

# MASTERARBEIT

**Titel der Masterarbeit**

Functional user interfaces for controlling and  
monitoring the N2Sky and its services

**Angestrebter akademischer Grad**  
Master of Science

Wien, 2018

|  |  |
|--|--|
| <b>Verfasst von:</b>                     | Andrii Fedorenko, Bakk<br>MatNr. 01349252        |
| <b>Studienkennzahl lt. Studienblatt:</b> | A 066 926  |
| <b>Studienrichtung lt. Studienblatt:</b> | Masterstudium Wirtschaftsinformatik              |
| <b>Betreuer:</b>                         | Univ.-Prof. Dipl.-Ing. Dr. techn. Erich Schikuta |

## **Zusammenfassung**

## **Abstract**

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                           | <b>6</b>  |
| 1.1      | Motivation . . . . .                          | 6         |
| 1.2      | Terms and definitions . . . . .               | 6         |
| 1.3      | Related work . . . . .                        | 6         |
| <b>2</b> | <b>N2Sky Architecture</b>                     | <b>6</b>  |
| 2.1      | Current Architecture Analysis . . . . .       | 6         |
| 2.1.1    | Architecture design . . . . .                 | 6         |
| 2.1.2    | Components . . . . .                          | 8         |
| 2.1.3    | Implementation . . . . .                      | 8         |
| 2.1.4    | Usability and user experience . . . . .       | 9         |
| 2.2      | Redesign motivation . . . . .                 | 9         |
| 2.2.1    | Redesign Process . . . . .                    | 9         |
| 2.2.2    | Refactoring the User Interface . . . . .      | 12        |
| 2.2.3    | Advanced project management . . . . .         | 13        |
| 2.2.4    | User Roles . . . . .                          | 14        |
| 2.2.5    | User Main Functions . . . . .                 | 16        |
| 2.3      | Novel N2Sky Architecture . . . . .            | 17        |
| 2.3.1    | Cloud infrastructure . . . . .                | 17        |
| 2.3.2    | Modular frontend application design . . . . . | 19        |
| 2.3.3    | The sample workflow . . . . .                 | 22        |
| 2.3.4    | Technology Stack . . . . .                    | 25        |
| <b>3</b> | <b>N2Sky Components</b>                       | <b>26</b> |
| 3.1      | N2Sky Frontend Application . . . . .          | 26        |
| 3.1.1    | N2Sky Layouts Design . . . . .                | 27        |
| 3.1.2    | N2Sky Fundamental Layout . . . . .            | 28        |
| 3.1.3    | N2Sky Main Layout . . . . .                   | 29        |
| 3.1.4    | N2Sky Mobile Layout . . . . .                 | 30        |
| 3.1.5    | N2Sky Page Views . . . . .                    | 30        |
| 3.1.6    | N2Sky Dashboards . . . . .                    | 33        |
| 3.1.7    | Responsive design . . . . .                   | 33        |
| 3.1.8    | User Interface Elements . . . . .             | 36        |
| 3.1.9    | UI Elements in N2Sky . . . . .                | 37        |
| 3.1.10   | UI Components in N2Sky . . . . .              | 40        |
| 3.2      | N2Sky Services . . . . .                      | 44        |
| 3.2.1    | User Management Web Service. . . . .          | 44        |
| 3.2.2    | Model Repository Web Service. . . . .         | 45        |
| 3.2.3    | Cloud Management Web Service. . . . .         | 48        |

|          |   |           |
|----------|---|-----------|
| 3.3      | Continuous Monitoring System . . . . .                        | 48        |
| 3.3.1    | Monitoring requirements . . . . .                             | 50        |
| 3.3.2    | Applying Monitoring . . . . .                                 | 51        |
| 3.3.3    | Integration with N2Sky . . . . .                              | 54        |
| 3.3.4    | Monitoring dashlet Design . . . . .                           | 55        |
| 3.4      | Alerting Management System . . . . .                          | 56        |
| 3.4.1    | Alerting System Architecture . . . . .                        | 56        |
| 3.4.2    | Alert Rules . . . . .   | 58        |
| 3.4.3    | Integration with N2Sky . . . . .                              | 59        |
| 3.4.4    | Alerting System Design . . . . .                              | 59        |
| <b>4</b> | <b>ViNNSL 2.0 extended</b>                                    | <b>61</b> |
| <b>5</b> | <b>Requirements specification for Administration Module</b>   | <b>61</b> |
| 5.1      | General Definition . . . . .                                  | 61        |
| 5.2      | Affected users . . . . .                                      | 61        |
| 5.3      | Administration Dashboard . . . . .                            | 62        |
| 5.4      | Openstack Dashboard . . . . .                                 | 63        |
| 5.4.1    | OpenStack Nova Service . . . . .                              | 65        |
| 5.4.2    | OpenStack Neutron Service . . . . .                           | 67        |
| 5.4.3    | OpenStack Images Service . . . . .                            | 70        |
| 5.4.4    | OpenStack Vitrage Service . . . . .                           | 71        |
| 5.5      | Cloudify Dashboard . . . . .                                  | 75        |
| 5.6      | Dashboard Settings . . . . .                                  | 75        |
| 5.7      | Monitoring System . . . . .                                   | 75        |
| 5.7.1    | Monitoring charts creation . . . . .                          | 76        |
| 5.7.2    | Monitoring charts representation . . . . .                    | 77        |
| 5.8      | Alert System . . . . .  | 78        |
| 5.8.1    | Alerting rule creation . . . . .                              | 78        |
| 5.8.2    | Alerts representation . . . . .                               | 80        |
| <b>6</b> | <b>Requirements specification for Main Application Module</b> | <b>80</b> |
| 6.1      | General Definition . . . . .                                  | 80        |
| 6.2      | Affected users . . . . .                                      | 80        |
| 6.3      | N2Sky Dashboard . . . . .                                     | 81        |
| 6.4      | Neural Networks Repository . . . . .                          | 81        |
| 6.5      | Models Repository . . . . .                                   | 81        |
| <b>7</b> | <b>Tutorial</b>   | <b>81</b> |
| <b>8</b> | <b>User Cases</b>   | <b>81</b> |

---

|   |           |
|---|-----------|
| <b>9 Developer Guide</b>                                  | <b>81</b> |
| 9.1 System configuration . . . . .                        | 81        |
| 9.1.1 Setting up the database . . . . .                   | 81        |
| 9.1.2 Setting up the Cloud Web Service . . . . .          | 82        |
| 9.1.3 Setting up the Model Repository Service . . . . .   | 82        |
| 9.1.4 Setting up the User Management Service . . . . .    | 83        |
| 9.1.5 Setting up the N2Sky frontend . . . . .             | 83        |
| 9.1.6 Setting up the Monitoring System . . . . .          | 83        |
| 9.1.7 Setting up Alert Management System . . . . .        | 84        |
| 9.2 Continuous integration . . . . .                      | 86        |
| 9.3 API Documentation . . . . .                           | 86        |
| 9.3.1 N2Sky Monitoring System API Documentation . . . . . | 86        |
| <b>Literaturverzeichnis</b>                               | <b>87</b> |
| <b>Anhang</b>   | <b>89</b> |

# 1 Introduction

## 1.1 Motivation

## 1.2 Terms and definitions

## 1.3 Related work

# 2 N2Sky Architecture

## 2.1 Current Architecture Analysis

Current N2Sky architecture a monolithic standalone application. It was heavily maintainable and not scalable. Before proposing the new architecture design it is important to find the weak parts of the current application and define what functionality is still possible to reuse in order gain new functionality with a reasonable time.

### 2.1.1 Architecture design

The current N2Sky design based on RAVO architecture [8], which extends default SPI stack. SPI stack contains Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The architecture had two phases of integration. In the first phase, the SPI was extended and applied to the N2Sky application. On the last phase, all mandatory components were implemented and integrated as it presented in figure 1.

Every service represents the particular layer:

- *IaaS layer* responsible for managing the resources. It contains the logical and physical resources of N2Sky components as well as infrastructure enabler like protocols, procures etc.
- *PaaS layer*, which provides an access to resources via API. In the previous N2Sky system, it served as a domain-independent tool. It also contains the neural network layer.
- *SaaS* is a service, which container the User interfaces (UIs).

Everything as a service is a good idea but build upon it the layered architecture could cause some issues. Since every layer depending on the layer above, there a possibility if the services from bottom layer go then the application is not usable at all.

Besides stand-alone-application design, which was implemented, another one issue is encapsulation. Since every service needs to communicate with other services via API the order of layers could be violated. This problem can cause not only security issues, bus also consistency, namely it can break the desired workflow process.

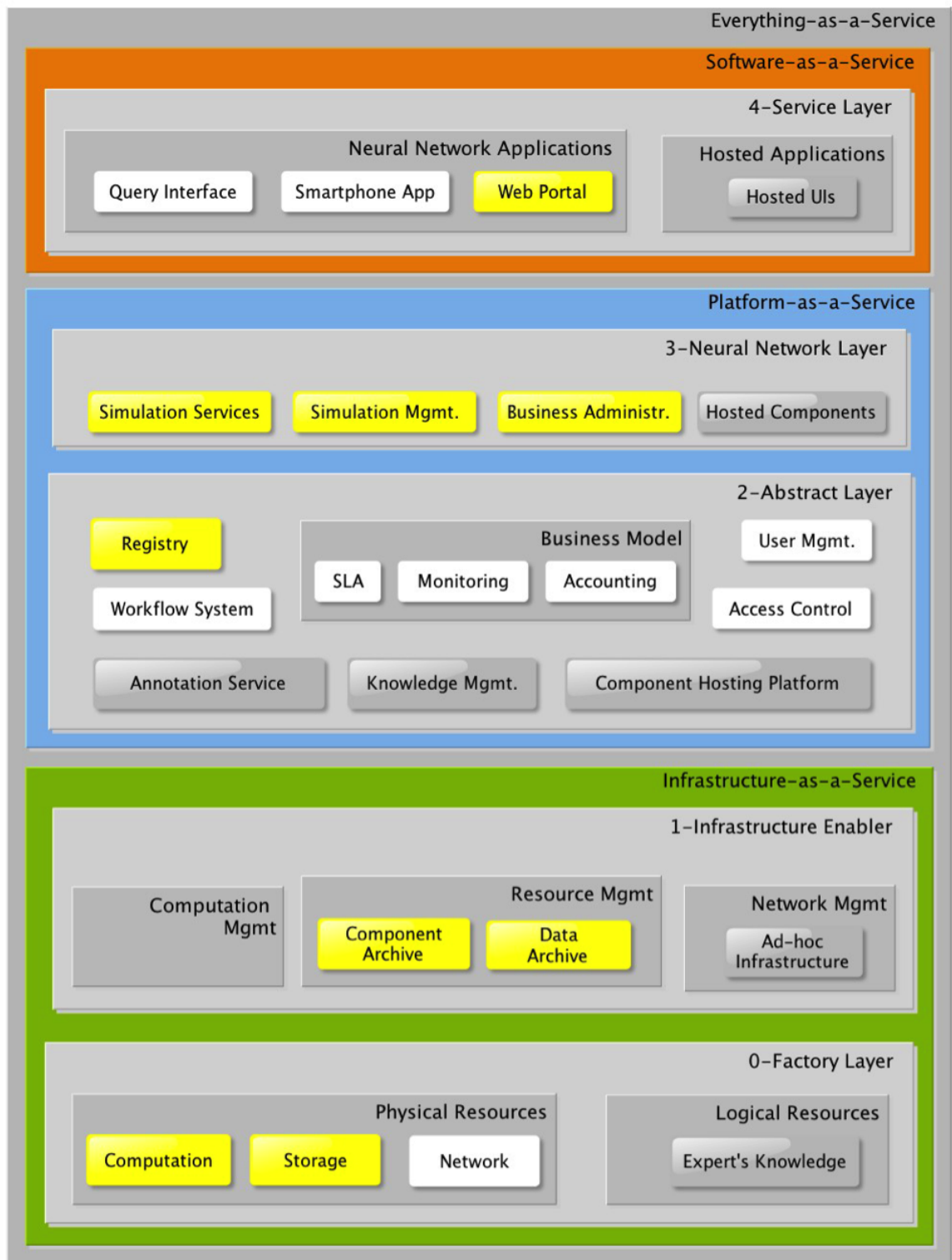


Figure 1: Current N2Sky architecture

Unfortunately, it is impossible to distribute the services, since they are all coupled together into collections of layers.

### 2.1.2 Components

N2Sky was an imposing application, which contained multiple services and was working as a whole. Following main components were listed in N2Sky:

- *The component archive* was responsible for storage and file management.
- *The data archive* is the component, which managed the neural network specific resources in XML and JSON formats.
- *The registry*, which was a base component of the N2Sky architecture. This component was responsible for searching for services within the cloud. It also contained instructions and rules, which were used by users.
- *Monitoring* was divided into two subcategories: business activity monitoring (BAM), which gathered the business data and technical monitoring, which was responsible for environmental monitoring and performance statistics.

Considering that the registry is a central component, because of it responsible for services management, it can slow down the whole process. All requests are going through this services. If the services are not accessible or high latency or waiting time occurs, the UI as well as API services would be not available.

It would be great to have the same service but distribute them on multiple servers. Every service in the previous version of N2Sky is gigantic, it is possible to divide services into microservices and encapsulate sensible components from public access.

### 2.1.3 Implementation

The previous version of N2Sky is fully written in Java using Spring framework and Java servlets. The Tomcat server was used, because of the Java programming language. The Java Spring framework is good for an enterprise application, but the N2Sky needs lots of computations so the high-end supercomputers are necessary pre-requirement in this case.

Unfortunately, it is impossible to map the current architecture into the microservices approach. Only one solution is to make replicas of the application, but then the sessions and data consistency have to be tracked in every replica.

The mobile application of the N2Sky is built with raw HTML5 and JavaScript. Today, looking on this mobile application it is close to impossible to maintain it. Nevertheless, the mobile version is also wrapped into layered architecture and in implementation, it is still one project. The solution to this problem is the responsive design. The responsive



design could be fully adaptive to multiple devices from desktop PCs to mobile with a small screen resolution.

The whole system is deployed on Amazon Web Services (AWS), which is good if the owner has enough resource to keep it working. The best solution would be to have a dedicated server with a real physical memory and computational abilities.

### 2.1.4 Usability and user experience

## 2.2 Redesign motivation

Application redesign is a project, which takes a lot of work. But at some point every designer faced a refactoring project. It has a lot to do with user experience. Bad user experience will make users stop use an application and leave negative feedback on application in general.

### 2.2.1 Redesign Process

There is data, information and user experience of previous version of N2Sky to work with. Making redesign it is already known who the users are and what they are trying to achieve. Using this information it is possible to build an aims for a future user interface and user experience.

**Finding problems.** There are multiple problems with the application. One of the most crucial is that user interface is not intuitive understandable.



Figure 2: Current N2Sky User Interface

After signing in user getting subscription form, paradigm service and paradigm metadata views without any field description. Small titles unfortunately not always self-describing. Application in general oriented on the group of users, who are came from IT area. In some forms they can type queries, but the fields are not type safe and there is no autocomplete. Representation of neural networks trained model is not readable. The model represented as a raw JSON or XML file, user can not download it. The last point is a design in general that does not look up-to-date and user attractive.

**User interviews and Questionnaire.** User insights are very important. It helps to understand a nature of the problem. But if user find something confusing, the interviewer need to dig deeper to stress the importance of particular insight. Unfortunately there were no analytics data and any reviews regarding UI and UX, so with a small group of colleges the current UI of N2Sky was reviewed. The following questions were derived (Q1-Q5):

- Q1: "How N2Sky can help you with a developing of your neural network?"
- Q2: "What was the most difficult part by creating a new model?"
- Q3: "Did you face any problems during spawning your neural network? If yes, then what kind?"
- Q4: "Did you find out something new, when other users were performing testing against your neural network?"
- Q5: "What did you miss during using an N2Sky?"

There were 5 students interviewed. All students were together in one group. Summarise answers (A1-A5) according to questions (Q1-Q5):

- A1: N2Sky gives possibility to test the own neural network. Unfortunately, if the user does not have a neural network it is impossible to test the application.
- A2: User face difficulties during creation of the new neural network model. User interface user technical jargon and it is not intuitively understandable.
- A3: Spawning a neural network was pretty clear process, but it was not really clear if neural network ready to use or not.
- A4: Logging of the training data is very useful. The neural network owner can see how his network behaves with a different training and testing data.
- A5: User wants perform testing and training on already existing neural networks.

**Current application design mapping.** After studying the answers it is possible to highlight weak parts of application. This approach will show a big picture of the current application design:

- Arbitrary user needs to know multiple technologies and programming languages just simply to reuse existing neural network.
- Too much information on every view. The purpose of view is overloaded. Each view has too much functionality, which makes user to lose a focus.
- Application works relatively slow. Even if there are some processes behind happening, the user does not know it.

Important is to face the problems, but does not "reinvent the wheel". As **Joel Spolsky** the founder of Netscape and CEO of Stack Overflow said, "throwing away the whole program is a dangerous folly". That is why it was decided to consider the problems of current N2Sky design and reuse working ideas in refactored system

**Application Maintenance.** N2Sky was monolithic standalone application, which included all services in one and was deployed as a whole. The application was not distributed. In case if one of the services doesn't work correct, the whole application is not usable. Originally the previous version of N2Sky was written fully on Java. There were hundreds classes, providers and services in one project. Developer will spend hours to maintain this kind of project. To find an issue in a big application is always a challenge. Small changes are causes a subsequence changes. If the software breaks after change, then it will additional high effort to fixed it. As Robert Cecil Martin wrote in his book "Clean Code": "The code is hard to understand. Therefore, any change takes additional time to first reengineer the code and is more likely to result in defects due to not understanding the side effects?" [<https://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882>]. He categorizes this kind of code into "Smell" code. Unfortunately N2Sky from maintain perspective had all problems, which Mr. Martin

That is why N2Sky is shifted from monolithic system to container based system with an independent micro services which located on cloud environment. The frontend and frontend services are lightweight and easy to maintain parts of the big application. If something goes wrong, the developer knows exactly where is the problem. It is close to impossible to break something else during fixing because of independence of services. Additionally there are monitoring and alerting systems which are supporting developers during maintenance. Early it was not possible to say if application works correctly or even it still running. Users could get a bad experience while they using an application in case if it does not work correctly.

But now it is possible not only notify an application administrator about some problems, but also to predict potential threads.

### 2.2.2 Refactoring the User Interface

User Interface (UI), an abbreviation of user interface, allows the interaction of a user with a program through graphical visualization made by text, icons, buttons and pictures. While deciding the design of a user interface there are some highly recommended features also known as heuristics, which was invented by Jakob Nielsen. It was decided to apply the following 10 general principals of interaction design to N2Sky:

**Simple and natural dialogue.** N2Sky will have an simple UI which is understandable for any user, even if the user is not an expert. Every icon and every navigation or action button will be self-describing. The application will follow the slogan "less is more". No more overloaded views. The idea of N2Sky UI design is that one particular view is responsible for one particular function or group of functions which are coupled tight together.

