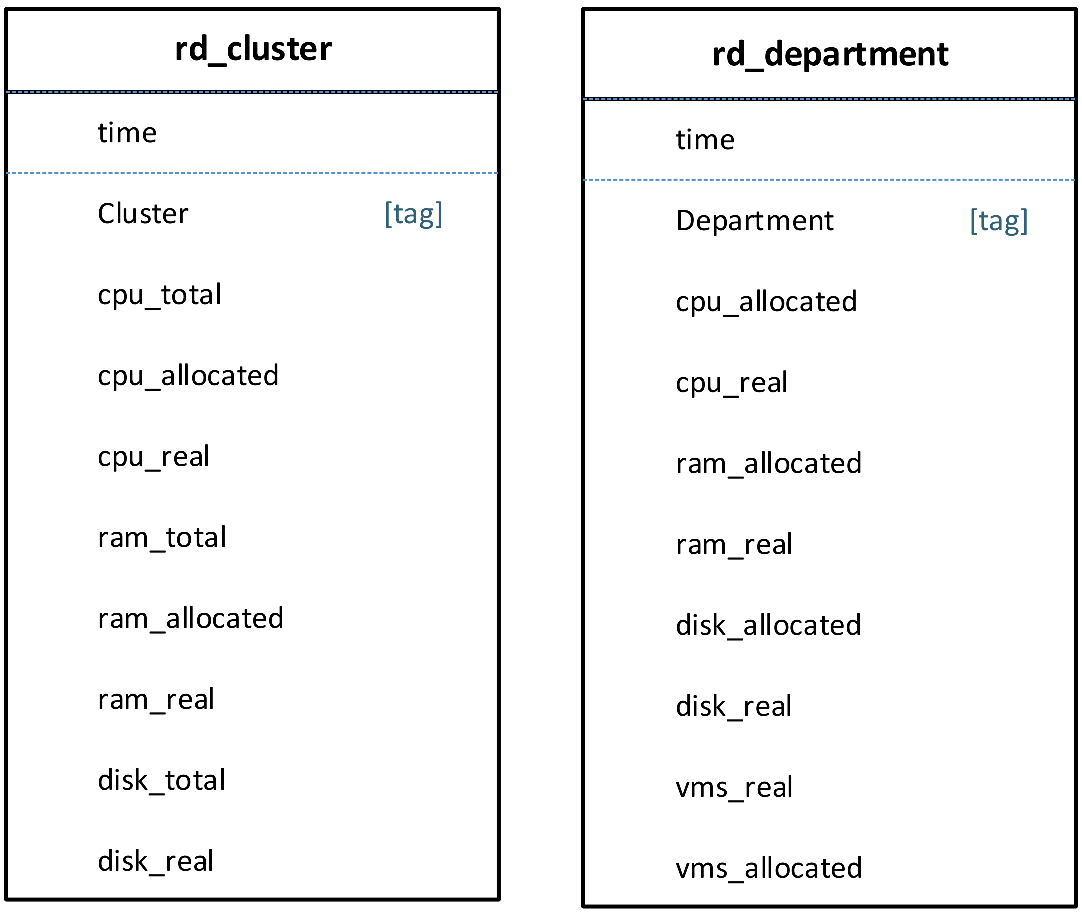


Figure 6 – The database scheme

Description of developed module for visualization of statistics

After creating and configuring database and running scripts to collect and store statistics it was necessary to develop charts, graphs and tables to display it for managers.

Two panels to display resource usage statistics has been developed:

- resource distribution over clusters;
- resource distribution over departments;

Dashboards (or panel) are a collection of widgets that give an overview of the reports and needed metrics. Dashboards let monitor many metrics at once, so one can quickly check the health of accounts or see correlations between different reports.

The process of creating each of the panels is shown below.

Dashboard of statistics of resources distribution over clusters

The request (List. 9) fills the table (Table 1) that displays the last state about resources used in each cluster.

Listing 9. Request the latest data of clusters
```
SELECT LAST("cpu_total") / 100 AS "Total CPU, cores",
       LAST("cpu_allocated") / 100 AS "Allocated CPU, cores",
       LAST("cpu_allocated") / LAST("cpu_total") *100 AS
"Allocated CPU, %",
       (LAST("cpu_total")-LAST("cpu_allocated")) / 100 AS "Free
CPU, cores",
       (LAST("cpu_total")-
LAST("cpu_allocated"))/LAST("cpu_total")*100 AS "Free CPU, %",
       LAST("cpu_real") / LAST("cpu_total") *100 AS "Real CPU
usage, %",
       LAST("ram_total") /1024/1024 AS "Total RAM, GB",
       LAST("ram_allocated")/1024/1024 AS "Allocated RAM, GB",
       LAST("ram_allocated") / LAST("ram_total") *100 AS
"Allocated RAM, %",
       (LAST("ram_total")-LAST("ram_allocated"))/1024/1024 AS
"Free RAM, GB",
       (LAST("ram_total")-
LAST("ram_allocated"))/LAST("ram_total")*100 AS "Free RAM, %",
       LAST("ram_real") / LAST("ram_total") *100 AS "Real RAM
usage, %"
FROM "rd_cluster"
WHERE $timeFilter
GROUP BY "Cluster" fill(NONE)
```