**Speak the users language.** Developing N2Sky was concentrated on user perspective. There is no technical jargon for arbitrary user.

**Minimize user memory load.** There is no multiple options, functions or menus on one view. There is also no multiple ways to do the same thing. N2Sky teach user how to make things done with a one existing and convenient workflow.

**Consistency.** N2Sky has similar layouts, fonts, colors, icons types structures and organization throw entire application. The user should get the same visual experience on every view.

**Feedback.** Every action, process or even error will be notified. User will know exactly what is happening with a system with clear and understandable messages.

**Clearly marked exits.** Every push to action button has short and clear caption.

**Shortcuts.** N2Sky has multiple user types. One of the types is the expert users, which are advanced user in neural network and artificial intelligence topics. For this kind of user is provided more technical jargon, but this UI is separated with an arbitrary users.

**Good error messages.** Every notification is clear inclusive an error message if occurs. Every message has a prices and simple description.

**Prevent errors.** In N2Sky implemented logical structure of UI components. There are constraints which are helps user in workflow. For example user will always get a default value of any input.

**Help and documentation.** N2Sky will have tutorials that describe the user workflow.

The expert users will also get a API documentation with a detailed description and sample requests.

Each and every one of these heuristics is connected to a crucial idea that is usability. By mentioning this idea, a straightforward relation with the UX follows since this is also one of the key concept that grows along the rapid development of technology. UX is known as user experience and it describes the perspective and feelings a user gets when interacting with product. It deepens into such aspect as the users inner circumstances and the nature of the created design. The goal is to achieve such a system that offers distinguished user experience and accomplishes the most of aspects. As above mentioned, usability. (1) Putting both of UI and UX in a comparison, to all appearances the user interface is the target on the appearance and functionality of the product and its tangible details. Furthermore, the user experience is the general experience that the user manages throughout the whole use. (2) UI will concentrate on the appearance and design of the product, rather than the functionality. The intent of it relies in the visual design and layout. UI covers issues such as how a button is supposed to look like, how the errors are going to appear or is it visually comprehensible meaning which colors or font type shall be used for a better perceptibility of the product. (6) UX points its focus on the involvement of the user while interacting. It is measured by a variety of tests and researches done to achieve a higher satisfaction on the users side. (4) Though their differences, the only matter which relates both UI and UX is their priority, in other words, the user. When expanding the concept of both these definitions it can be concluded that one co-exists with the other. There would not be user interface without user experience and vice versa.

TODO // CHECK MERISI DOKU

### 2.2.3 Advanced project management

In order to make N2Sky competitive on the market, the modern approach in planning of the project has to be done. Today the most common and effective way in development and planning is well-constructed Functional Requirements Specification (FRS), which helps to estimate effort and advice in the future planning.

FRS is a document that defines functionality, which application or some parts of application must perform [1]. N2Sky is sociotechnical system, meaning that is strongly interacts with humans.

FRS for N2Sky is described in natural language with formal methods in order to establish specifications between development process and end-user consuming. Ever N2Sky module has FRS, which is explicit and points on system functionality. A good FRS must be unambiguous, consistent and correct [7].

The formal or semi-formal methods are serving for analysing and validating FRS. The main purpose of that to limit interpretation errors. The problem occurs when FRS is fully written in natural language, when designers does not have required technical knowledge in order to use other languages. One of the typical solution is defects detection technics, which require some effort from designer [24].

In N2Sky some methods and technics were applied in order to make specification clean and clear as possible. Despite that in FRS will be always some parts which are leading to misunderstanding [7].

FRS is a part of engineering phase. Following qualities for N2Sky were defined and applied during this phase [9] as it shown in “Fig. 3” :

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

#### 2.2.4 User Roles

In order to make the N2Sky user interface understandable for arbitrary users as well as professional for advances users, it was decided to separate the user roles. Every user role has own way of interaction with the application.

Every user has some specific area within he works. For example, just registered user does not need to know the current environment monitoring information. These restrictions were motivation to create some user roles in order to restrict of grand some functionality of N2Sky as it shown on “Fig. 4”.

- *Arbitrary User.* The Arbitrary User a user, who does not have a deep knowledge of the neural network field or know any programming language. His main purpose is not a contribution, but the usage already existing neural networks and trained models. The main goal of the arbitrary user is to study neural networks within N2Sky platform. The arbitrary user can also evaluate the trained models or execute training against existing neural networks. This kind of user does not have to use his own training data, he just wants to see the behavior of neural



Figure 3: Applied on N2Sky Requirement Specification Qualities

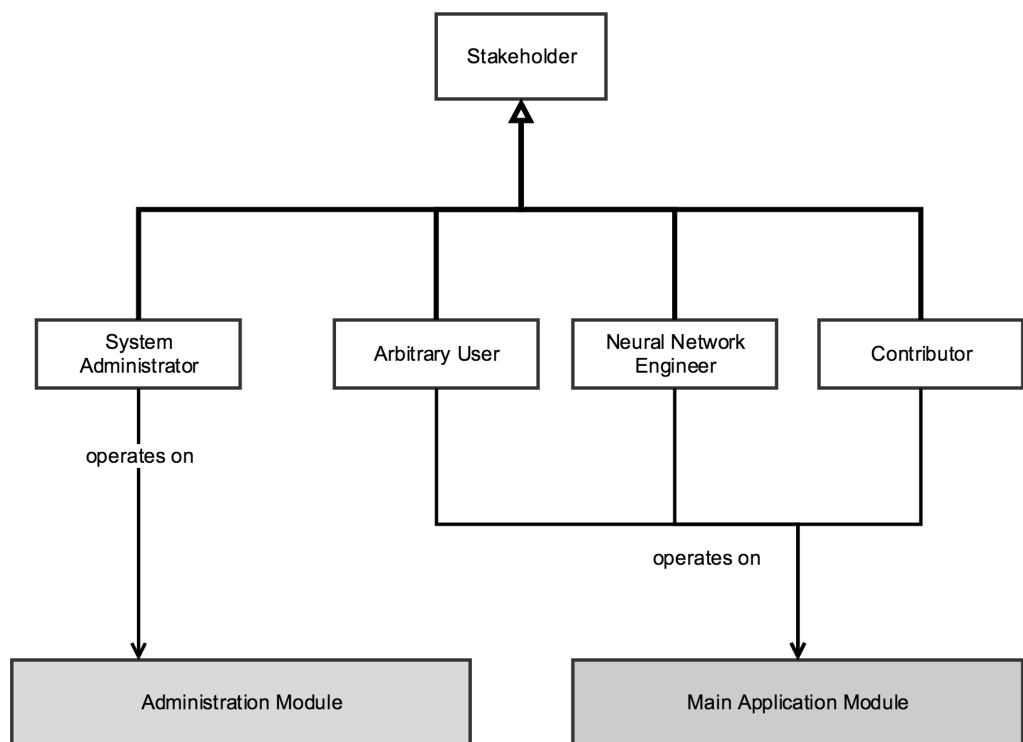


Figure 4: User roles hierarchy and modules where they operating on (marked grey)

| User Role            | Administration Module |                     | N2Sky Main Application Controller |                           |
|----------------------|-----------------------|---------------------|-----------------------------------|---------------------------|
|                      | OpenStack Management  | Cloudify Management | Neural Networks Management        | Trained Models Management |
| System Administrator | +                     | +                   | +                                 | +                         |
| Moderator            | -                     | -                   | +                                 | +                         |
| Consumer             | -                     | -                   | -                                 | -                         |
| Developer            | -                     | -                   | -                                 | -                         |
| Contributor          | -                     | -                   | -                                 | -                         |

Table 1: User Roles main functions considering "Administration Module" and "N2Sky Main Application Controller". "+" for allowed, "-" for disallowed

networks. The arbitrary user can be converted to Neural Network Engineer user if he has enough knowledge of it.

- *Neural Network Engineer.* The Neural Network Engineer is an arbitrary user. The Neural Network Engineer has an access only to his own dashboard and publicly available resources on the main application module. He can perform the semantic search for available neural network paradigms and use them. This user can create own neural network instance from existing neural network paradigm. He can also train the running neural network instances and evaluate their trained models. This user can share his trained neural network by making it public.
- *Contributor.* The Contributor is an expert user, which has enough knowledge and experience to create his own neural network. This user can create neural network paradigms using the ViNNsL template schema and publish them on N2Sky. This user can deploy neural networks on the N2Sky environment as well as on his own environment by providing training and testing endpoints. The goal of the contributor is the study how his networks will behave with different network structures, input parameters and training data that is provided by other users.
- *System Administrator.* System Administrator is a user who has a full access to the application including environment management, monitoring and alerting features. The administrator can manage OpenStack and Cloudify instances. He also can shadow any N2Sky user to observe the application from other perspectives. The administrator has access to all dashboards in every module.

### 2.2.5 User Main Functions

In more detailed overview of user roles it is possible to define permission and main functions. Permissions describes users allowed page view. If user has access to particular page view the main user function can be defined. The main user function characterise allowed behaviour on particular page view.

As it was mentioned in previous chapter, N2Sky is an modular application. Permissions and main functions of Administration Module and N2Sky Main Application Controller, which is a part of Main Application Modules, described in Table 1.



| User Role            | N2Sky Main Application Module |                          |                         |                            |
|----------------------|-------------------------------|--------------------------|-------------------------|----------------------------|
|                      | Paradigm Creation             | Neural Networks Creation | Neural Network Training | Training Models Evalutaion |
| System Administrator | -                             | -                        | -                       | -                          |
| Moderator            | -                             | -                        | -                       | -                          |
| Consumer             | -                             | -                        | +                       | +                          |
| Developer            | +                             | +                        | +                       | +                          |
| Contributor          | -                             | +                        | +                       | +                          |

Table 2: User Roles main functions considering "N2Sky Main Application Module". "+" for allowed, "-" for disallowed

System Administrator has permissions to all components. Moderator has access only to N2Sky Main Application Controller. Since both of this user roles extending Contributor user role, the permissions for main application module are also granted.

Consumer, Developer and Contributor user roles have no access to any administration parts of N2Sky. This users do not have any main function in this area, but they can operate N2Sky Main Application Module as it shown on Table 2.

As it was mentioned before System Administrator as well as Moderator has access to N2Sky Main Application Module, but they do not have a main function there. On the other hand all user roles, which are extending consumer can contribute in a N2Sky Main Application Module, except Consumer itself. Consumer has access to all page views on this module, but he has another purpose.

## 2.3 Novel N2Sky Architecture

N2Sky is a simulation platform, which allows all involved stockholders to use it for computational purposes. In order to achieve high performance and scalability, it was decided to use the microservices architecture. This approach is used not only for back-end services but also with frontend and its services.

The user-centered design is a fundamental requirement for N2Sky. Looking back on past experiences with the application, there were identified the real capabilities and needs of users. N2Sky was moved from a complex monolithic system to an easily understandable application. Every interested user without having a deep knowledge in the neural network field can freely use N2Sky. The goal was to save and gain the current functionality of the application and decrease the visual complexity of it.

### 2.3.1 Cloud infrastructure

Upon the N2Sky microservices the the load balancer located, which includes shared cloud resources. Since in N2Sky environment needs to support thousands of Docker containers [15] simultaneously, only container orchestration like Cloudify [21] with a microservices architecture can perform this. Cloudify is a container orchestration tool, which provides communication between cloud platforms and manages container deployment and execution. Docker container is a operating-system-level virtualization.

Containers technology used across entire system from frontend and services to monitoring and alert management system.

The novel N2Sky architecture can be scaled horizontally as well as vertically on demand as it presented in figure 8.

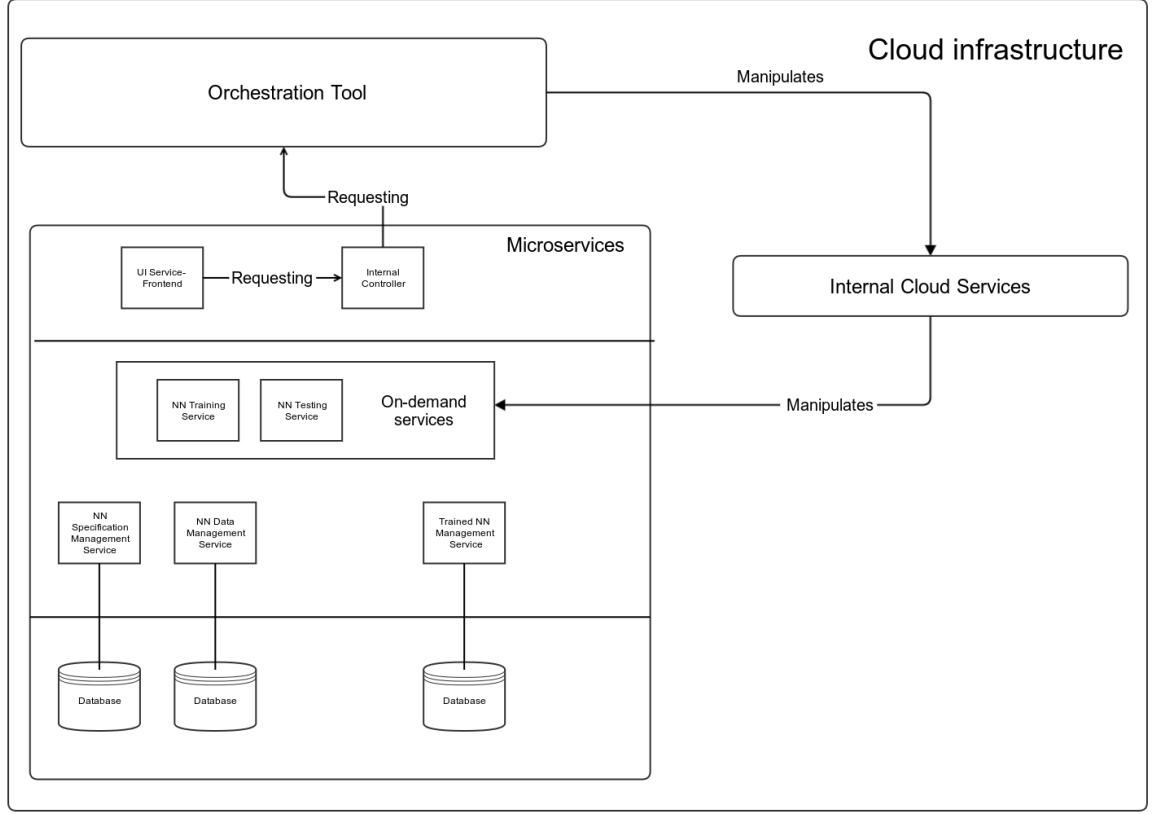


Figure 5: Novel N2Sky Architecture

- *Orchestration Tool* is the Cloudify framework, which communicates with a cloud infrastructure namely create and manage new instances. Every instance is a Docker container.
- *Internal Controller* is a service, which works directly with the Cloudify tool. This service handles user request in order to create, delete or update Docker container.
- *UI Service-Frontend* is a modular application namely it is the Administration Module. The user over UI can request container management, which will be redirected to Internal Controller.
- *On-demand services* are not exposed publicly services, which can be controlled only by internal web services. This approach helps to encapsulate the services pool, which increases security in the entire application.

- *Neural Network Specification Management Service* is a service, which is responsible for controlling existing neural network paradigms, namely train, retrain and persist trained models.
- *Neural Network Data Management Service*, this service located right on top of the database and provides API for adding external data sources.
- *Trained Neural Network Management Service* is a service, which responsible for evaluating trained models.

### 2.3.2 Modular frontend application design

Maintain large application like N2Sky with the monolithic approach is unwieldy. Since N2Sky supports microservices approach in the backend it was decided to apply the same solution on the frontend.

Microservices in frontend are small independent web applications, which are consolidated into one application. The main benefits of this approach are:

- *Maintainability*. It is possible to divide application between different teams. Developers do not even need to have some knowledge about other parts of the application.
- *Diversity of technologies*. Monolithic approach makes the whole application stick to one framework. Microservices allowing to use any technology without the need to rewriting the application.
- *Independent deployment*. Every application has some releases periods, every release accompanies with redeployment procedure. There is no need to redeploy the whole application, but just only required components.

Microservices approach in frontend application is shown in “Fig. 6”. This breaks the entire application into small micro applications.

- *Shell*. is a top-level component, which wraps Components picker and Container for the component. It contains application configuration.
- *Component picker*. Is a router, which manages the micro applications.
- *Container for Component*. Container, where the component will be injected.
- *Micro Application*. Independent application, which can be written in any programming language but has to use one of the shards.
- *Shards*. Is a code base, which is shared between Micro Applications. Shards can have multiple levels.

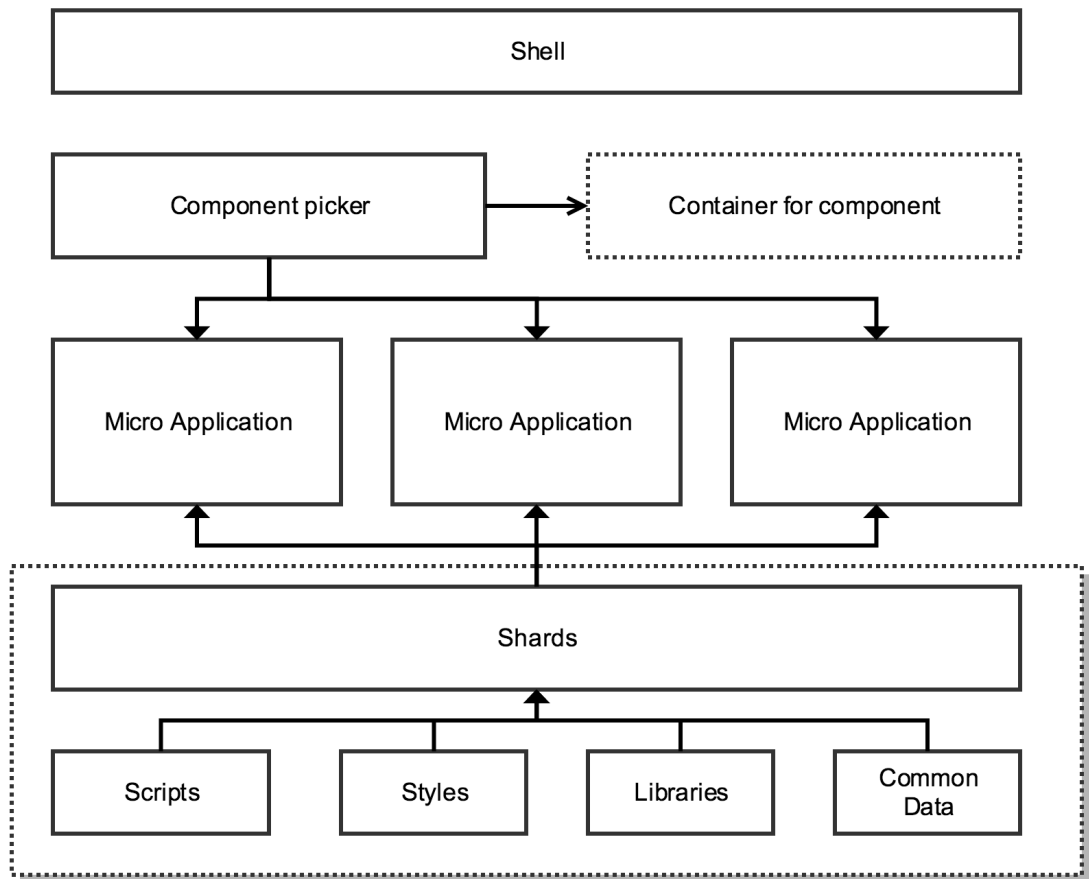


Figure 6: Microservices approach in frontend application

The central concept of the application is to support the Software as a Service (SaaS) and Platform as a Service (PaaS) distributions [23]. N2Sky consists of two modules: administration module, main application module as it shown in “Fig. 7”.