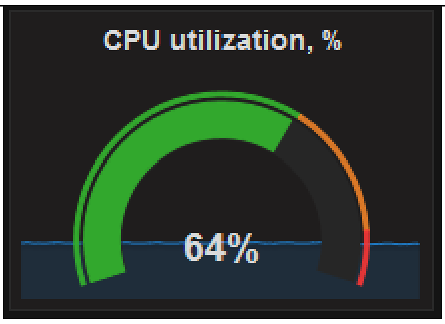
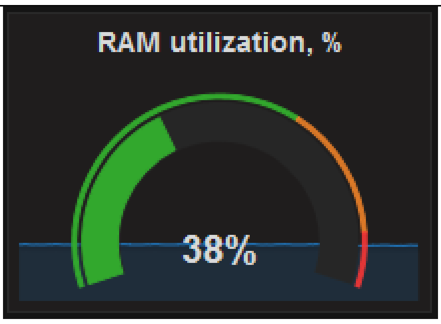
Table 1. Resource distribution over clusters (table view)

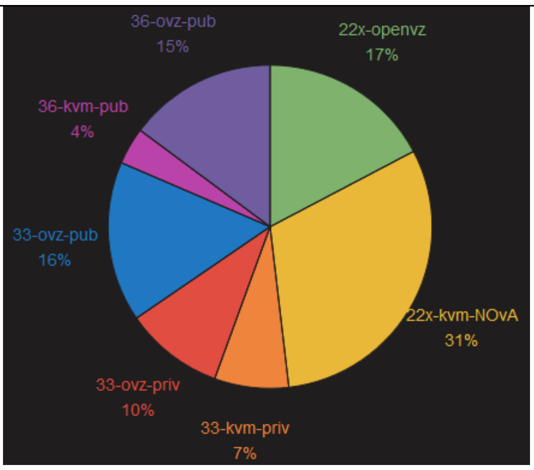
Cluster	**Total CPU, cores**	**Allocated CPU, cores**	**Allocated CPU, %**	**Free CPU, cores**	**Free CPU, %**	**Real CPU usage, %**	**Total RAM, GB**	**Allocated RAM, GB**	**Allocated RAM, %**	**Free RAM, GB**	**Free RAM, %**	**Real RAM usage, %**
36-ovz-pub	48	71	148	-23	-48	3.0	65.50	101.00	154	-35.50	-54	28.3
36-kvm-pub	12	8	67	4	33	8.8	23.04	12.00	52	11.04	48	41.2
33-ovz-pub	52	56	108	-4	-8	4.6	148.51	91.43	62	57.09	38	12.3
33-ovz-priv	32	24	75	8	25	0.4	61.22	35.00	57	26.22	43	10.3
33-kvm-priv	24	16	67	8	33	1.2	46.08	18.50	40	27.58	60	28.2
22x-ovz-NOvA	0	0	0	0	0	0	0	0	0	0	0	0
22x-kvm-NOvA	100	1	1	99	99	0.1	234.51	0.50	0	234.01	100	1.3

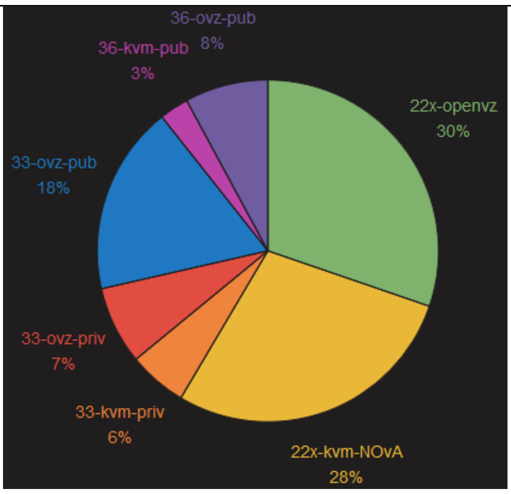
22x-openvz	56	29	52	27	48	0.1	251.06	51.00	20	200.06	80	1.5
Total	**324**	**205**	**63**	**119**	**37**	**1.7**	**829.93**	**309.43**	**37**	**520.50**	**63**	**8.7**

The Table 2 contains other requests and graphical forms based on obtained the results.

Table 2. The forms for displaying resource distribution

Description	**Request**	**Form**
Display statistics of CPU usage for selected cluster on percentage form	`SELECT (LAST("cpu_allocated")/ LAST("cpu_total") *100) FROM "rd_cluster" WHERE "Cluster" =~ /^$selected_cluster$/ AND $timeFilter GROUP BY TIME($interval) fill(NULL)`	
Display statistics of RAM usage for selected cluster on percentage form	`SELECT (LAST("ram_allocated")/ LAST("ram_total") *100) FROM "rd_cluster" WHERE "Cluster" =~ /^$selected_cluster$/ AND $timeFilter GROUP BY TIME($interval) fill(NULL)`	

Display statistics of distribution CPUs over clusters	`SELECT (LAST("cpu_total")/100)`
	`FROM "rd_cluster"`
	`WHERE $timeFilter`
	`  AND CLUSTER !~ "/---Total for all clusters---/"`
	`GROUP BY "Cluster"`
		

Display statistics of distribution RAM over clusters	`SELECT`
	`LAST("ram_total")`
	`      /1024/1024`
	`FROM "rd_cluster"`
	`WHERE $timeFilter`
	`  AND CLUSTER !~ "/---Total for all clusters---/"`
	`GROUP BY "Cluster"`
		

Line graph allows to estimate changes of the resources usage. For example, Fig. 7 shows the change of the number of CPUs over clusters which allocated to users. This graph is obtained from a query, which is listed in List. 10.

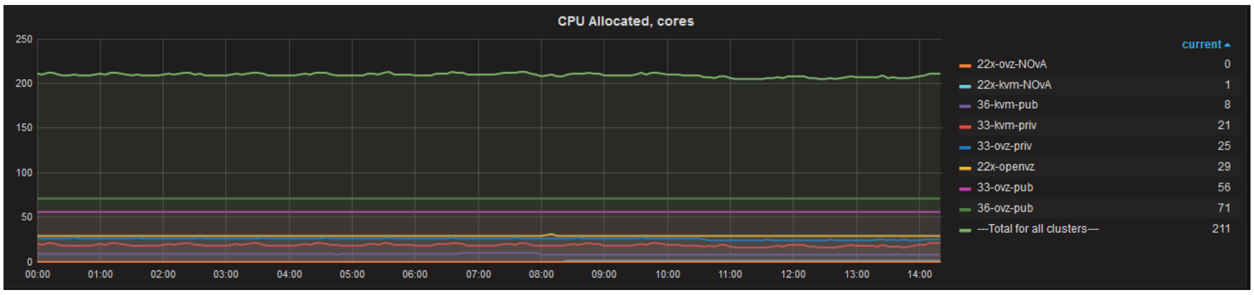


Figure 7 – Request the data about allocated processors

Listing 10. Request the data about processors
```
SELECT (mean("cpu_allocated")/100)
FROM "rd_cluster"
WHERE $timeFilter
  AND "Cluster" =~ /^$selected_cluster$/
GROUP BY TIME($interval),
         "Cluster" fill(NULL)
```

Similarly, graphs and charts for undocumented metrics are created. The result is a full dashboard shown on the Fig. C.1 in Appendix C.

Dashboard of statistics of resources distribution over departments

The request (List. 11) fill the table (Table 3) that displays the last state about resources used in each cluster.

Listing 11. Request the latest data on departments
```
SELECT LAST("cpu_allocated") AS "Allocated CPU, cores",
       LAST("cpu_real") AS "Real CPU usage, cores",
       LAST("cpu_real") / LAST("cpu_allocated") *100 AS "Real
CPU usage, %",
       LAST("ram_allocated")/1024 AS "Allocated RAM, GB",
       LAST("ram_real") /1024 AS "Real RAM usage, GB",
       LAST("ram_real") / LAST("ram_allocated") *100 AS "Real
RAM usage, %",
       LAST("vms_allocated") AS "Allocated VMs",
       LAST("vms_real") AS "Real VMs usage",
       LAST("vms_real") / LAST("vms_allocated") *100 AS "Real
VMs usage, %"
FROM "rd_department"
WHERE $timeFilter
GROUP BY "Department" fill(NONE)
```

Table 3. Resource distribution over departments (table view)

Department	Allocated CPU, cores	Real CPU usage, cores	Real CPU usage, %	Allocated RAM, GB	Real RAM usage, GB	Real RAM usage, %	Allocated VMs	Real VMs usage	Real VMs usage, %
VBLHEP	33	27	81.8	47.00	41.00	87.2	8	4	50
NICA	7	6	85.7	7.00	6.93	99.0	3	2	66.7
LIT	320	173	54.1	571.06	246.50	43.2	245	116	47.3
FLNP	2	0	0	4.00	0	0	1	0	0
EGI_Federated_Cloud	1	0	0	1.00	0	0	1	0	0
DLNP	24	17	70.8	44.00	33.00	75.0	12	5	41.7
CERN	10	1	10.0	18.00	2.00	11.1	10	1	10.0
BES3	5	5	100.0	10.00	5.00	50.0	5	5	100.0
---Total---	**402**	**229**	**57.0**	**702.06**	**334.43**	**47.6**	**285**	**133**	**46.7**

The result is a full dashboard about the resource distribution over departments which is shown on the Fig. C.2 in Appendix C.

.

Conclusion

Thus, the developed system allows to store the JINR cloud infrastructure usage information in a long-term mode, give the ability to retrieve usage statistics for a long period of time and makes assessment of actual cloud resources usage. In addition to that, one can evaluate an actual use of the cloud resources during certain period of time. The designed charts and graphs showing statistics in a convenient format can be used for annual reports of the Laboratory of Information Technologies. All technologies and tools used in that development are free open source software supporting and developing by a large community of users.

The developed system utilizes OpenNebula XML-RPC which is stable and is not changed over OpenNebula releases It makes developed service independent of OpenNebula version and is not blocking issue any longer for JINR cloud software upgrades.

References

1. N. Balashov, A. Baranov, V. Korenkov, N. Kutovskiy, A. Nechaevskiy, R.Semenov. JINR Cloud Service: Status and Perspectives. Trudy Instituta sistemnogo programmirovania RAN [Proceedings of the Institute for system programming of the RAS], 2015, vol. 27, issue 6, pp. 345-354
2. OpenNebula Release Notes 5.0 / OpenNebula Project
   URL: http://docs.opennebula.org/5.0/intro_release_notes/release_notes
3. JINR: OpenNebula Sunstone – Cloud Operations Center
   URL: https://cloud.jinr.ru/
4. Grafana - Beautiful Metrics, Analytics, dashboards and monitoring
   URL: http://grafana.org/
5. DB-Engines Ranking - popularity ranking of time Series DBMS
   URL: http://db-engines.com/en/ranking/time+series+dbms
6. Ruby client for InfluxDB
   URL: https://github.com/influxdata/influxdb-ruby

7. Class XMLRPC Client (Ruby 2.3.1)
   URL:http://ruby-doc.org/stdlib-2.3.1/libdoc/xmlrpc/rdoc/XMLRPC/Client
8. Module Nokogiri
   URL:http://www.rubydoc.info/github/sparklemotion/nokogiri/Nokogiri
9. InfluxDB – Time-Series Data Storage / InfluxData
   URL: https://influxdata.com/time-series-platform/influxdb/