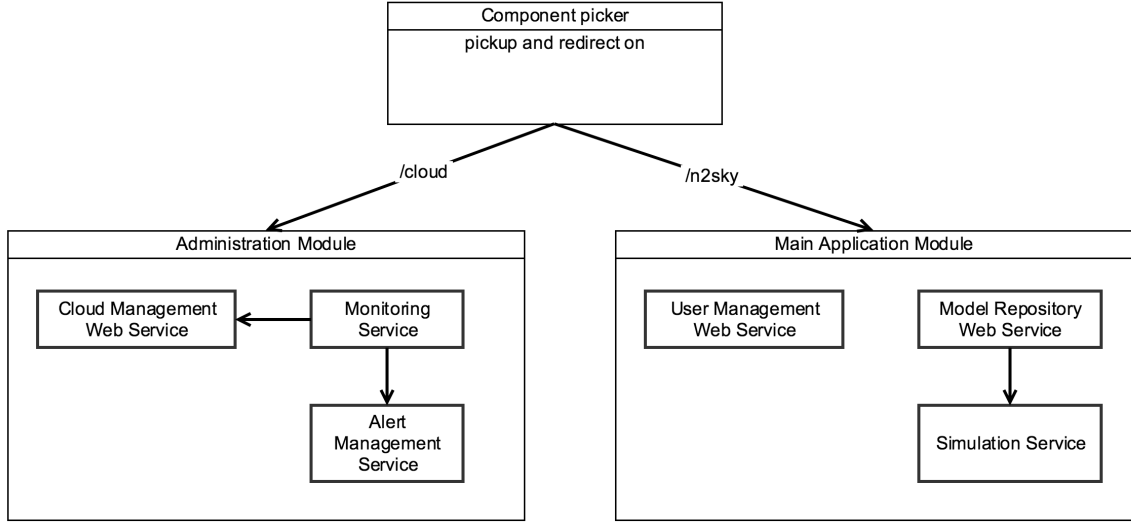


Figure 7: N2Sky frontend application and its services

- *N2Sky component picker.* When the user goes to N2Sky web portal, first he will be dealing with a component picker. The component picker is a small service, which redirects the user depending on URL path.
  - */cloud* redirects to "Administration module"
  - */n2sky* redirects to "Main application module"
- *Administration module* The administration module allows the system administrator to control the environment. The module supports OpenStack and Cloudify monitoring. Managing is possible through the application dashboard. It also contains custom monitoring and an alerting management system, which can be installed on any server within the N2Sky user interface. The administration module implements PaaS. It is fully configurable and wrapped into the open source project in order to make the module accessible to the third-party applications.
- *Main application module* The main application module is the central module of N2Sky. Within this module, users can use, train and test existing neural networks. It is possible to reuse the neural network paradigms and create own neural network. N2Sky allows deploying own network and store data in the cloud. Module services are supporting the SaaS distribution. Experts can use an application directly through the N2Sky API or they can integrate N2Sky services into their own application.

### 2.3.3 The sample workflow

The central figure is the N2Sky Web/Mobile portal. Is is frontend application of the N2Sky, which consist of modular subsystems. Since the frontend application has responsive design it supports desktop devices as well as mobile devices. To support Software as a Service distribution every web service can work independently. It means that the stakeholders can use N2Sky via web portal as well as use N2Sky API. N2Sky API allows stakeholders:

- Authorise in the System
- Create new neural from existing paradigm
- Deploy own neural network on N2Sky environment
- Perform training against own as well published neural network
- Perform testing against trained models

Almost everything is available for arbitrary stakeholders, except cloud services. In order to cloud service as a service, the user has to install it on his own cloud environment. This approach supports Platform as a Service distribution. Cloud services only for system administrator and for granted users are available.

The sample workflow overview, which shown in “Fig. 8” represents microservices architecture in action:

#### 1. *Contributor User*

- a) The Contributor user authorize in the N2Sky portal using his browser on desktop PC or mobile device. He will be redirected to his own dashboard according to permissions, which will be received from User Management Web Service.
- b) The user described his own neural network paradigm using the ViNNsL template and deploying it in the N2Sky cloud.
- c) The Contributor user perform training of his neural network using the N2Sky platform. Since the user is an expert he can perform this operation using Simulation Service via Model Repository Web Service API.
- d) The user publishes his paradigm via N2Sky UI or available API.
- e) The Contributor user awaiting until other N2Sky users will use his neural network paradigm in order to monitor the behavior of the neural network.
- f) The user modify, redeploy and retrain his neural network after first results.

#### 2. *Neural Network Engineer User*

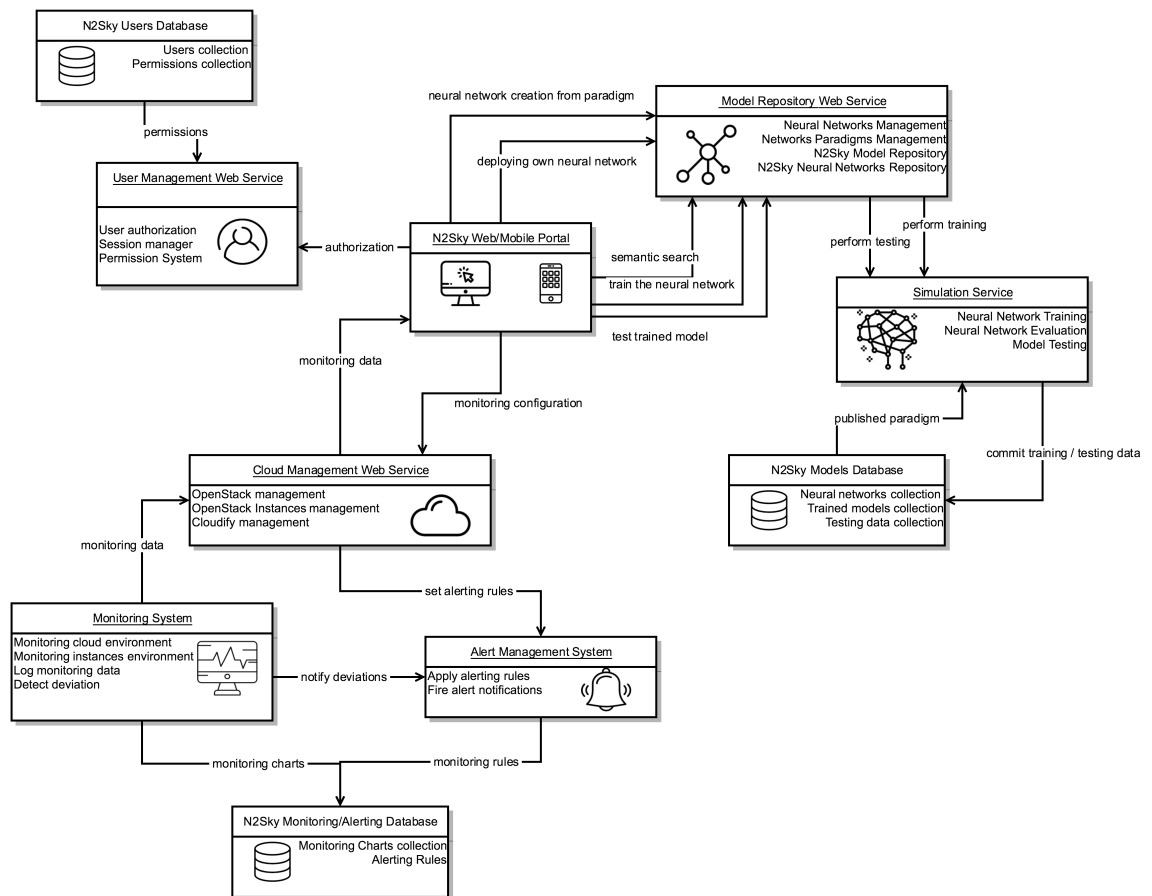


Figure 8: The sample workflow

- a) The neural network engineer user authorize in the N2Sky portal using his browser on desktop PC or mobile device. He will be redirected to his own dashboard according to permissions, which will be received from User Management Web Service.
- b) From the dashboard, the user creates a neural network from existing paradigm using Model Repository Web Service.
- c) The user perform training against his newly created neural network using the N2Sky platform.
- d) If the user satisfied with a trained model he can perform testing using the N2Sky platform.
- e) The neural network engineer user publish his neural network and trained model in order to make it available for other N2Sky users.

### 3. *Arbitrary User*

- a) The arbitrary user authorize in the N2Sky portal using his browser on desktop PC or mobile device. He will be redirected to his own dashboard according to permissions, which will be received from User Management Web Service.
- b) Since the user does not much knowledge in neural network field, he performs a semantic search in order to find some neural network as well as trained models according to his needs.
- c) The user copy existing neural network and some trained models into his project.
- d) The user perform training from N2Sky platform against copied neural network with the default input parameters data.
- e) The user evaluate trained neural network model with the default parameters.

### 4. *System Administrator*

- a) The system administrator authorize in the N2Sky portal using his browser on desktop PC or mobile device. He will be redirected to administration dashboard.
- b) The user observes cloud environment.
- c) The system administrator creates the new monitoring chart with a specific metrics and add it to the administration dashboard.
- d) The user creates alert against newly created monitoring
- e) The user is notified by Alert Management System, that some event occurs.



### 2.3.4 Technology Stack

N2Sky today is the cross-platform handy application with a responsive design. Create own framework from scratch would be time consuming. To build cross-platform framework just for N2Sky is an absurd. After some research of most popular and common used frontend frameworks following candidates were listed:

**Vaadin.** Java framework, which compiles Java code into JavaScript components. Vaadin supports cross-platform application, but not fully. It is possible to wrap an application into container and deploy it only as an android application.

**Benefits.** Easy to develop in Java. Developer does not have to think about JavaScript functionality. There are dozen of predefined components like: buttons, input fields, frames etc. Customisation is also possible.

**Obstructions.** Deployment process is a blockage process. There is no "hot redeployment" available. Even if developer win some time by build components in Java, he will lose much more time by continuous redeployment.

Java application need some server which supports JVM, it means that server should have much more memory then some other frameworks, which are written on JavaScript language.

**ReactJS.** JavaScript framework, which supports JSX programming language.

**Benefits.** The main idea is to write HTML code in JavaScript. ReactJS supports hot redeployment. React-Native extension for this framework, which allows to wrap the whole application mobile as well as desktop application.

**Obstructions.** Difficult to support a big project. JSX is not a type safe programming language. Exception handling also need to be done by developer.

**AngularJS.** JavaScript framework, which supports TypeScript programming language.

**Benefits.** The main idea is to write JavaScript code in HTML. AngularJS same as ReactJS supports hot redeployment. It does not have native support for all mobile devices, but is possible to wrap it using IONIC framework.

**Obstructions.** AngularJS compiles TypeScript code into JavaScript core and sometimes compilation fails because TypeScript is a new language and it is not fully adopted for browsers.

Vaadin does not fit N2Sky needs, but AngularJS and ReactJS could fit perfectly. Both of this frameworks are written on JavaScript and have big corporations behind: AngularJS was developed by Google and ReactJS was developed by Facebook. Since N2Sky has to support fully cross-platform architecture it was decided to choose ReactJS. With this framework N2Sky has a potential to be multi-platform application in the future.

Furthermore, backend has microservices architecture to support scalability. After choosing JavaScript framework for frontend it will make sense to use JavaScript in a backend too, so that server would be small and fast. Each one of the microservices is developed on NodeJS Framework server, which implies efficiency and lightweight.

N2Sky is a cloud-based system. OpenStack cloud platform supports this approach. Since N2Sky uses OpenStack API for administration dashboard, the original OpenStack dashboard was no more needed. Every backend and frontend service is deployed on OpenStack instance. Every instance is absolutely scalable, which allows to find a best environment for every service.

Every OpenStack instance is a server with either Debian or Ubuntu operational system. Every instance as well as OpenStack itself has to be monitored 24/7, that is why there was Monitoring Management System developed. The basis of this system is Prometheus monitoring system, which gives full access to all information of any server where it was installed. Prometheus has an open API, which is used by N2Sky. Using Prometheus there were charts and graphs developed and integrated into N2Sky. Alert Management System, which is a part of N2Sky, is also using Prometheus API in order to detect deviation and notify on event occurred.

As a database it was decided to choose NoSQL one. There were two NoSQL databases under consideration Elasticsearch and MongoDB. Elasticsearch supports indexes. It is possible to configure index so, that it is impossible to insert something which is not mapped by the index. MongoDB does not use index approach, but MongoDB client supports schema. It was decided to use MongoDB schema and mapped ViNNLS schema to it in order to make it more understandable for other developers.

For continuous delivery and quick configuration it was decided to use Jenkins Continuous Integration system. In case if the whole system will go down it is possible to restore every service with a Jenkins Profiles.

## 3 N2Sky Components

In the concept of N2Sky lies minimalistic and modern design. Dimmed tones of the UI components are used to make end-user to feel like he is using professional expert system. Every component and element was carefully thought out in order to keep the same atmosphere through the whole application.

### 3.1 N2Sky Frontend Application

Familiar elements and components help users easier navigate through the application. It is important to have common components and elements and do not mix up all together. Every component and every GUI element should have self-describing purpose.

Building user interface components and elements is pretty same as develop some UML digram. It is possible to group common parts and maximise reusability [18].

It is important to consider the multiple end-users, devices, platforms as well as environments will be used. Throw heterogeneous context only needed elements has to be displayed. It is necessary to make view prototypes to determine if the particular component comfortable and easy to use [12].

### 3.1.1 N2Sky Layouts Design

Design of N2Sky layouts was concentrated on principles of User interface design which were described in [19]:

**Organise.** All UI elements and components has to be ordered and not be chaotic.

Randomise position or logically not understandable can confuse the user.

**Consistency.** There are few types of consistency:

- Internal consistency, when elements are represented on the same place in the familiar components.
- External consistency, when few elements are looking little bit different, but the same functionality. This happens often on different devices, especially on mobile devices.
- Real-world consistency, which bring real world symbols into application UI elements.

**Screen Layout.** There are few screen layouts:

- Grid layout, which organise all components into blocks, like: menus, navigation bars etc.
- Standardise the screen layout, which mostly used on screens with a restrictions.
- Group related elements, which is usable for smaller screens.

In N2Sky Grid layout is used since it is possible easily to apply responsive design and reorganise grit items.

**Navigability.** Navigation has to be always on user focus or at least easily to find how where and how to user navigation elements.

**Economize.** Following rules has to be applied:

- Simplicity
- Clarity
- Distinctiveness

- Emphasis

**Communicate** . The layout has to apply accuracy, typography, symbolism, multiple views to helps user to communicate with the application.

### 3.1.2 N2Sky Fundamental Layout

N2Sky Fundamental Layout is represents basic styling of N2Sky application as it shown in “Fig. 9”. This layouts applies styling like:

- Colors
- Formating
- Fonts
- Positioning of elements and components

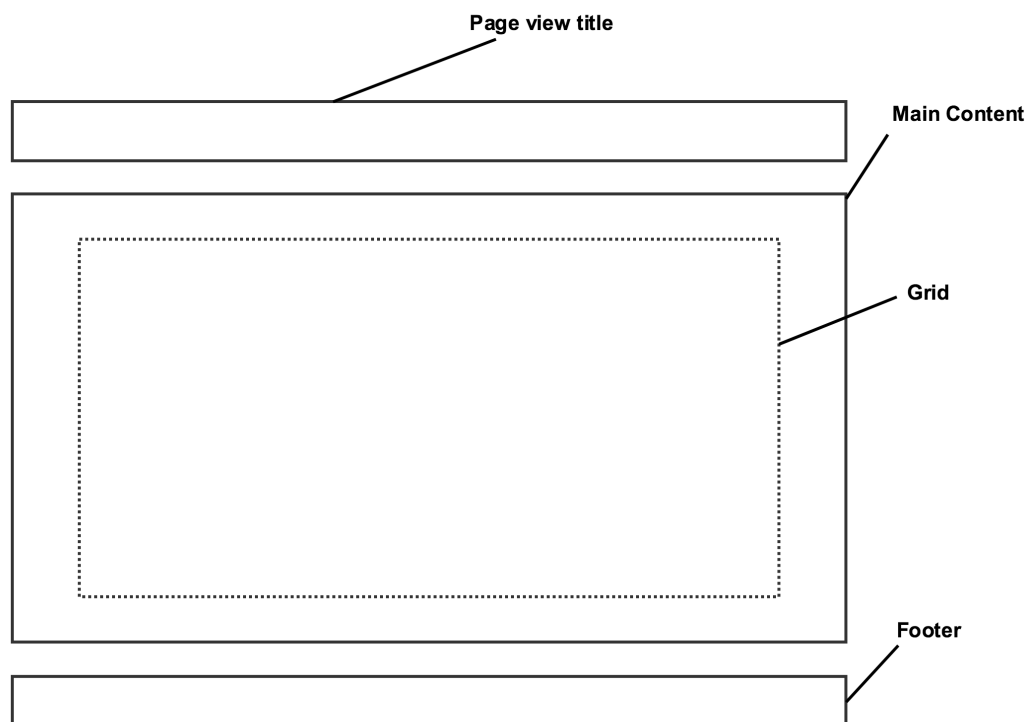


Figure 9: N2Sky Fundamental Layout

Fundamental Layout is a base layout which contains following elements:

- Page view title is a component with a caption and optionally an icon or push-to-action button
- Main Content is centred component, which represent primary context. This component is always in focus of user and can contain one or more grid components.

- Grid in basic layout represent positioning of elements, which can be displayed inside it. In basic layout in grid component is possible to add any elements.
- Footer is a component which goes across the whole application with a static data. Normally it is contacts and terms and conditions.

### 3.1.3 N2Sky Main Layout

N2Sky Main Layout, which is shown in “Fig. 10”, is extending N2Sky Fundamental Layout. Any changes in basic layout will be automatically reflected in main layout. Following components were added:

- Main Navigation Menu, which displayed as a vertical bar. This component has two states: on mouse over it will be extended and menu items with a caption text and icon will appear, on mouse away from the component only menu items icon will be displayed.
- Menu items are icons with a captions components. Which items will be shown or hidden are depends on user permissions.
- Grid items are block components, which can has multiple size, but fixed within one grid. Context of grid item can be customised.