Acknowledgement

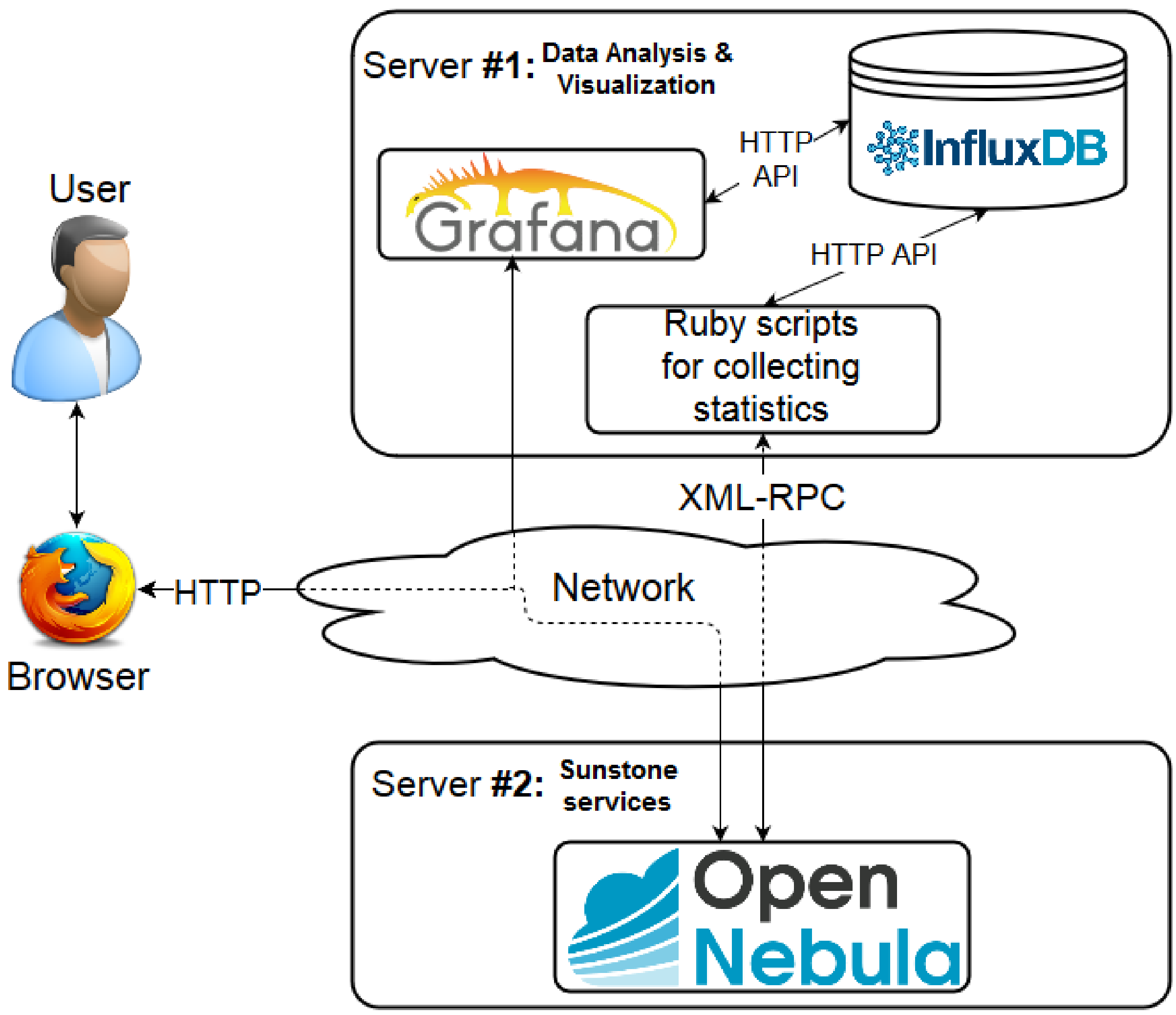
I would like to express my appreciation to Nikolay Kutovskiy, my practice supervisor and S. Z. Pakuliak, Director of University Centre of Joint Institute for Nuclear Research, for giving me an opportunity to participate in Student Summer Program 2016. I also wish to acknowledge the help provided by Nikita Balashov and Aleksandr Baranov for consulting me in my work.

I would like to thank to Elena Karpova for excellent organization of our practice and helped us to see greatest physical things and facilities at JINR.

Finally, I feel all the gratitude for the woman who works in the dining room and makes my food warm.

Appendix A

The architecture of developed system



Server #1: Data Analysis & Visualization
InfluxDB
HTTP API
Grafana
HTTP API
Ruby scripts for collecting statistics
User
XML-RPC
HTTP
Network
Browser
Server #2: Sunstone services
Open Nebula