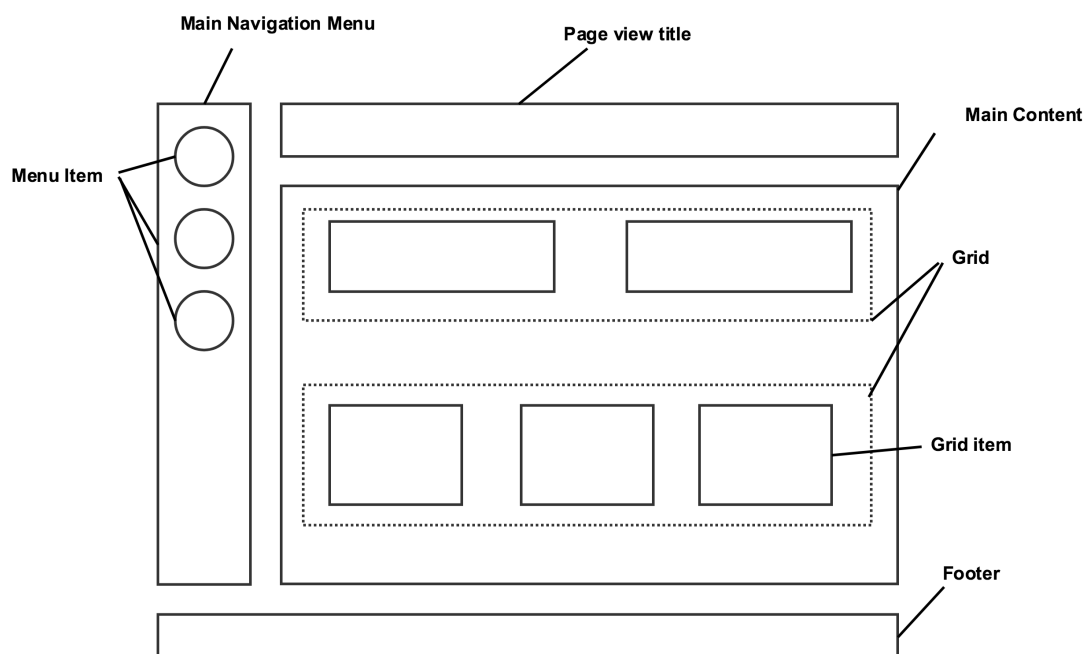


Figure 10: N2Sky Main Layout

### 3.1.4 N2Sky Mobile Layout

N2Sky supports mobile devices. For mobile devices the mobile layout was developed as it shown in “Fig. 11”

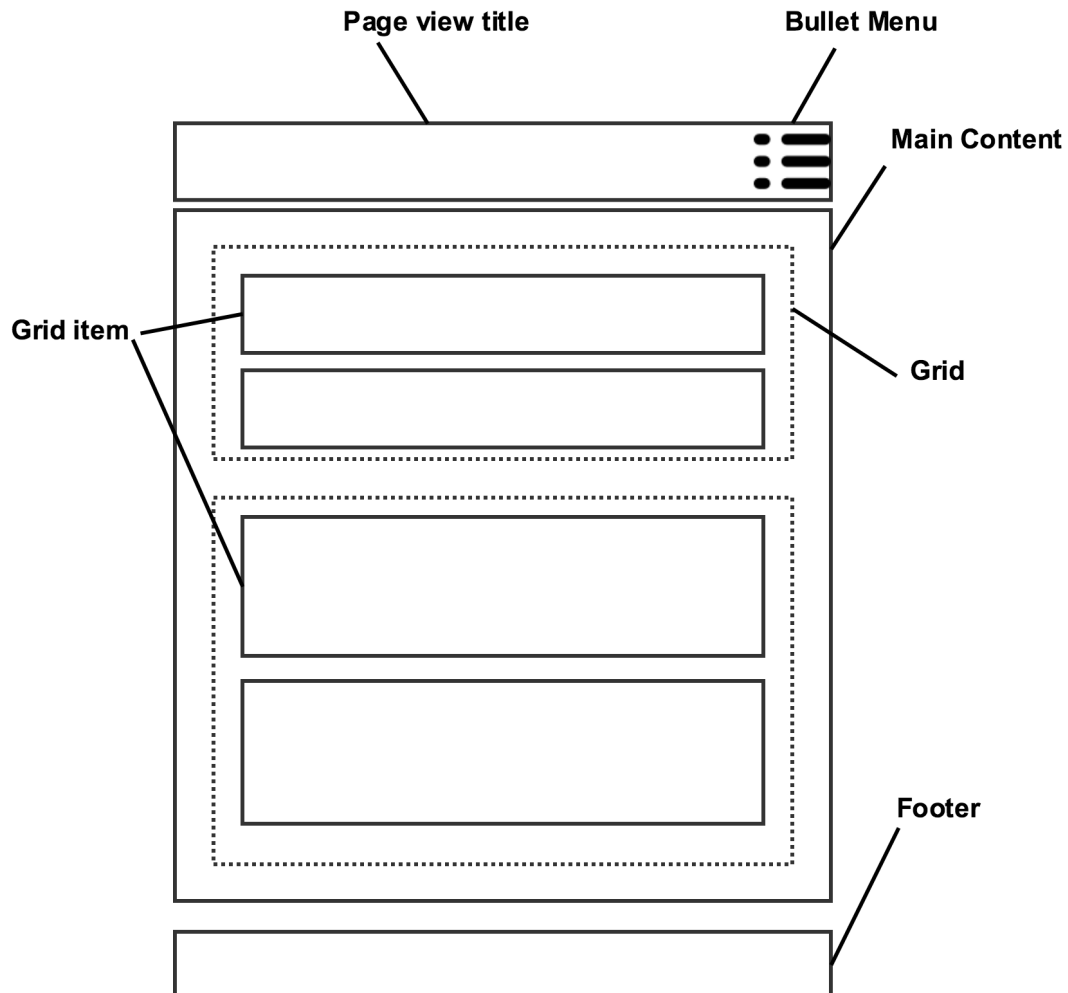


Figure 11: N2Sky Mobile Layout

All components context remain the same, but the positioning is changed. All grid items are vertically located and the width of grid items are the same as the device screen size.

Additionally next to page view title component is Bullet Menu located. The normal desktop view is hidden and instead bullet menu perform the same functionality. On mouse click the menu will appear as a overlay of the current view.

### 3.1.5 N2Sky Page Views

As it was mentioned in subsubsection 2.3.4 "Technology Stack", N2Sky frontend application developed on ReactJS framework. Part of ReactJS framework is React-Router,

which contains all page views of the application and redirects it accordingly to URL path:

```

1 <Provider store={store}>
2     <Router history={browserHistory}>
3         <Route path="/" component={Auth}/>
4         <Route path="/signup" component={Reg}/>
5         <Route component={AbstractDashboardLayout}>
6             <Route path="/cloud" component={DashboardsOverview}/>
7             <Route path="/user/profile" component={UserProfile}/>
8             <Route path="/openstack"
9                 component={OpenStackMainDashboard}/>
10            <Route path="/openstack/project/:id"
11                component={OpenStackProjectDashboard}/>
12            <Route path="/openstack/server/:projectId/:serverid"
13                component={ServerDetailsDashboard}/>
14            <Route path="/openstack/vitrage/:templateId"
15                component={VitrageDetailsView}/>
16            <Route path="/alert" component={AlertDashboard}/>
17        </Route>
18        <Route component={AbstractDashboardLayout}>
19            <Route path="/n2sky" component={N2SkyDashboard}/>
20            <Route path="/n2sky/available"
21                components={AvailableNetworksOverview}/>
22            <Route path="/n2sky/models"
23                components={ModelsRepository}/>
24            <Route path="/n2sky/paradigm/create/:projectId"
25                components={AddNNFromParadigm} readOnly={false}/>
26            <Route path="/n2sky/paradigm/nn/:id"
27                components={AddNNFromParadigm} readOnly={true}/>
28            <Route path="/n2sky/network/:id"
29                component={NetworkDetails}/>
30            <Route path="/n2sky/network/:id/test/:model_id"
31                component={NetworkTestDetails}/>
32            <Route path="/n2sky/project/:id"
33                component={ProjectDashboard}/>
34        </Route>
35    </Router>
36 </Provider>

```

Every "Route" is a page view redirector and contains:

- Path, which responsible for URL redirection.
- Component, which is a page view itself
- Other props, which are optional

As it was mentioned in subsubsection 2.3.2 "Modular frontend application design", N2Sky contains of two modules: Administration module and main application module.

Both of this modules sharing the same main application layout and having following page views:

- **Administration Module:**

**Cloud Dashboard View, path: `"/cloud"`.** It is the main dashboard of Administration module, which contains overview dashlets of every modules component.

**OpenStack Dashboard View, path: `"/openstack"`.** This view contains all information about OpenStack and the main control centre of the OpenStack services.

**OpenStack Project, path: `"/openstack/project/:id"`.** Route to the OpenStack project, which contains information about servers, flavours, images etc. of particular project. In path "id" is the id of OpenStack project.

**Server Details View, path: `"/openstack/server/:projectid/:serverid"`.** This view contains information about OpenStack instance. The URL path needs OpenStack project id and the id of the OpenStack instance.

**Vitrage Details View, path: `"/openstack/vitrage/:templateid"`.** Vitrage it a monitoring service of OpenStack and its instances. This view contains information about this service. The template id is required.

**Alert Management Dashboard View, path: `"/alert"`.** This view is a dashboard of Alert Management System. This view contains information about alerting rules and alerting events.

- **Main Application Module:**

**Authentication View, path: `"/"`.** First page, which user sees, when he loading the application. Authentication View contains login form.

**Registration View, path: `"/signup"`.** Registration View contains registration form.

**N2Sky Dashboard, path: `"/n2sky"`.** Main Application Module view is the N2Sky Dashboard. This view contains information about logged in user projects, neural networks and trained models.

**Available Networks View, path: `"/n2sky/available"`.** This view is the neural network repository of the N2Sky. It is possible to copy available neural networks to own project.

**Models Repository View, path: `"/n2sky/models"`.** The trained neural network are showed in this view. It is possible to copy published models to own project.



**Neural Network Create View, path: `"/n2sky/paradigm/create/:projectid"`.**

This view represents workflow of creation of neural network from existing paradigm. Newly created neural network will be saved in user's project, hence this project id is required.

**Neural Network Details View, path: `"/n2sky/network/:id"`.** The details of created neural network. It can be as well loggedin user neural network or neural network from other user. User permissions shows visibility level. The networks id is required.

**Neural Network Test View, path: `"/n2sky/network/:id/test/:model_id"`.** From this view user can evaluate neural network this his own input parameters. Neural network id and trained model id are required.

**Project Dashboard View, path: `"/n2sky/project/:id"`.** This view shows the neural networks and models, which were created in particular project. The project id is required.

### 3.1.6 N2Sky Dashboards

The purpose of dashboard is to embed business intelligent (BI) objects into single page in order to make an overview of highlighted titles of BI objects [17].

A dashboard has a structure layer upon its dashlets. It is allow the user to manage layout and properties of dashlets. A dashboard is composed by dashlets. Every dashlet contains specific context and data. Dashboard defines:

- Dashlets to be displayed
- The layout of dashboard and positioning of its dashlets
- The common context of particular dashlets

In “Fig. 12” is shown the typical N2Sky dashboard template. A dashboard in N2Sky has a grid structure. Every dashlet has one of more items. Every item can contain any UI component, but it has to be feet into assigned dashlet without overlays.

### 3.1.7 Responsive design

Since N2Sky is cross-platform application it has to support multiple screen resolutions as well as readability and usability.

Considering readability first of all the typography has to be mentioned. During development of N2Sky to find the fonts, which will be displayed same and will be readable across multiple devices was a challenge. The typeface [2] properties like latter spacing, width, weight etc were adjusted multiple times. The common problem was to audit how typeface represents on mobile devices and desktop computers [25]. Even such a

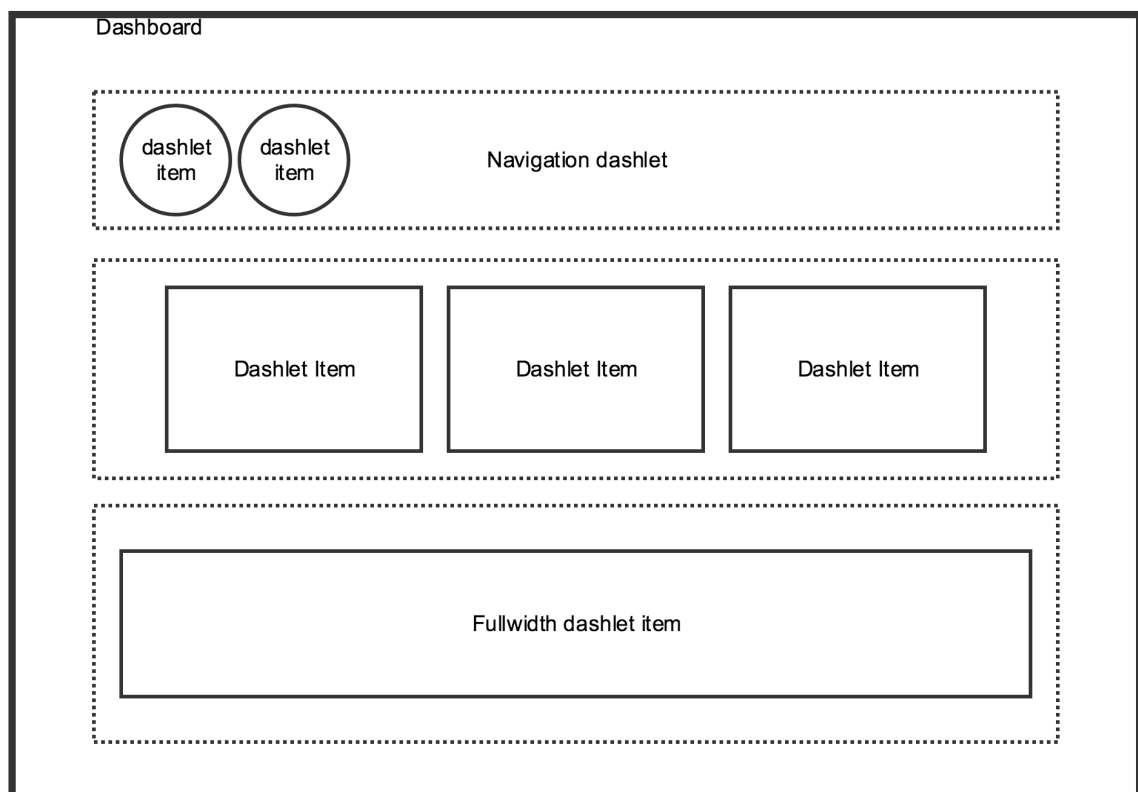


Figure 12: N2Sky Dashboard template

standard fonts like "Arial" and "Times New Roman" were looking unreadable on mobile devices as it shown in “Fig. 13”.



Figure 13: Difference between sans-serif font Arial (A) and the browser serif font Times New Roman (B)

The expansion of cross-platform applications brought some freedom to developers and designers. Today is possible to develop an application simultaneously for mobile devices, desktop computers and web browsers. The problem is came, when the mobile and tablet devices manufacturers started to produce devices with a different resolutions. Today versioning of mobile screen resolution came over 1000 devices, but N2Sky was concentrating only on major market players as it shown in “Fig. 14” [16].

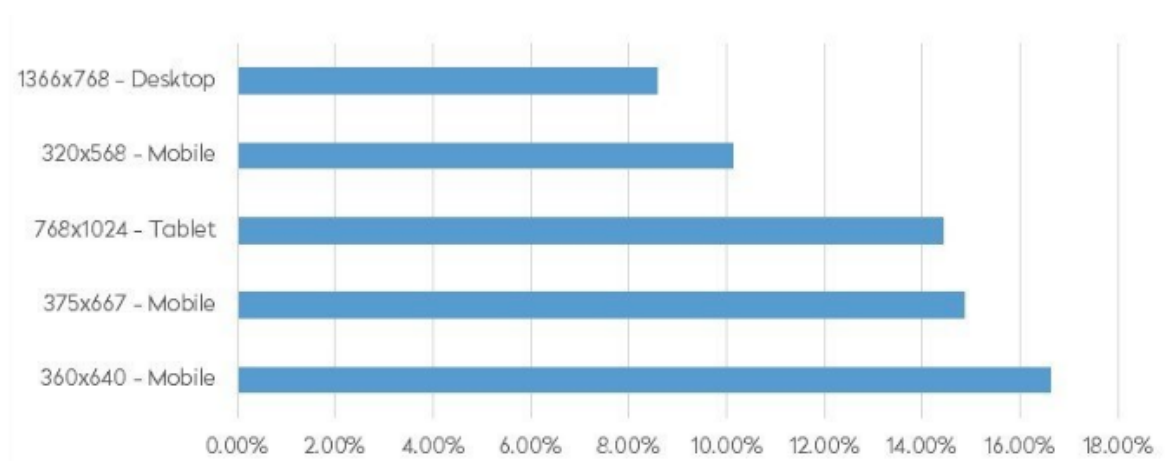


Figure 14: Average screen resolution of mobile, tablet and desktop devices on 2017

N2Sky is focused on mobile and tablet devices, because it is an extensive trend nowadays.

### 3.1.8 User Interface Elements

It is possible to divide all UI elements into groups [10]:

**Input Controls.** Input controls determine user input action. Input actions are keyboard typing or mouse clicking. Following UI elements are the part of input controls:

- Checkboxes
- Radio buttons
- Dropdown Lists
- List boxes
- Buttons
- Toggles
- Text fields
- Date field

**Navigational Components.** Navigation between page views. Navigational components includes also some particular request from and to user. Following UI elements are the parts of navigational components:

- Breadcrumb
- Slider
- Search field
- Pagination
- Slider
- Tags
- Icons

**Informational Components.** This components are addition to workflows. It can help user to perform some actions or they can to inform user, that the action will occur or already occurred. Informational Components contains of following UI elements:

- Tooltips
- Icons
- Progress bar
- Notifications
- Message boxes

- Modal windows

**Containers.** Containers are components, which are hiding additional information or not focused information, where user need to perform action in order to see it.