Figure A.1 – The scheme of architecture of the components interaction

Appendix B

The implementation of main classes

Listing B.1. Implementation of class **OneParser**

```
#!/usr/bin/env ruby
ONE_CONF = 'configs/one.conf'.freeze

require 'yaml'
require 'xmlrpc/client'
require 'nokogiri'

require_relative 'host'
require_relative 'cluster'
require_relative 'user'

class OneParser
  def initialize(hostname)
      @hostname = hostname
      public_cloud_one_conf = YAML.load(File.read(ONE_CONF))
      @localhost = public_cloud_one_conf[@hostname]
      @local_client = XMLRPC::Client.new(@localhost['hostname'],
@localhost['rpc_path'], @localhost['port'])
      @local_credentials =
"#{@localhost['username']}:#{@localhost['password']}"
  end


  def get_system_version
      begin
          response = @local_client.call('one.system.version',
@local_credentials)
      rescue
        puts "#{Time.now.utc}: Error. Cannot call 'one.system.version'...
Exit"
        exit -1
      rescue XMLRPC::FaultException => e
          puts 'Error:'
          puts e.faultCode
          puts e.faultString
          exit -1
      end
      version = response[1]
      version
  end


  def get_clusterpool_info
    begin
        response = @local_client.call('one.clusterpool.info',
@local_credentials)
    rescue
      puts "#{Time.now.utc}: Error. Cannot call 'one.clusterpool.info'...
Exit"
      exit -1
    rescue XMLRPC::FaultException => e
        puts 'Error:'
        puts e.faultCode
        puts e.faultString
        exit -1
    end
```

```
      if response[0] != true
        puts "Error:" + !response[0]
        puts response[2]
        exit -1
      end
      doc = Nokogiri::XML.parse(response[1])

      clusters = []
      doc.xpath('//CLUSTER_POOL/CLUSTER').each do |cluster|
        c = Cluster.new cluster.xpath('ID').text.to_i
        c.name = cluster.xpath('NAME').text
        cluster.xpath('HOSTS/ID').each do |host_id|
          h = Host.new host_id.text
          c.add_host h
        end
        clusters.push c
      end
      return clusters
    end


    def get_host_info host
      begin
          response = @local_client.call('one.host.info', @local_credentials,
  host.id)
      rescue
        puts "#{Time.now.utc}: Error. Cannot call 'one.host.info'... Exit"
        exit -1
      rescue XMLRPC::FaultException => e
          puts 'Error:'
          puts e.faultCode
          puts e.faultString
          exit -1
      end
      if response[0] != true
        puts "Error:"
        puts response[0]
        puts response[2]
        exit -1
      end
      doc = Nokogiri::XML.parse(response[1])

      host.name = doc.xpath('//HOST/NAME').text
      host.cpu_total = doc.xpath('//HOST_SHARE/MAX_CPU').text.to_i
      host.cpu_allocated = doc.xpath('//HOST_SHARE/CPU_USAGE').text.to_i
      host.cpu_real = doc.xpath('//HOST_SHARE/USED_CPU').text.to_i
      host.ram_total = doc.xpath('//HOST_SHARE/MAX_MEM').text.to_i
      host.ram_allocated = doc.xpath('//HOST_SHARE/MEM_USAGE').text.to_i
      host.ram_real = doc.xpath('//HOST_SHARE/USED_MEM').text.to_i
      host.disk_total = doc.xpath('//HOST_SHARE/MAX_DISK').text.to_i
      host.disk_real = doc.xpath('//HOST_SHARE/USED_DISK').text.to_i
    end


    def get_userpool_info
      begin
          response = @local_client.call('one.userpool.info',
  @local_credentials)
      rescue
        puts "#{Time.now.utc}: Error. Cannot call 'one.userpool.info'... Exit"
        exit -1
      rescue XMLRPC::FaultException => e
          puts 'Error:'
          puts e.faultCode
          puts e.faultString
          exit -1
      end

      if response[0] != true
        puts "Error:" + !response[0]
        puts response[2]
        exit -1
      end
      doc = Nokogiri::XML.parse(response[1])

      users = {}
      departments = []
      doc.xpath('//USER_POOL/USER').each do |user|
        id = user.xpath('ID').text.to_i
        u = User.new id
        u.name = user.xpath('NAME').text
        u.name = user.xpath('NAME').text
        u.gname = user.xpath('GNAME').text
        u.lab = user.xpath('TEMPLATE/LAB').text
        if !u.lab.to_s.empty?
          departments.push(u.lab)
          users.store(id.to_s, u)
        end
      end
      doc.xpath('//USER_POOL/QUOTAS').each do |quotas|
        id = quotas.xpath('ID').text.to_i
        if users.include?(id.to_s)
          u = users[id.to_s]
          u.cpu_allocated = quotas.xpath('VM_QUOTA/VM/CPU').text.to_i
          u.cpu_real = quotas.xpath('VM_QUOTA/VM/CPU_USED').text.to_i
          u.ram_allocated = quotas.xpath('VM_QUOTA/VM/MEMORY').text.to_i
          u.ram_real = quotas.xpath('VM_QUOTA/VM/MEMORY_USED').text.to_i
          u.disk_allocated =
  quotas.xpath('VM_QUOTA/VM/SYSTEM_DISK_SIZE').text.to_i
          u.disk_real =
  quotas.xpath('VM_QUOTA/VM/SYSTEM_DISK_SIZE_USED').text.to_i
          u.vms_allocated = quotas.xpath('IMAGE_QUOTA/RVMS').text.to_i
          u.vms_real = quotas.xpath('IMAGE_QUOTA/RVMS_USED').text.to_i
        end
      end
      departments.compact!
      departments.uniq!
      return departments, users.values
    end


    def get_user_info user
      begin
          response = @local_client.call('one.user.info', @local_credentials,
  user.id)
      rescue
        puts "#{Time.now.utc}: Error. Cannot call 'one.user.info'... Exit"
        exit -1
      rescue XMLRPC::FaultException => e
          puts 'Error:'
          puts e.faultCode
          puts e.faultString
          exit -1
      end

      if response[0] != true
        puts "Error:" + !response[0]
        puts response[2]
```