- Accordion
- Semihidden elements

### 3.1.9 UI Elements in N2Sky

N2Sky supports almost all common web user interface elements. Every element was developed in order to be reusable. Since N2Sky supports responsive design, the UI elements should also be responsive. Every UI element is absolute customised. Following UI elements were created:

**Accordion** Accordion is a list of items, which is accessible on mouse click. In N2Sky accordion works more like a modal window. With this kind of functionality other data, which surround accordion will not be stretched or squeezed, but will appear on top of elements “Fig. 15”:

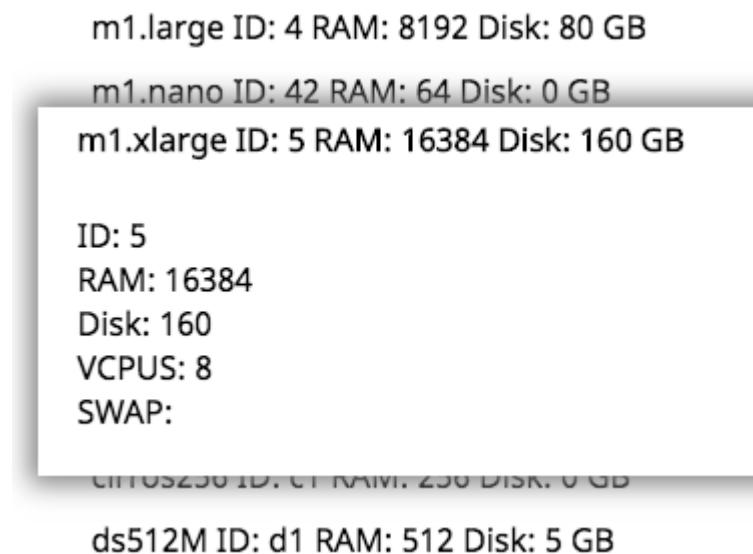


Figure 15: Customised N2Sky accordion UI element

**Buttons.** Idea behind was to make buttons more interactive and understandable to use “Fig. 16”. Buttons contain caption and icon in SVG format in order to support high quality image in all devices.

Buttons are using on hover animation. When mouse over the button, then button icon slide to the middle to show that action can be performed “Fig. 17”.



Figure 16: Customised N2Sky button UI element

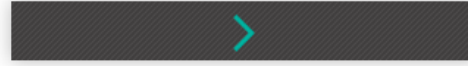


Figure 17: Customised N2Sky button UI element animation

**Icons.** In N2Sky all icons are in Scalable Vector Graphics (SVG) format. With SVG the icons does not loosing quality in any device [5]. Since it is a vector graphic it easy to edit an icon with programming language code. N2Sky Logo, which is represented in “Fig. 18”, is also made in SVG format.



Figure 18: Customised N2Sky logo in SVG format

Is it possible to import code in some graphical vector editor in order change colour, path (vector graphic itself), metadata etc. Following code demonstrate N2Sky Logo in SVG format. The whole vector path was shortened.

```

1 <?xml version="1.0" standalone="no"?>
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
3     "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
4 <svg version="1.0" xmlns="http://www.w3.org/2000/svg"
5     width="266.000000pt" height="267pt" viewBox="0 0 266 267"
6     preserveAspectRatio="xMidYMid meet">
7   <metadata>
8     N2Sky Logo. 2018
9   </metadata>
10  <g transform="translate(0.000000,267.000000) scale(0.100000,-0.100000)"
11    fill="#6b6b6b" stroke="none">
12    <path d="M1302 2638 c-9 -9 -12 -83 -12 -270 0 -245 -1 -259
13 -37 -26 -170 21 -63 23 -132 46 -152 52 -23 7 -38 17 -38 27 1
14 80 222 90 255 87 284 -2 23 -9 32 -25 34 -27 4 -46 -23 -72 -106
15 -77 -34 -95 -8 -18 -22 -51 -32 -74 -10 -24 -21 -43 -25 -43 -5 0 -30
16 76 -76 118 -100 136 -128 92 -12 -20 -10 -26 101 -219 35 -60 109
17
18 ...

```

```

19
20 -18 74 -40 36 -22 67 -40 70 -40 11 0 253 -152 269 -169 11 -12 15 -27 12
21 -36 26 -103 -14 -32 -19 -60 -35 -62 -35 -3 0 -34 -18 -70 -40 -36 -22 -68
22 -40 -70 -40 -3 0 -22 -11 -43 -23 -142 -90 -172 -96 -201 -39 -20 40 -47 168
23 -55 268 1-5 61 122 22 c67 13 156 30 197 38 85 16 106 30 95 63 -7 21 -11 22
24 -69 16 -33 -4 -88 -10 -121 -15 -178 -26 -178 -26 -170 -1 4 13 21 46 37 74
25 57 100 63 116 52 134 -6 9 -22 17 -34 17 -24 0 -47 -33 -130 -180 -75 -132
26 -74 -130 -61 -206 6 -38 16 -102 21 -141 6 -40 15 -98 20 -129 6 -30 10 -72
27 10 -93 10 -38 -162 4 c-151 3 -167 5 -212 28 -82 43 -86 57 -86 320 10 226 38
28 34 c21 19 53 47 72 61 19 14 49 39 67 55 17 16 54 47 82 69 60 49 79 79 65
29 103 -18 28 -47 25 -84 -10 -20 -18 -52 -45 -72 -60 -20 -15 -42 -34 -50 -41
30 -23 -23 -92 -67 -105 -67 -10 0 -13 27 -13 104 0 57 -3 111 -6 120 -7 19 -45
31 21 -62 4z"/>
32 </g>
33 </svg>

```

**Notification messages.** Notification messages are the part of information component.

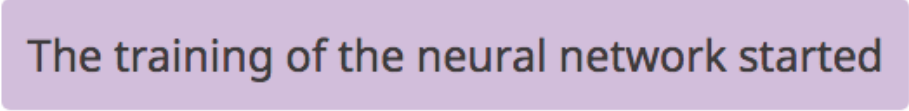
In N2Sky used two types of notification messages:

- Warning message, which tells, that something goes wrong as it shown in “Fig. 19”. It can be error in the application as well as user workflow error.
- Informational message that gives information about event which will occur or already occurred as it shown in “Fig. 20”.



Authentication failed. User not found.

Figure 19: Customised N2Sky warning message UI element



The training of the neural network started

Figure 20: Customised N2Sky informational message UI element

**Navigation.** In N2Sky user can navigate to another one page or change part of the page view via tabs, navigation buttons and menus. Navigation elements are bind into group of element or UI components. Following navigation component which is shown in “Fig. 21”, contains navigation elements as well functional icon.

Navigation bars can contain tabs, buttons, icons and non-functional text. “Fig. 22” shows navigation bar an custom table.



Figure 21: Customised N2Sky navigation UI element


| TESTING RESULTS |                           |                           | TEST THE MODEL    |
|-----------------|---------------------------|---------------------------|--|
| User            | Testing Data              | Result                    | Testing Date   |
| Fedorenko       | [[0,0],[0,1],[1,0],[1,1]] | [[ 0.] [ 1.] [ 1.] [ 1.]] | 2017-12-01 at 15:56  |

Figure 22: Customised N2Sky navigation bar with a table

### 3.1.10 UI Components in N2Sky

Groups of UI elements form UI components. Components, like elements, are also fully reusable, only context of components is changing. Following custom UI components were developed in N2Sky:

**Grid Item Component.** Grid in N2Sky is responsive. It can change positions and size of grid items like it shown in “Fig. 23”.

Grid item contains following UI element:

**Header.** Header is the first component on which the user focuses, that is why it should be short, but highlighted. Following UI elements are included:

- Functional icons-buttons on the left side (optional)
- Title of grid item (mandatory)
- Non-functional icons (optional)

**Main context.** Component, which contains context information. This component can be fully customised. It is possible to put there list of items, plain text or even image.

**Footer.** Footer is an optional element and contains only button UI element.

**Main navigation menu.** Every N2Sky view use main navigation menu, namely menu is injected in abstract view, which is extended by all other view components. Menu has a menu items, which contain caption and an icon. As it was mentioned before, N2Sky has two modules: administration module and main application module. Both modules are represented in menu and menu items visibility depending on



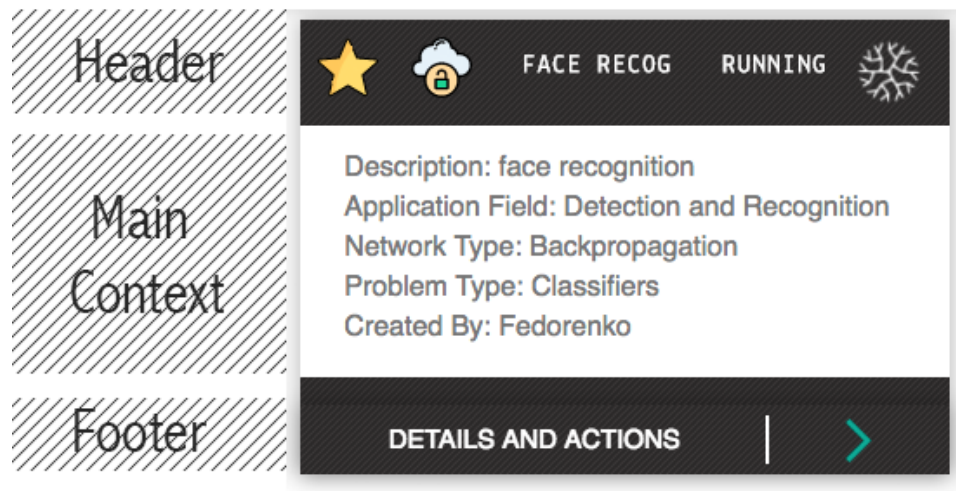


Figure 23: Responsive N2Sky Grid Item UI comonent

logged-in user permission. Menu for arbitrary user is shown in “Fig. 24” and has following menu items:

- Profile
- N2Sky Dashboard
- Available neural networks
- Models repository

If end-user is a system administrator, he can see additional menu form administration module as is demonstrated in “Fig. 25”. This menu contains dropdown submenus:

- OpenStack Dashboard
- Cloudify Dashboard
- Alert System
- Dashboards Settings

**Modal windows.** N2Sky use modal windows almost in every view as it shown in “Fig. 26”. Modals are responsive and touch screen friendly. It has 3 elements:

- Title (mandatory)
- Context (mandatory)
- Submit button (optional)

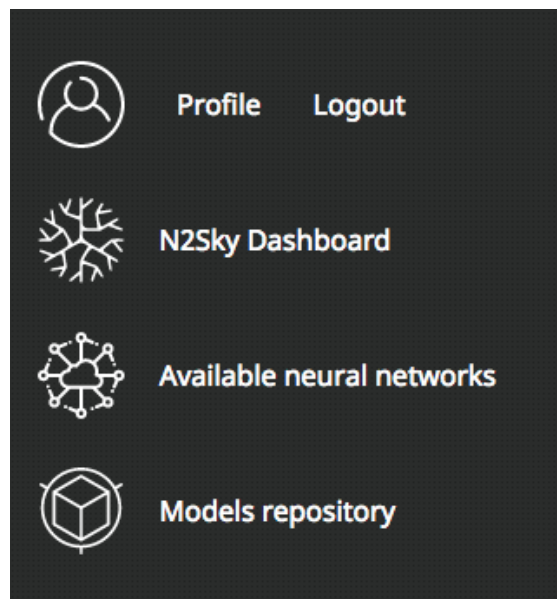


Figure 24: N2Sky Main Navigation Menu for arbitrary user

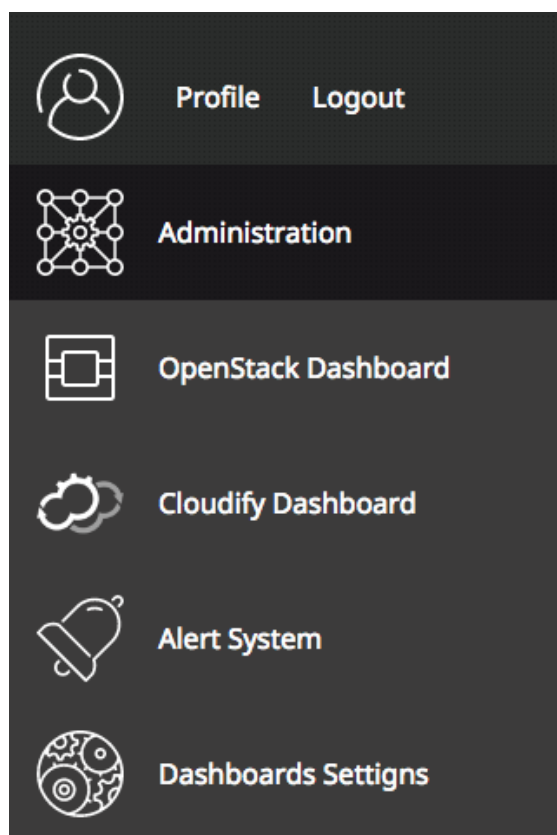


Figure 25: N2Sky Main Navigation Menu for system administrator

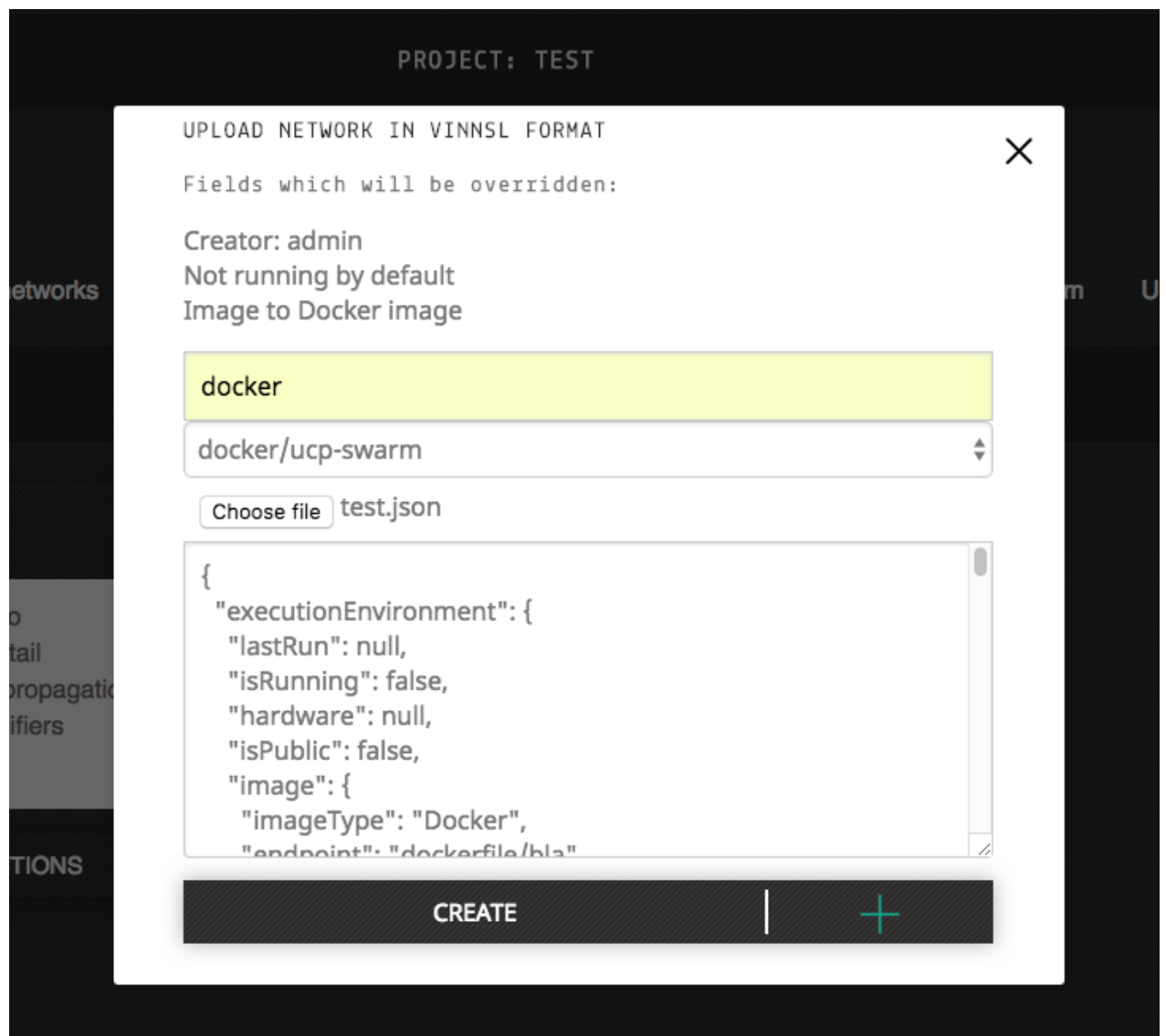


Figure 26: N2Sky Modal Window UI component

Modals can be used to represent form as well for informational purposes. When modal is open the background is dunned in order to focus user on modal context. Modal context itself can be fully custom. It is possible to put any UI element in context.

## 3.2 N2Sky Services

N2Sky implements microservices architecture. It has 3 main web services as it shown in “Fig. 8” :

- User Management Web Service
- Model Repository Web Service
- Cloud management Web Service

Every web service use other services which are not exposed to the public. It was made in order to support application encapsulation. Encapsulation of web application helps to prevent security issue. One of the crucial crucial process in N2Sky is the neural network training. This process takes almost all resources of the environment, that is why it is not exposed. Such a processes can be triggered only by web service, which can be blocked if environment is overloaded.

Consider the following web services in details and which processes and services they are encapsulate:

### 3.2.1 User Management Web Service.

This web service responsible for permissions and user management and it has its own database. User can authorise in the system and get a session token. Every token is an unique collection of numbers and latin letters. Token is assign to the authorised user and will be save until user is active. If user is not active in the next 3 hours, the session token will disappear. If user still have an active browser session, the authorisation token will be revalidate. If user trying to make some illegal request or behave too active, the authorisation token will be revoked and the system administrator will be notified about this incident.

Every user encapsulate permission level. There two different types of permissions:

- Administrator permission. It means, that the user has full granted permission throw entire application.
- Regular permission. The user has access only for his own as well as public available resources.

As it shown in “Fig. 27” User Management Web Service has following accessible services:

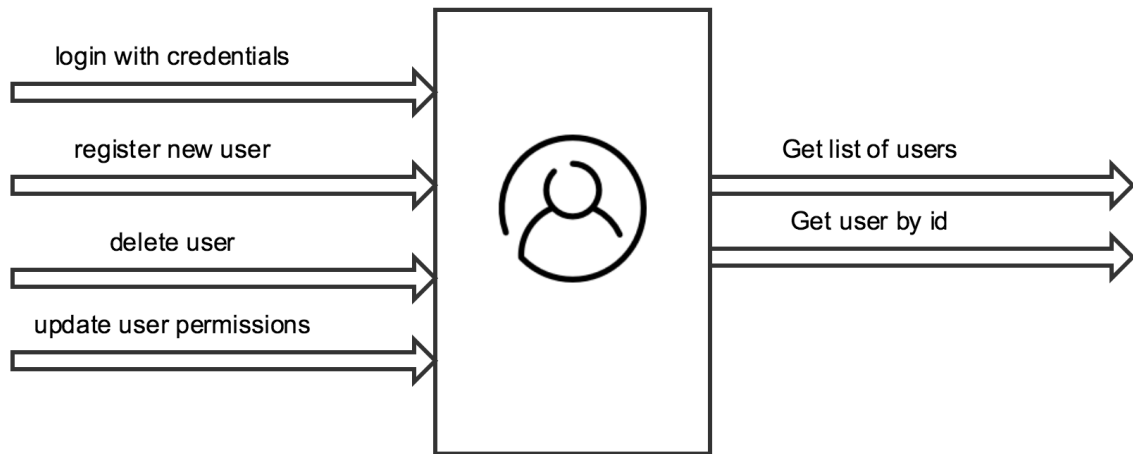


Figure 27: N2Sky User Management Web Service

- Login with credentials
- Register a new user
- Delete user
- Update user permissions
- Get list of users
- Get user by user ID

Detailed information about web service API and API documentation is written in subsection 9.3

### 3.2.2 Model Repository Web Service.

Model Repository Web Service is the core service of N2Sky. Authorised user can create a new project, add there neural network from chose paradigm or deploy own one. Every newly created project is assign for one user and can not be shared, only system administrator can look up into other users projects. This functionality also limited by User Management Web Service.

Using this service user can create a neural network from proposed paradigms as well as upload his own neural network in **ViNNSL** format. This functionality exposed via service so that every user can use it either from N2Sky web portal or via HTTP request directly on web service.

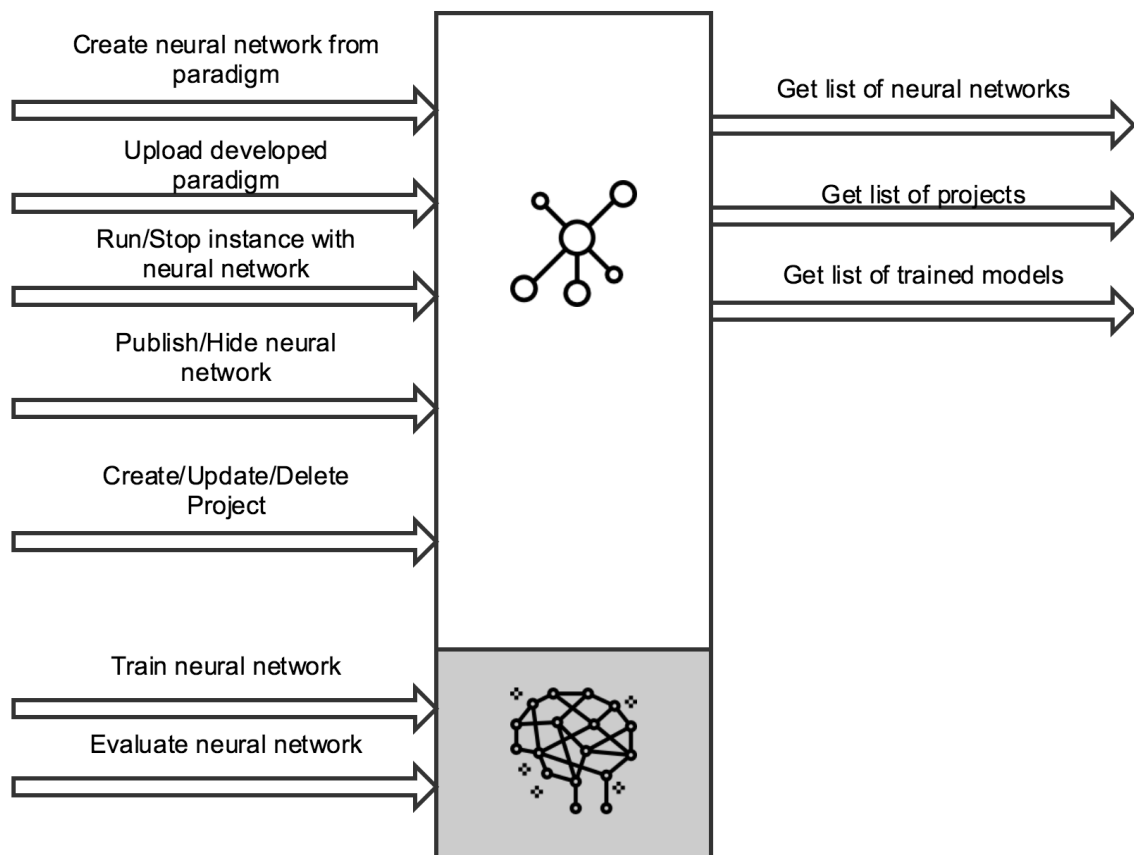


Figure 28: N2Sky Model Repository Web Service

As it shown in “Fig. 28’ Model Repository Web Service has following accessible services:

- Create neural network from paradigm
- Upload developed neural network paradigm
- Run/Stop instance of neural network
- Publish/Hide neural network
- Create/Update/Delete Project with neural networks
- Get list of neural networks
- Get list of projects
- Get list of trained models
- Train neural neural network (accessible only for model management service itself)
- Evaluate neural network (accessible only for model management service itself)

Detailed information about web service API and API documentation is written in subsection 9.3

There are two services embedded to the Model Repository web service and not exposed:

**Training service.** This service provides neural network training functionality. It is not possible to perform training to make direct request on service endpoint.

In order to perform training the user has to know training input parameters and type of training file which can be accepted. This information is stored in ViNNsL schema.

Only Model Repository web service can trigger this service after being insured that environment available and ready to perform tests. Training is a long term process, but it does not block an entire application. This service writes log data to the instance. Model Repository repository makes a callback to training service in order to check if training is completed. If training is still processing, the Model Repository Service will fetch the log data in order to present current training result. User can also decide to stop training process if he is satisfied with the current result.

If user performs training on his own neural network he can also log result about his network and environment behaviour.

**Testing service.** The testing service allows users to evaluate a trained model. Testing data is described in ViNNsL format. Normally testing is not a long term process, because it is running against trained neural network model. Since there is absolute freedom by neural network structure customisation, the testing process can be inefficient and could take some resources from the environment. Considering this fact it was decided to encapsulate this process too.

### 3.2.3 Cloud Management Web Service.

Cloud web service is originally available only for system administrator. This service can manage OpenStack environment and Cloudify container management system. On every OpenStack instance as well as on OpenStack itself the monitoring management system service is installed. Every monitoring management system has its own metrics configurations and alerting rules. The monitoring service is only exposed via Cloud management web service.

Cloud management service supports Platform as a Service distribution. The system administrator can configure the service according to his needs. All rules, including configuration rules for monitoring and alert management systems, can be adjusted on demand.

As it shown in “Fig. 29” Cloud Management Web Service has following accessible services:

- Get user’s Dashboard settings
- Get user’s saved metrics (reference to subsection 3.3 )
- Get OpenStack Services: flavours, servers, networks, images etc.
- Create / Update user’s OpenStack Dashboard settings

Detailed information about web service API and API documentation is written in subsection 9.3

## 3.3 Continuous Monitoring System

In client-server architecture especially for OpenStack cloud platform computer servers are important. Servers could go down, but the system-administrator does not have to work 24/7 in order to monitor the system and wait until some problem occurs. A system administrator could use some monitoring tools like Nagios for OpenStack in order to get information about servers state. Monitoring tool can give information about system, network and infrastructure [4]. The purpose of the monitoring tool is to find some problems and inform the system administrator about some problem occurred. This tool can not solve the problem, but it can give information what went wrong.



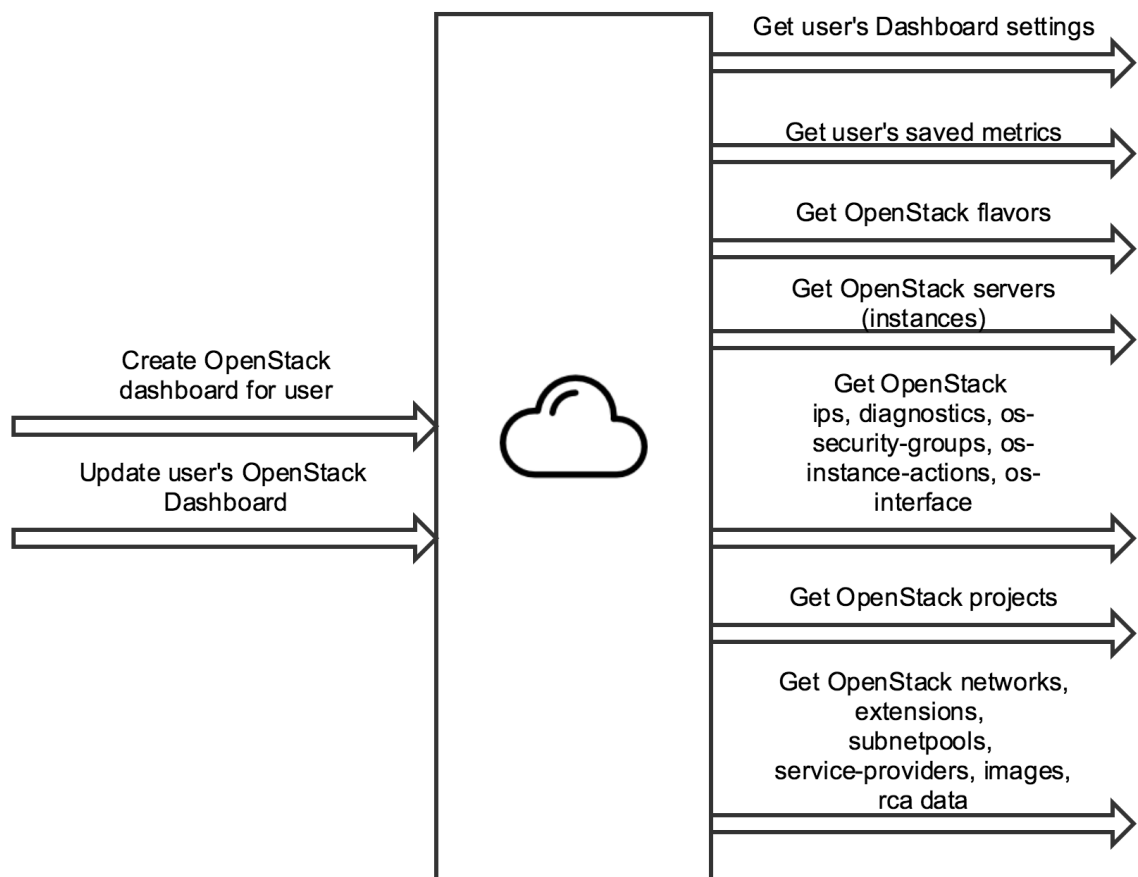


Figure 29: N2Sky Cloud Management Web Service

### 3.3.1 Monitoring requirements

The base monitoring system is readable and understandable representation of the graph. Graphs allow you to see objects being monitored and recognise metric from these objects. The good monitoring graph gives meaningful description, helps quickly to detect and determine issues via representation. This kind of graph should serve as a motivation for action to solve problems. There are some simple rules, which makes graphing well:

**Consistency.** Representation should correctly reflect reality. All objects, which are represented on the graph, must be correlated with a real data on the machine.

**Graphs need to make sense.** All lines represented on the chart have to be readable and understandable. Fake or unreadable information could cause problems. Metrics set should be small, one metric represent one object. There is no need to put multiple objects which does not bind directly to one chart.

**Stacked area vs. multiline area.** Not every chart should have the same visual representation of the lines. Depending on the case we can decide which type of area to use. If there are small time series with a high frequency it is better to use multiline area and stacked area on longer time series but with a bigger metric set respectively.

**Understanding graph before starting to analyse it.** Since there are going to be multiple charts with a different metrics we need to make sure that every user can understand the meaning of particular graph. Good naming, fulfilled content and correct positioning are very important.

**Data hierarchy.** It is important to define groups, metrics, data points and nested levels on the chart. Groups help to bind similar objects together. Data points give information of time stamps. Metrics is actual graph representation. Nested level is multiple line metrics. All mentioned data should be visible and accessible.

**Clarity.** Designing a chart it is important to take into consideration that there are multiple devices with a different screen resolution. Too many lines on limited space will make chart unreadable. If there are lots of charts on one page it makes people confused what a meaning of this page. That is why it is a good practice to create multiple pages with grouped charts.

**Perspective.** It is important to put graphs in such perspective so that any deviation will be easily noticeable.

**Appeal.** All charts are people oriented, people today like simple and clean appearance of the applications and if an application has lots of charts they need to be with an appropriate design.

**Control and managing.** It should be possible for any user to manipulate a graph.

Either to change time series or remove some metric, customization of the graph makes the whole application more attractive.

### 3.3.2 Applying Monitoring

To build continues monitoring system there is need to user a toolkit with an active ecosystem. Searching for a proper toolkit the toolkit should fulfilled some specific requirements that going to be use in N2Sky:

- Proper and self-describable metric name with a key pairs
- Possibility to query metrics and join them in one graph.
- Not resource-intensive
- Support HTTP/HTTPS protocol
- Collect and push to repository time series
- Scalable

After researching it was decided to go for Prometheus Monitoring Toolkit [13]. This tool support all requested requirements. One of most interesting feature is that Prometheus can be used on any UNIX environment. Since in Openstack multiple instance with a different operational system can be created, Prometheus will match exact our needs. Originally Prometheus was created by SoundCloud team in 2012. The core of monitoring application is Prometheus server, which collects time series data from the moment it was executed on environment as it shown in “Fig. 30”. All components are written on Go language and support multiple modules for monitoring different environment metrics. To understand the nature of Prometheus it is necessary to explain its architecture.

The core Prometheus server pull all metrics from jobs which are instrumented, if the service is unavailable for instrumentation it can be pulled from push gateway. All metrics and logs data is stored locally so there is no distributed storage. It is possible to query this data to retrieve more specific information about particular metrics of joint metrics. N2Sky uses Prometheus API to build own customised dashboards. The common components of Prometheus architecture:

**The Prometheus server.** This is the base element in the whole architecture. The server include services which collection, storing and retrieving nodes. The principle is scrapping or pulling. It means that the data fetched with some interval, which can me configured and stored accordantly as a time series. Prometheus support different modules, each module represents some node. The nodes expose

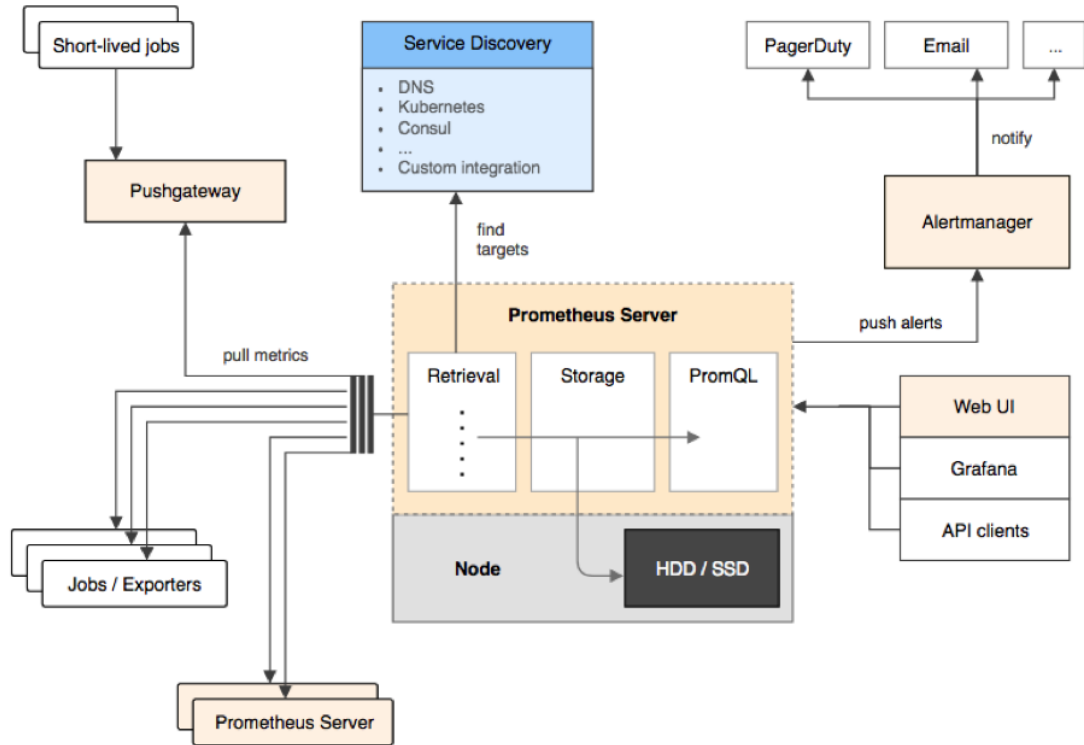


Figure 30: Prometheus monitoring architecture

these ports that Prometheus uses for retrieving the data. For example in N2Sky we are using Node Exporter Module which gives possibility to collect almost all essential data like CPU, RAM, HDD/SSD etc.

**Push gateway.** There are some nodes, which are not exposing these endpoints. In this case collection of the data throw Push gateway is possible. Prometheus short-lived jobs are executed to capture the data and convert it to the time series that can be used by Prometheus.

**Alert Manager.** Monitoring consist of multiple metrics, each metric can be analysed. It is possible to subscribe to particular metric in order to detect metric behaviour namely metric deviations. Alert Management System used for firing events, it is possible to receive alert notification over multiple channels like Emails, SMS, Push notification etc.

Metric notation Following example represent metric notation:

```
1 node_filesystem_avail {method="GET", endpoint="/api/posts", status="200"}
```

The metric naming is always self-described. Requested metric "node\_filesystem\_avail" means available free space in the filesystem. Every operational system has different metic naming. N2Sky will propose list of available metrics. In curly brackets defined the type of request, an endpoint and expected HTTP response status.