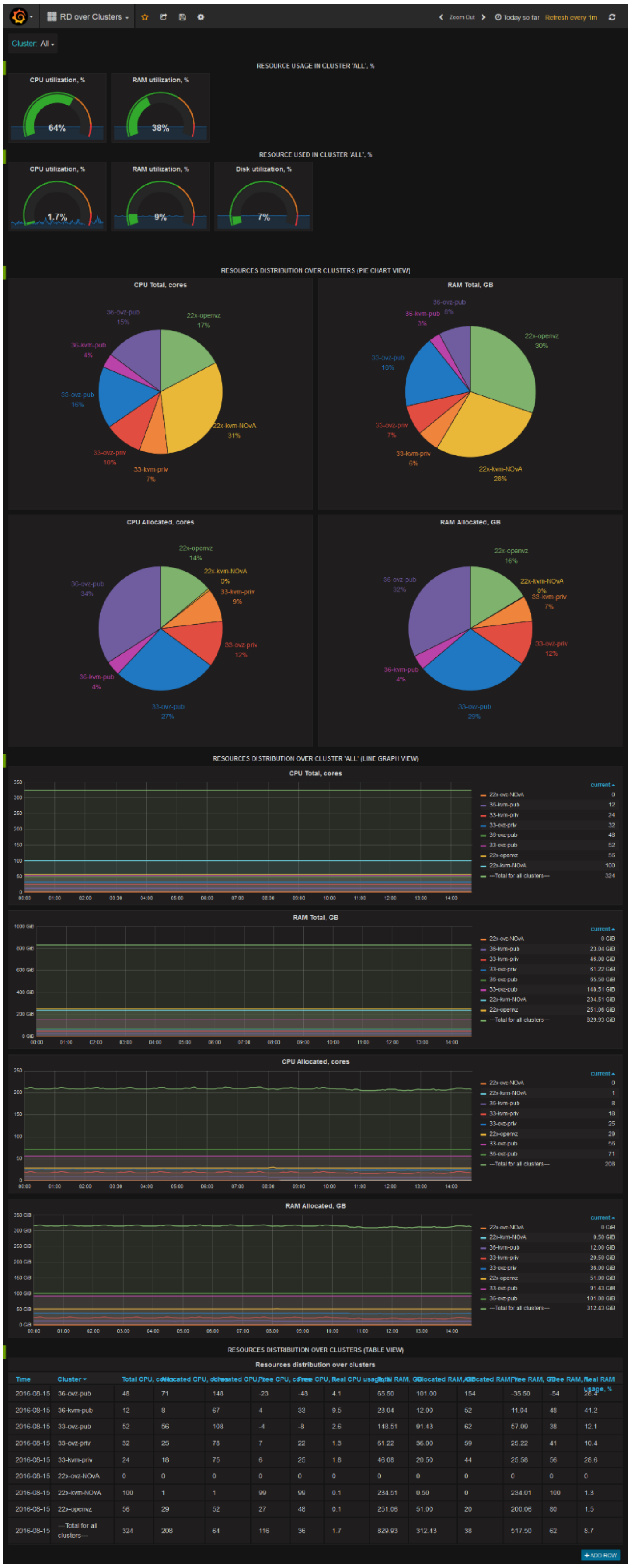
```
        exit -1
      end
      doc = Nokogiri::XML.parse(response[1])

      user.name = doc.xpath('//USER/NAME').text
      user.gname = doc.xpath('//USER/GNAME').text
      user.lab = doc.xpath('//USER/TEMPLATE/LAB').text
      user.cpu_allocated = doc.xpath('//USER/VM_QUOTA/VM/CPU').text.to_i
      user.cpu_real = doc.xpath('//USER/VM_QUOTA/VM/CPU_USED').text.to_i
      user.ram_allocated = doc.xpath('//USER/VM_QUOTA/VM/MEMORY').text.to_i
      user.ram_real = doc.xpath('//USER/VM_QUOTA/VM/MEMORY_USED').text.to_i
      user.disk_allocated =
doc.xpath('//USER/VM_QUOTA/VM/SYSTEM_DISK_SIZE').text.to_i
      user.disk_real =
doc.xpath('//USER/VM_QUOTA/VM/SYSTEM_DISK_SIZE_USED').text.to_i
      user.vms_allocated = doc.xpath('//USER/VM_QUOTA/VM/VMS').text.to_i
      user.vms_real = doc.xpath('//USER/VM_QUOTA/VM/VMS_USED').text.to_i
    end
end
```