After executing this query the data will be retrieved from logs and represent as a time series as it shown on “Fig. 31”.

| Element  | Value        |
|--|--------------|
| node_filesystem_avail{device="/dev/sda2",fstype="ext4",instance="localhost:9100",job="node",mountpoint="/boot"}  | 327512064    |
| node_filesystem_avail{device="/dev/sda5",fstype="ext4",instance="localhost:9100",job="node",mountpoint="/"}  | 53616574464  |
| node_filesystem_avail{device="/dev/sdb1",fstype="ext4",instance="localhost:9100",job="node",mountpoint="/home"}  | 906128850944 |
| node_filesystem_avail{device="/dev/sdb2",fstype="ext4",instance="localhost:9100",job="node",mountpoint="/var"}   | 417015595008 |
| node_filesystem_avail{device="/dev/sdb2",fstype="ext4",instance="localhost:9100",job="node",mountpoint="/var/lib/docker/aufs"}   | 417015595008 |
| node_filesystem_avail{device="/dev/sdb2",fstype="ext4",instance="localhost:9100",job="node",mountpoint="/var/lib/docker/plugins"}  | 417015595008 |
| node_filesystem_avail{device="/dev/sdb3",fstype="ext4",instance="localhost:9100",job="node",mountpoint="/opt"}   | 279920013312 |
| node_filesystem_avail{device="/dev/sdb4",fstype="ext4",instance="localhost:9100",job="node",mountpoint="/usr/local"}   | 186646953984 |
| node_filesystem_avail{device="gvfsd-fuse",fstype="fuse.gvfsd-fuse",instance="localhost:9100",job="node",mountpoint="/run/user/1000/gvfs"}  | 0            |
| node_filesystem_avail{device="none",fstype="aufs",instance="localhost:9100",job="node",mountpoint="/var/lib/docker/aufs/mnt/37b40f519e26a974577833d0668a34e04134e5fd2537f1fb4c2fe6b7b0d53539"}       | 417015595008 |
| node_filesystem_avail{device="nfs",fstype="nfs",instance="localhost:9100",job="node",mountpoint="/run/docker/netns/1-szz5t6pxd8"}  | 0            |
| node_filesystem_avail{device="nfs",fstype="nfs",instance="localhost:9100",job="node",mountpoint="/run/docker/netns/618ed9d7179f"}  | 0            |
| node_filesystem_avail{device="nfs",fstype="nfs",instance="localhost:9100",job="node",mountpoint="/run/docker/netns/ingress_sbox"}  | 0            |
| node_filesystem_avail{device="shm",fstype="tmpfs",instance="localhost:9100",job="node",mountpoint="/var/lib/docker/containers/be36a80809d4fc59a1f8c3bd0da32b6e753e1ab7902d2676d50527d9e4458a74/shm"} | 67108864     |
| node_filesystem_avail{device="tmpfs",fstype="tmpfs",instance="localhost:9100",job="node",mountpoint="/run"}  | 1237848064   |
| node_filesystem_avail{device="tmpfs",fstype="tmpfs",instance="localhost:9100",job="node",mountpoint="/run/lock"}   | 5238784      |
| node_filesystem_avail{device="tmpfs",fstype="tmpfs",instance="localhost:9100",job="node",mountpoint="/run/user/1000"}  | 1248632832   |
| node_filesystem_avail{device="tmpfs",fstype="tmpfs",instance="localhost:9100",job="node",mountpoint="/run/user/123"}   | 1248690176   |

Figure 31: Metric response

One of the requirements to our monitoring system is scalability. One of the greatest features of Prometheus is that even if environment going to be overloaded it will generate the same amount of metrics anyway. Hence the amount of events is independent on amount on generated time series. Talking about requirements it is important to mention here that there is possibility to build joint metrics namely build multiple time series using this kind of metric:

**Counter** The counter is a metric is representing a simple numerical value which can be incremented but not inverse. One of the typical examples is a number of expectations to be occurred.

**Gauge** The gauge is a metric, which also represents a simple numerical value like counter, but it is bidirectional. It means that this value can be decremented. The common example is CPU usage, which can go up and down.

**Histogram** The histogram is a metric, which represent observations. It is stored as a bucket, which can be pulled. Any bucket can be configured depending on the need. It can be sum of values or count of events, which are observed.

**Summary** The summary is a metric, which is similar to histogram but it calculates configurable quantities.

### 3.3.3 Integration with N2Sky

Prometheus supports query language, which is a key feature for this tool. The Prometheus query language, or promql, is an expressive. With Prometheus the self-described metric name can be choose. Prometheus converts all metric so that every human can understand what exactly particular metric means. Lets take a previous example with a metric "node\_filesystem\_avail". This metric will show the folders on root and available memory on each of it.

```

1      node_filesystem_avail
2          { device="/dev/sdb4",fstype="ext4",
3            instance="localhost:9100",job="node",
4            ssmountpoint="/usr/local"}
```

Following request means that on "/usr/local" 186.6 GB is available.

There is also possibility to check a response code, which especially useful for alerting.

```

1      node_filesystem_avail {status="500"}
```

This request returns some response code 500 namely internal server error.

For building proper dashboard for monitoring it is important to provide customisation that is why Prometheus supports time duration:

- s - seconds
- m - minutes
- h - hours
- d - days
- w - weeks
- y - years

Using the time duration with an offset it is possible to get exact metric on demand. Building query with a Prometheus can bring lots of advantages. For example there is query which use counter with a available node file system metric:

```

1      took( 3, sum(
2          rate(api_http_requests_total{status=500}[1h]
3      ) )
4      by (endpoint)
5      )

```

This query is already complicated, but it can be extended by multiple new rules and constrains.

In N2Sky was developed monitoring service, which use microservices approach like in entire application. It was decided to get rid of complex queries and provide some intuitive way of creating metrics. First of all the time range add a complexity. It was decided that the user should give only time interval and step. Lets say the user want to see CPU load for a last hour with a step 30 seconds. It makes creation of metric more intuitive, no more range like "from", "to" and type of ranges. All this can be solved with a one simple request. Second part is a storing of metric. Instead of every time build a query the monitoring service saving requested by user metric. In this case every user will get his own customised metric. The service uses Mongo DB for storing the metric configuration. Every collection has it own schema. When a user make a request to save a metric the schema have to be filled with a requested by user data.

### 3.3.4 Monitoring dashlet Design

Since there are multiple machine and services to monitor there was need to create a dedicated dashboard design. At first lets take a look on the environments we have to monitor:

**Openstack Machine.** It is dedicated machine, our cloud base for development and running instances.

**Openstack Instances.** Virtual machines with a different OS.

**Docker Containers.** Virtual machines in the Openstack instances.

One of the most important part in application design is to maximise reusability of the components.

“Fig. 32” shows http request directions in milliseconds metrics. Each dashlet item contains:

- Title, which represents readable and self-described metric name.
- Server, which shows to wich instance this metric belongs.
- Chart is the monitoring data itself.
- Tooltip, which appear on metric mouse over. Tooltip shows following data:

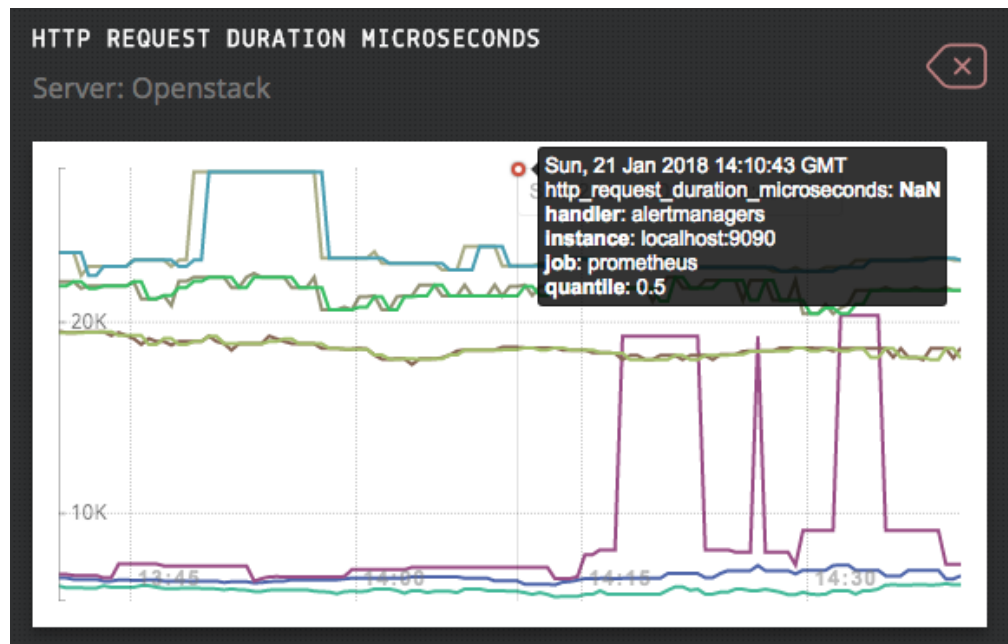


Figure 32: N2Sky Monitoring Dashlet

- Date of particular position.
- Name of metric
- Instance environment
- Handler is a alert manager listener (reference to subsection 3.4)
- Cron job
- Quintile

### 3.4 Alerting Management System

Today the most trending subject in monitoring area is a prediction and automated detection. It makes people free from 24/7 managing, maintaining and monitoring system. For monitoring we are using Prometheus tool. Since this tool saving constantly log data about system it is possible to reuse this logs to build an alert system. Prometheus tool provides an Alert Manager module. Natively this tool supports different notification methods like email notification or some request on Slack.

#### 3.4.1 Alerting System Architecture

Since the Alert Manager is a part of Prometheus Tool it has its own binary. The idea behind is to have only one Alert Manager and have monitoring tool on multiple machines. If machine goes down or even Prometheus itself, the Alert Manager can catch and deliver this event. To understand how Alert Manager works it is essential to understand the architecture of the whole system “Fig. 33”.



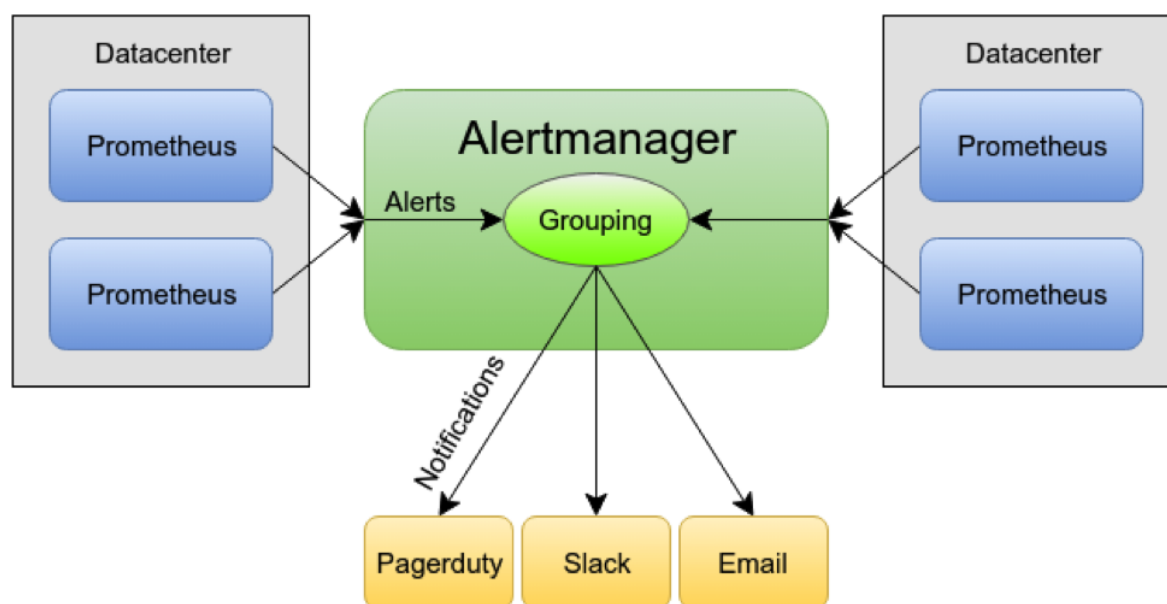


Figure 33: Alerting System Architecture

This architecture is a typical messaging platform. Messaging Service send messages between multiple clients. It is implement Producer-Consumer Pattern. In the Alert Manager Architecture the role of producer is taken by Prometheus Datacenter. Alert Manager consumes the messages. Consumer knows nothing about producer and just subscribe on event. With this approach it is possible to attach multiple producers [20].

Alerts can be collected in groups by datacenter. It means if an event occurs on multiple machines it can be packed into one notification and fired accordingly. In Prometheus configuration need to be setup only two things: reference on Alert Manager and Alert Rules. When Alert Manager consume an event it just dispatch it via notification “Fig. 34”..

**API** Alert manager API has only one endpoint, which gives list of events that occurred.

```
1 /api/v1/alerts
```

As a response the event information is sent [14].

```

1 [
2   {
3     "labels": {
4       "<label>": "<value>"
5     },
6     "annotations": {
7       "<label>": "<value>",
8     },
9     "startsAt": "<date>",
10    "endsAt": "<date>"
11    "generatorURL": "<url>"

```

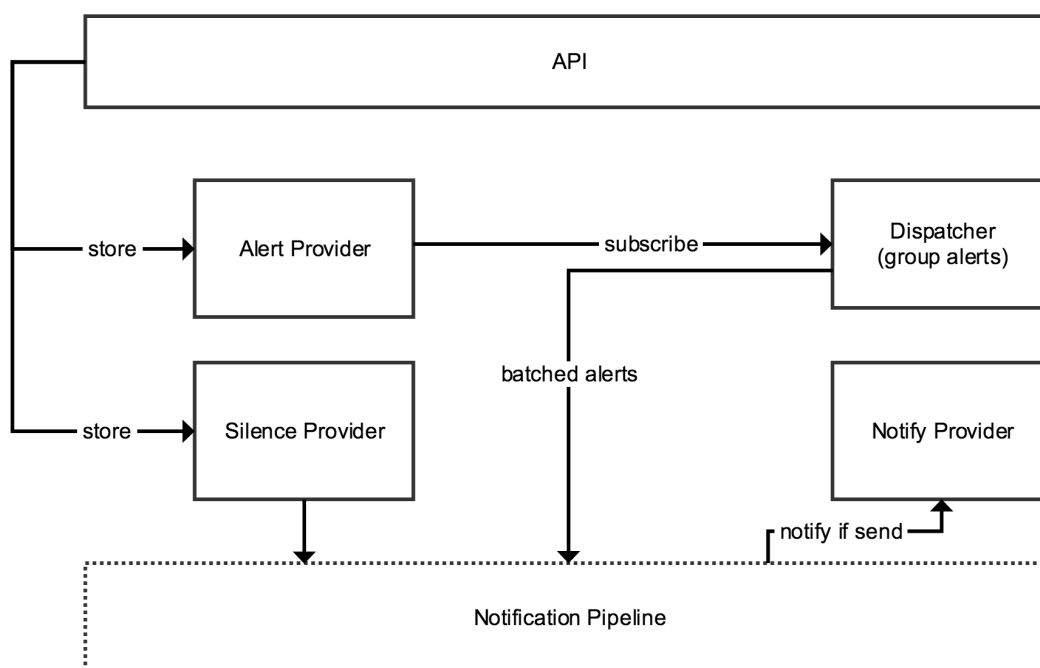


Figure 34: Communication within Alerting Management System

```

12   }
13 ]

```

Following response shows timestamp of event, additional information as an annotation and name of event (alert).

**Silences.** Silences is a commands, which are muting alerts for specific time. It can be configured via web interface.

**Dispatcher.** Dispatcher is a grouping of alert with a similar nature into single notification. This will be send as a batch to Notification pipeline.

**Notification Pipeline.** Is a pipepline, which consists channel, router and filter. It takes rules from Alert Provider and Silence Provider and alerts from Dispatcher. If notification is ready it can be send.

### 3.4.2 Alert Rules

As it was mention one of the main changes in Prometheus setup is to configure the alert rules. Every Prometheus Monitoring Tool can have it own alerting rules, which can be defined. There is also a possibility to reference on some common alerting rule for every monitoring system on every machine.

```

1      groups :
2      - name: test
3        rules :
4      - alert: Error
5        expr: job:request_latency_seconds:mean5m
6          {job="localhost:5000"} > 0.5
7        for: 5m
8        labels :
9          severity: page
10       annotations :
11       summary: latency

```

Following example of alerting rules shows the typical alerting rule. Most important parts are expression, which is applied on Monitoring System and severity level. More details about creation of alerting rules is located in Development Guide chapter "Setting up Alert Management System" subsection 9.1.7.

Alerting rules are instructions to the monitoring system it can be user as well for alerting as for recording.

Recording rules allows to pre-compute frequently needed expressions or expressions, which are resource or time consuming. These rules are saving result in a new set of time series. It is like indexing this data, so that prevent expansive I/O methods. The rules are being executed sequentially with a predefined interval. With a alerting rules it is possible to define alert namely deviation by particular expression from Prometheus Tool and its exports (modules). It allows building an alert even on combined query.

In case if Alert Manager is not available all alerts are saving into buffer. As soon Alert Manager online all events will be fired sequentially.

### 3.4.3 Integration with N2Sky

Alerting System is represented as a module in N2Sky. Alerting Client is the additional configuration upon Prometheus Monitoring System. When Prometheus will be executed it should have reference on Alerting System and its rules. Since the client should be installed on each OpenStack instance it was integrated into image snapshot. When the new instance will be spawned with a OpenStack snapshot, the client will be automatically executed there.

Alerting Client fire alerts depending on configured rules. The rules can be created via user interface. The list of events N2Sky receive via its service, which fetch information from Alert System.

### 3.4.4 Alerting System Design

Developing Alerting System UI it was keep the same design as in entire application. The N2Sky UI components were reused and none of new components were created.

The main component is a grid item which contains information about event, which is occurred “Fig. 35”.

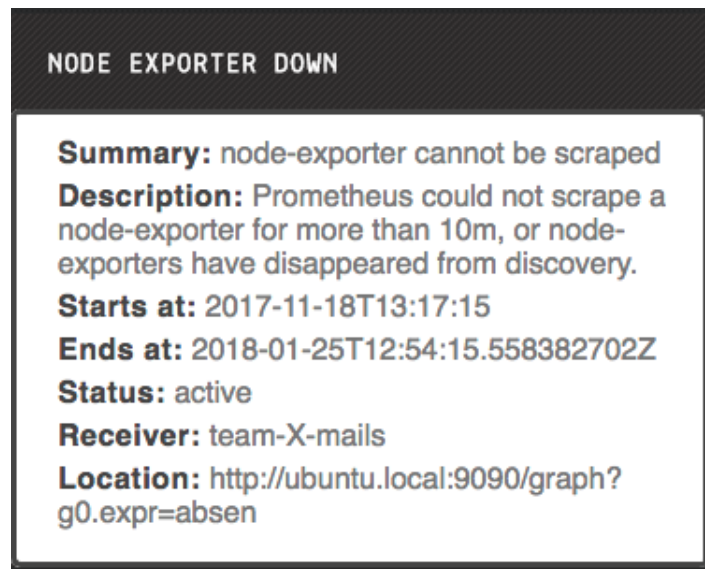


Figure 35: N2Sky Alerting Management System Event representation.

Following information is shown:

**Title.** The name of alert.

**Summary.** Short is a description of alert. If summary is too long it will be cut.

**Description.** The whole description of alert.

**Starts at.** Timestamp when event occurred.

**Ends at.** Timestamp when event is no more valid. If the timestamp is a current day and time, then problem still not fixed.

**Status.** Actual status of event. Can be active and inactive.

**Receiver.** The group of receivers. Normally group contains multiple receivers emails.

**Location.** The endpoint of server where monitoring system installed.

Every alert has its severity level. Depending on it the fired events will be represented differently. Every severity level has own colour on N2Sky as it shown in “Fig. 36”.

There are three types of severity levels:

- Critical, which shows that crucial event is occurred. For example if server goes down it is critical severity level. In N2Sky UI it has red colour.
- Warning, which shows that something goes wrong. For example not enough disk space on server. In N2Sky UI it has orange colour.