Listing B.2. Implementation of class **Influx**

```
#!/usr/bin/env ruby
INFLUX_CONF = 'configs/influx.conf'.freeze
NAME_OF_TOTAL_CLUSTER = '---Total for all clusters---'
NAME_OF_TOTAL_DEPARTMENTS = '---Total for all LABs---'

require 'yaml'
require 'influxdb'

require_relative 'host'
require_relative 'cluster'
require_relative 'department'

class Influx
  def initialize(hostname)
    f = YAML.load(File.read(INFLUX_CONF))[hostname]
    @measurement_cluster = f['measurement_cluster']
    @measurement_department = f['measurement_department']
    @influxdb = InfluxDB::Client.new      host:           f['host'],
                                          port:           f['port'],
                                          database:       f['database'],
                                          username:       f['username'],
                                          password:       f['password'],
                                          time_precision: f['time_precision']
  end


  def write_clusters_info clusters
      data = []
      total_clusters = Host.new 0
      total_clusters.name = NAME_OF_TOTAL_CLUSTER
      clusters.each do |c|
        data << {
            series: @measurement_cluster,
            values: { cpu_total:     c.cpu_total,
                      cpu_real:      c.cpu_real,
                      cpu_allocated: c.cpu_allocated,
                      ram_total:     c.ram_total,
                      ram_real:      c.ram_real,
                      ram_allocated: c.ram_allocated,
                      disk_total:    c.disk_total,
                      disk_real:     c.disk_real },
            tags: { Cluster: c.name }
        }
        total_clusters.cpu_total = total_clusters.cpu_total + c.cpu_total
        total_clusters.cpu_real = total_clusters.cpu_real + c.cpu_real
        total_clusters.cpu_allocated = total_clusters.cpu_allocated +
c.cpu_allocated
        total_clusters.ram_total = total_clusters.ram_total + c.ram_total
        total_clusters.ram_real = total_clusters.ram_real + c.ram_real
        total_clusters.ram_allocated = total_clusters.ram_allocated +
c.ram_allocated
        total_clusters.disk_total = total_clusters.disk_total + c.disk_total
        total_clusters.disk_real = total_clusters.disk_real + c.disk_real
      end
      data << {
        series: @measurement_cluster,
        values: { cpu_total:     total_clusters.cpu_total,
                  cpu_real:      total_clusters.cpu_real,
                  cpu_allocated: total_clusters.cpu_allocated,
                  ram_total:     total_clusters.ram_total,
                  ram_real:      total_clusters.ram_real,
                  ram_allocated: total_clusters.ram_allocated,
                  disk_total:    total_clusters.disk_total,
                  disk_real:     total_clusters.disk_real },
        tags: { Cluster: total_clusters.name }
      }
      @influxdb.write_points(data)
  end

  def write_departments_info departments
      data = []
      total_departments = Department.new NAME_OF_TOTAL_DEPARTMENTS
      departments.each do |d|
        data << {
            series: @measurement_department,
            values: { cpu_real:       d.cpu_real,
                      cpu_allocated: d.cpu_allocated,
                      ram_real:       d.ram_real,
                      ram_allocated: d.ram_allocated,
                      disk_real:      d.disk_real,
                      disk_allocated:    d.disk_allocated,
                      vms_real:          d.vms_real,
                      vms_allocated:     d.vms_allocated },
            tags: { Department: d.name }
        }
        total_departments.cpu_real = total_departments.cpu_real + d.cpu_real
        total_departments.cpu_allocated = total_departments.cpu_allocated +
d.cpu_allocated
        total_departments.ram_real = total_departments.ram_real + d.ram_real
        total_departments.ram_allocated = total_departments.ram_allocated +
d.ram_allocated
        total_departments.disk_real = total_departments.disk_real +
d.disk_real
        total_departments.disk_allocated += d.disk_allocated
        total_departments.vms_real += d.vms_real
        total_departments.vms_allocated += d.vms_allocated
      end
      data << {
        series: @measurement_department,
        values: { cpu_real:       total_departments.cpu_real,
                  cpu_allocated: total_departments.cpu_allocated,
                  ram_real:       total_departments.ram_real,
                  ram_allocated: total_departments.ram_allocated,
                  disk_real:      total_departments.disk_real,
                  disk_allocated:    total_departments.disk_allocated,
                  vms_real:          total_departments.vms_real,
                  vms_allocated:     total_departments.vms_allocated },
        tags: { Department: total_departments.name }
      }
      @influxdb.write_points(data)
  end
end
```

Appendix C

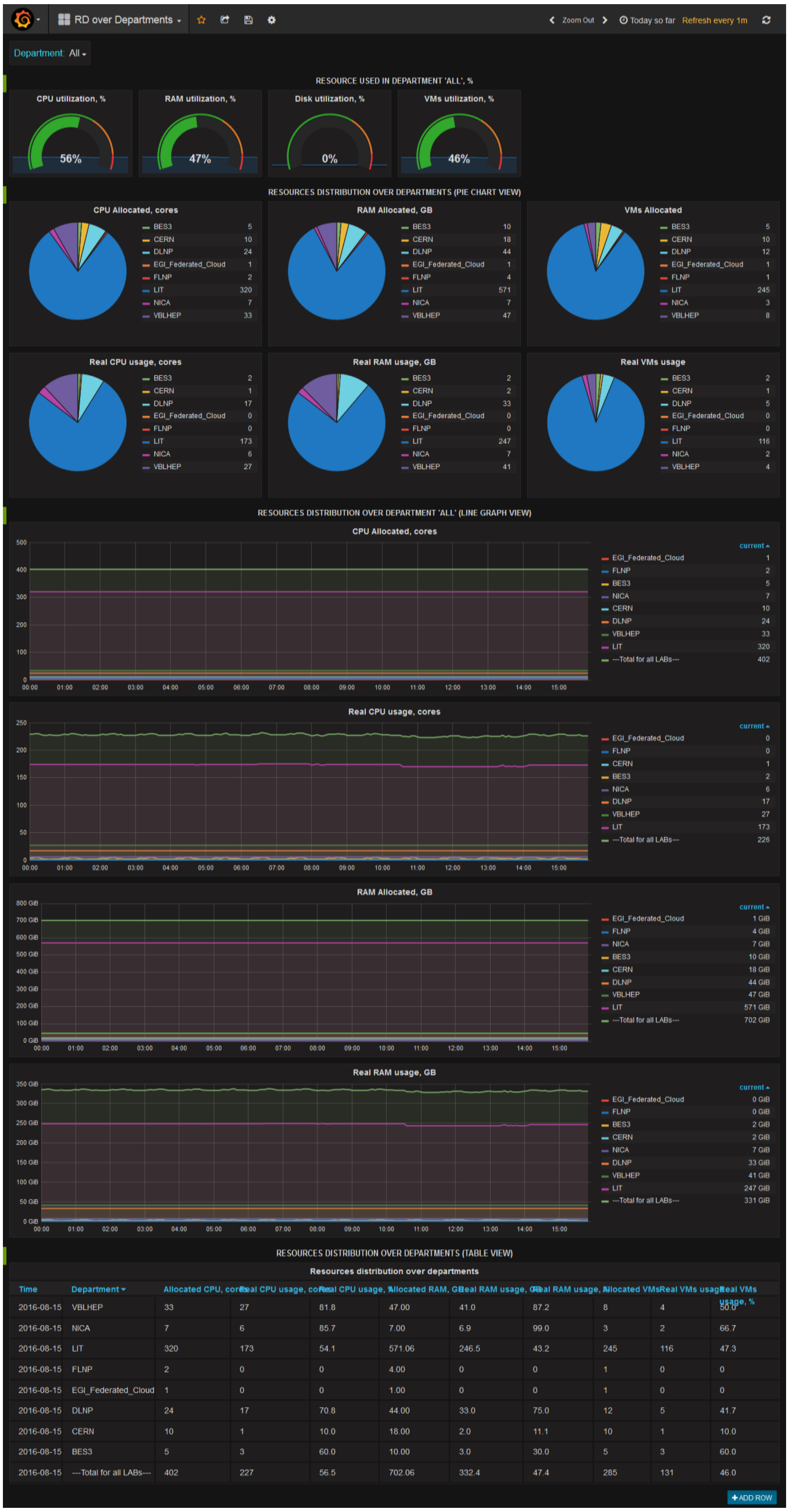
The developed dashboards



Resources distribution over clusters

Time	Cluster	Total CPU, cores	Allocated CPU, cores	Allocated CPU, %	Free CPU, cores	Free CPU, %	Real CPU usage, %	Total RAM, GB	Allocated RAM, GB	Allocated RAM, %	Free RAM, GB	Free RAM, %	Real RAM usage, %
2016-08-15	36-ovz-pub	48	71	148	-23	-48	4.1	65.50	101.00	154	-35.50	-54	28.4
2016-08-15	36-kvm-pub	12	8	67	4	33	9.5	23.04	12.00	52	11.04	48	41.2
2016-08-15	33-ovz-pub	52	56	108	-4	-8	2.6	148.51	91.43	62	57.09	38	12.1
2016-08-15	33-ovz-priv	32	25	78	7	22	1.3	61.22	36.00	59	25.22	41	10.4
2016-08-15	33-kvm-priv	24	18	75	6	25	1.8	46.08	20.50	44	25.58	56	28.6
2016-08-15	22x-ovz-NOvA	0	0	0	0	0	0	0	0	0	0	0	0
2016-08-15	22x-kvm-NOvA	100	1	1	99	99	0.1	234.51	0.50	0	234.01	100	1.3
2016-08-15	22x-openvz	56	29	52	27	48	0.1	251.06	51.00	20	200.06	80	1.5
2016-08-15	---Total for all clusters---	324	208	64	116	36	1.7	829.93	312.43	38	517.50	62	8.7

Figure C.1 – The dashboard of resource distribution over clusters



CPU Allocated, cores

BES3	5
CERN	10
DLNP	24
EGI_Federated_Cloud	1
FLNP	2
LIT	320
NICA	7
VBLHEP	33

RAM Allocated, GB

BES3	10
CERN	18
DLNP	44
EGI_Federated_Cloud	1
FLNP	4
LIT	571
NICA	7
VBLHEP	47

VMs Allocated

BES3	5
CERN	10
DLNP	12
EGI_Federated_Cloud	1
FLNP	1
LIT	245
NICA	3
VBLHEP	8

Real CPU usage, cores

BES3	2
CERN	1
DLNP	17
EGI_Federated_Cloud	0
FLNP	0
LIT	173
NICA	6
VBLHEP	27

Real RAM usage, GB

BES3	2
CERN	2
DLNP	33
EGI_Federated_Cloud	0
FLNP	0
LIT	247
NICA	7
VBLHEP	41

Real VMs usage

BES3	2
CERN	1
DLNP	5
EGI_Federated_Cloud	0
FLNP	0
LIT	116
NICA	2
VBLHEP	4

CPU Allocated, cores

	current
EGI_Federated_Cloud	1
FLNP	2
BES3	5
NICA	7
CERN	10
DLNP	24
VBLHEP	33
LIT	320
---Total for all LABs---	402

Real CPU usage, cores

	current
EGI_Federated_Cloud	0
FLNP	0
CERN	1
BES3	2
NICA	6
DLNP	17
VBLHEP	27
LIT	173
---Total for all LABs---	226

RAM Allocated, GB

	current
EGI_Federated_Cloud	1 GiB
FLNP	4 GiB
NICA	7 GiB
BES3	10 GiB
CERN	18 GiB
DLNP	44 GiB
VBLHEP	47 GiB
LIT	571 GiB
---Total for all LABs---	702 GiB

Real RAM usage, GB

	current
EGI_Federated_Cloud	0 GiB
FLNP	0 GiB
BES3	2 GiB
CERN	2 GiB
NICA	7 GiB
DLNP	33 GiB
VBLHEP	41 GiB
LIT	247 GiB
---Total for all LABs---	331 GiB

Resources distribution over departments

Time	Department	Allocated CPU, cores	Real CPU usage, cores	Real CPU usage, %	Allocated RAM, GB	Real RAM usage, GB	Real RAM usage, %	Allocated VMs	Real VMs usage	Real VMs usage, %
2016-08-15	VBLHEP	33	27	81.8	47.00	41.0	87.2	8	4	50.0
2016-08-15	NICA	7	6	85.7	7.00	6.9	99.0	3	2	66.7
2016-08-15	LIT	320	173	54.1	571.06	246.5	43.2	245	116	47.3
2016-08-15	FLNP	2	0	0	4.00	0	0	1	0	0
2016-08-15	EGI_Federated_Cloud	1	0	0	1.00	0	0	1	0	0
2016-08-15	DLNP	24	17	70.8	44.00	33.0	75.0	12	5	41.7
2016-08-15	CERN	10	1	10.0	18.00	2.0	11.1	10	1	10.0
2016-08-15	BES3	5	3	60.0	10.00	3.0	30.0	5	3	60.0
2016-08-15	---Total for all LABs---	402	227	56.5	702.06	332.4	47.4	285	131	46.0

Figure C.2 – The dashboard of resource distribution over departments