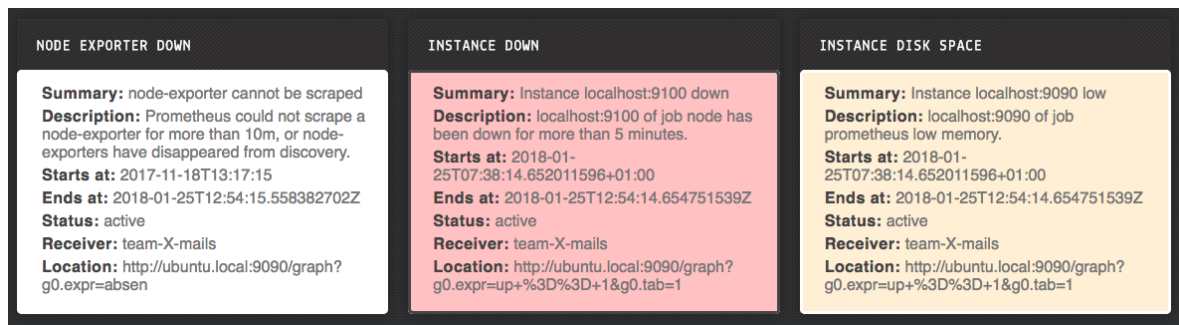


Figure 36: N2Sky Alerting Management System Severity Level

- Page, which shows that some information. For example lots of request occurred. In N2Sky UI it has white colour.

## 4 ViNNSL 2.0 extended

## 5 Requirements specification for Administration Module

### 5.1 General Definition

Administration Module Component (AMC) is an application, which is responsible for managing OpenStack and Cloudify. It embeds Monitoring System and Alert Management System. AMC is integrated in N2Sky cloud platform in order to support N2Sky environment and services. This component is implementing Platform as a service approach, hence it can be installed on any cloud server.

### 5.2 Affected users

The users, who has full knowledge about domain with correspondent permissions or domain owner can has an access to AMC.

In N2Sky only System Administrator has a proper main function in order to manage AMC. Following functions are available for System Administrator:

- Mange own dashboard view
- Control OpenStack Dashboard and its services
- Control Cloudify Dashboard and its services
- Manage monitoring charts
- Manage alerts and alerting rules

- Managet OpenStack instances (servers)
- Has an access to dashboards of other users

### 5.3 Administration Dashboard

Administration Dashboard represents overview of administration tools and important metrics, which is displayed in “Fig. 37”.

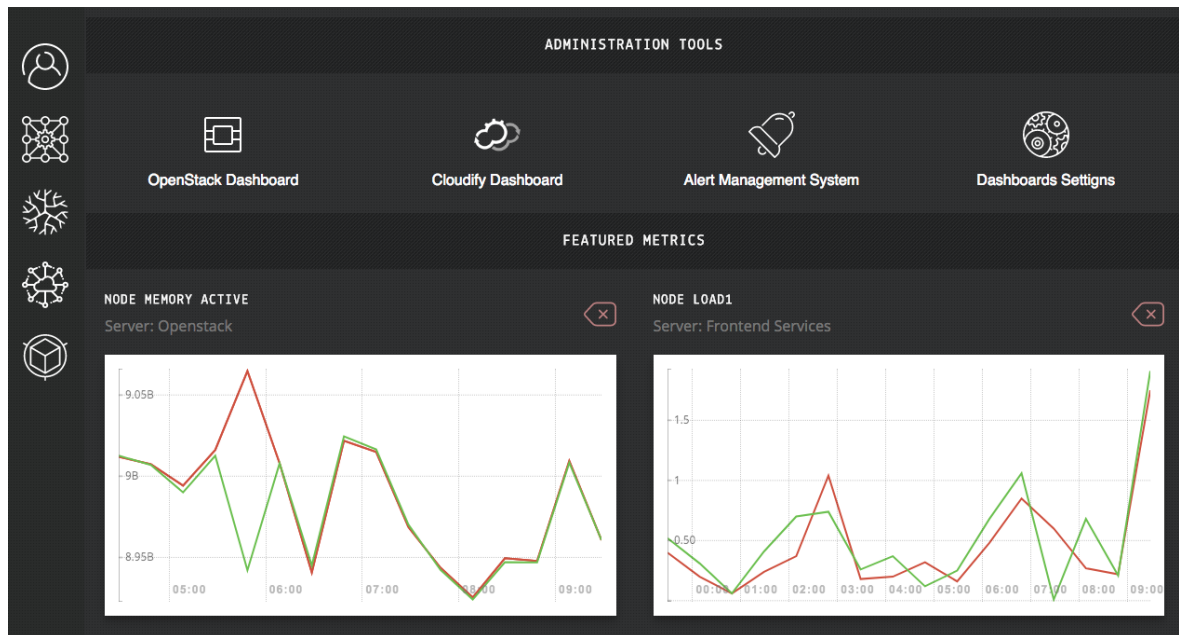


Figure 37: N2Sky Administration Dashboard.

After System Administrator login into system he will be automatically redirected to Administration Dashboard. By default url link is:

1 `<host>/cloud`

Administration tools has to be in focus of main dashboard view as it shown in “Fig. 38”. The collection of elements has to be under the title bar element, which has name "ADMINISTRATION TOOLS". Every element of administration tool has to contain icon in SVG [5] format and caption under it. Following administration tools has to be represented:

- OpenStack Dashboard. Link to view:

1 `<host>/openstack`

- Cloudify Dashboard. Link to view:

1 `<host>/cloudify`

- Alert Management System. Link to view:

1 `<host>/alert`

- Dashboards Settings. Modal popup window without redirection

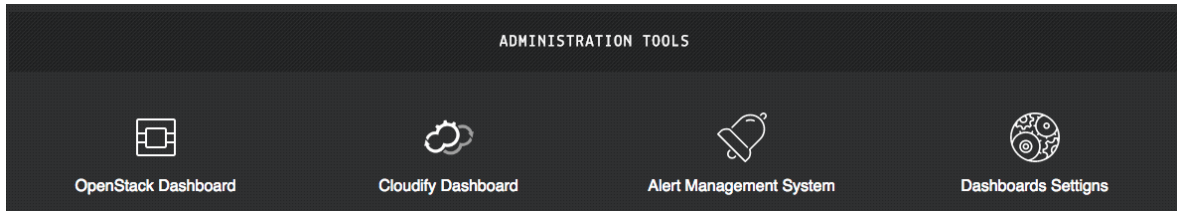


Figure 38: N2Sky Administration Dashboard. Administration tools component.

Next to administration tools the monitoring charts are showing. The title of this block is "FEATURED METRICS", which is the tile of bar upon the metrics. Every chart is a grid item element, the grid itself has to contain maximum two grid items in one row as it shown in "Fig. 39".

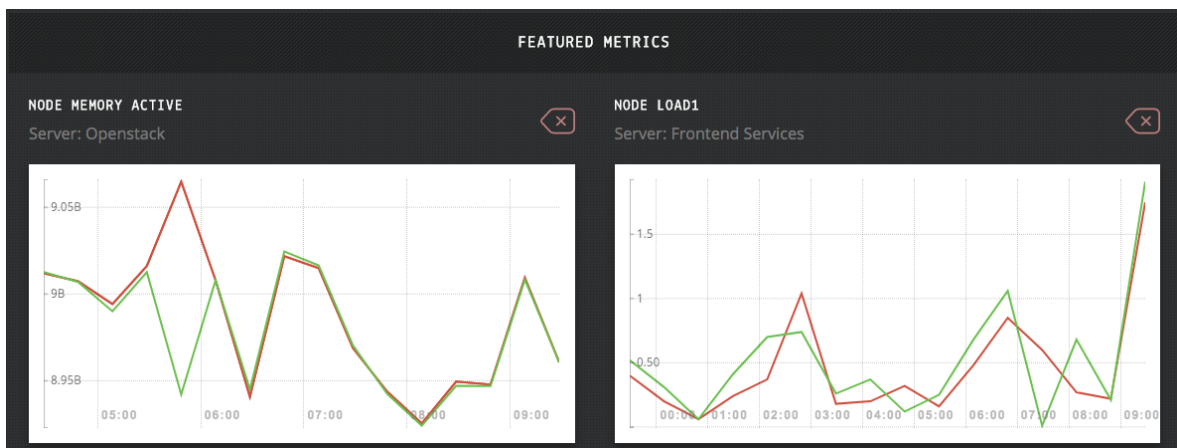


Figure 39: N2Sky Administration Dashboard. Featured Metrics component.

The user decides by him self which monitoring charts will be shown. The configuration is located in "Dashboard Settings" administration tool element. More details about creation of monitoring charts is located in subsection 5.7.

## 5.4 Openstack Dashboard

OpenStack Dashboard is the dashboard for managing OpenStack and its services and monitoring OpenStack and its instances. It looks similar to Administration Dashboard, except it is only OpenStack oriented and there is not additional tools as it shown in "Fig. 40".

This dashboard contains two blocks:

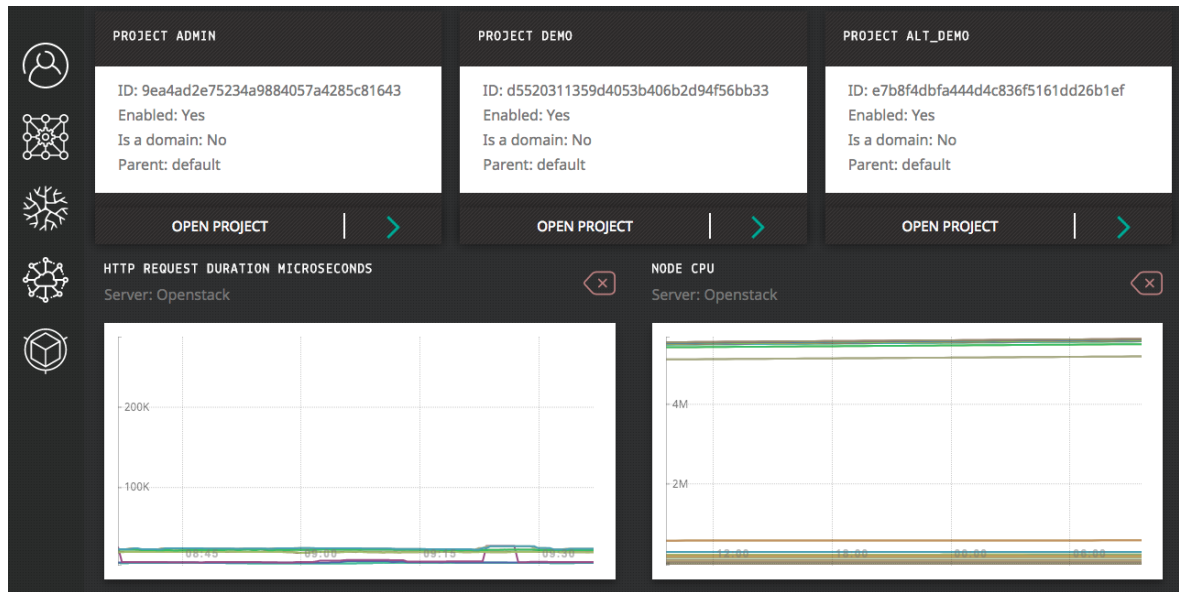


Figure 40: N2Sky OpenStack Dashboard.

**Project grid.** This block contains projects, which are available on OpenStack as it shown in “Fig. 42”.

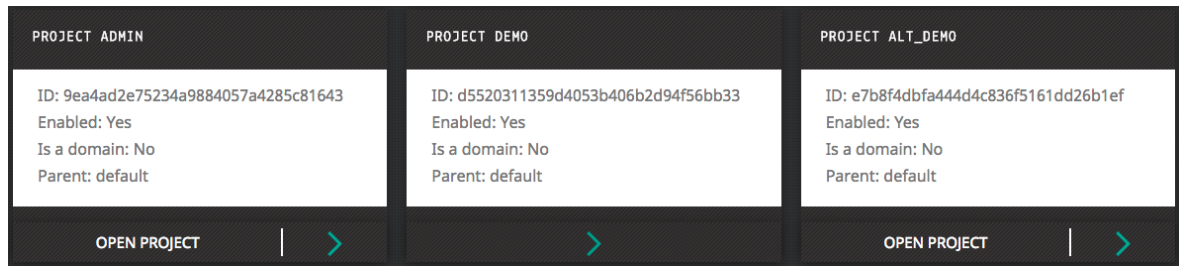


Figure 41: N2Sky OpenStack Dashboard. Projects grid

Every grid item represents brief overview about OpenStack Project:

- Name of the project, which is header of grid item.
- ID of project.
- Enabled, which stays if available (value YES) or not available (value NO).
- Is a domain, also contains simple yes or no values.
- Parent, which says if there any parent project which is current project linked.
- Button "Open Project", which redirects to project details.

**Monitoring Grid.** Similar to Administration Dashboard monitoring grid can be customised by users. Detail information about customisation is written in subsection 5.7.



After redirecting to project details the user will see information about specific project. Redirection should have following URL path:

1 `<host>/openstack/project/<id>`

Where `<id>` is OpenStack project ID.

### 5.4.1 OpenStack Nova Service

Nova is a compute service of OpenStack. It gives overview and managing of all existing virtual machine (servers or instances) [11].

Following tools for using compute service:

- Horizon. Web UI for managing OpenStack projects. It is not used in N2Sky system since N2Sky Web UI has its own interface for managing the OpenStack instances.
- OpenStack Client, which includes commands for nova as well as for OpenStack projects.
- Nova Client, which can be used like OpenStack Client, but N2Sky does not use it since OpenStack Client can provide all needed commands for nova service.

OpenStack project view has detailed description about server and its instances as it shown in “Fig. 42”.

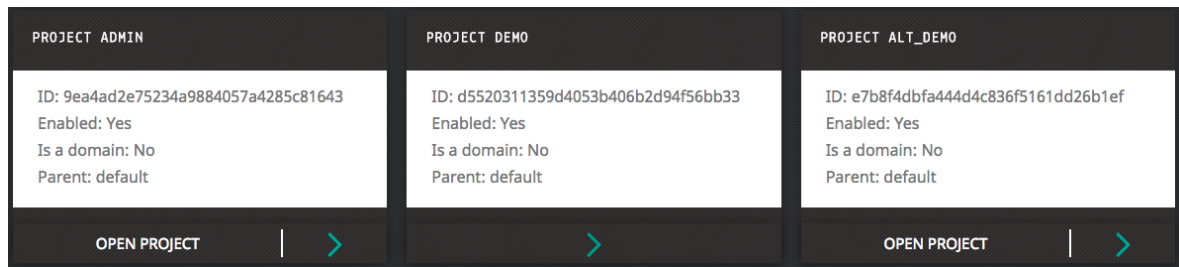


Figure 42: N2Sky OpenStack Dashboard. Project view

This view contains following components:

**Navigation Bar.** Navigation over OpenStack Services namely:

- NOVA
- NEUTRON
- IMAGES
- VITRAGE

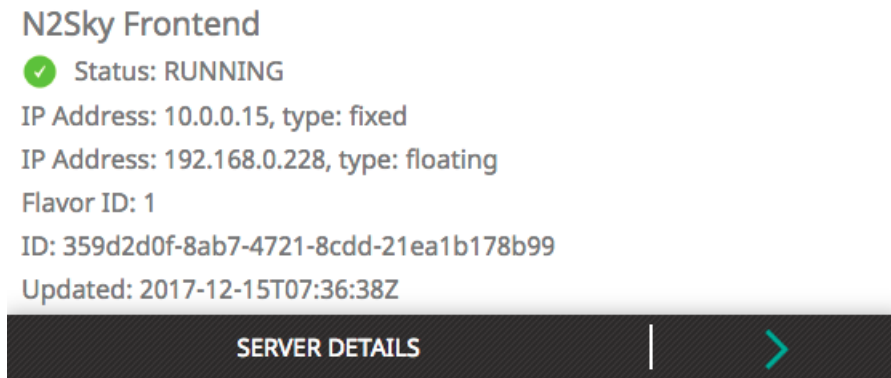


Figure 43: N2Sky OpenStack Dashboard. Server overview

**Servers.** Servers item grid is located on the left size of the screen and has an collection of servers (OpenStack instances) as it shown in “Fig. 43”.

The server overview grid contains following elements:

- Server name
- Status, which can be:
  - RUNNING - instance is running and available.
  - SHUTOFF - instance is manually or by scheduler shuted down.
  - ERROR - an error occur during spawning an instance or instance went down during run.
- IP Address fixed. The IP Address of instance inside OpenStack Cloud.
- IP Address floating. Assigned to instance IP address. Through this IP address an external access to instance is possible.
- Flavor ID. Reference on OpenStack flavor, which is located on the right side of OpenStack project view.
- ID of server (OpenStack instance).
- Updated is timestamp when instance was last time modified.

**Flavors.** A flavor is define an environment configuration. In the project details view there are list of flavors. On click on particular flavor of this list, the flavor details will appear as it shown in “Fig. 44”.

Following elements are displayed:

- Type of flavor, which is defined by OpenStack.
- ID of flavor. OpenStack instances have a reference on particular flavor ID
- RAM. Amount of available memory of particular flavor.
- Disk space in GB, which is available for particular flavor.

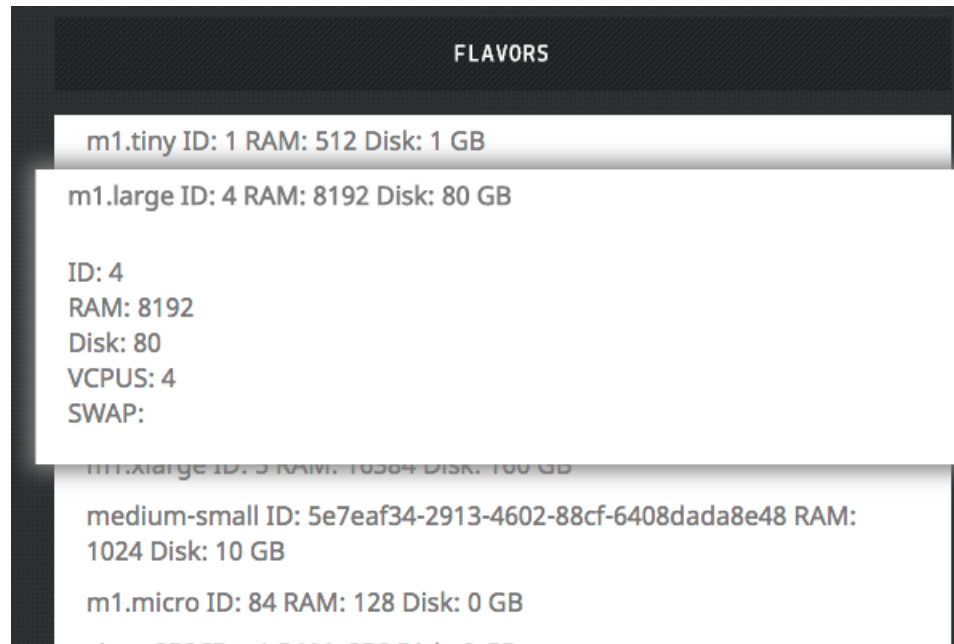


Figure 44: N2Sky OpenStack Dashboard. Flavors

- VCPUS. About of CPUs which will be available.
- SWAP in GB, which is set for particular flavor.

When user choose some server (instance), he will be redirected to server details view as it shown in “Fig. 45”.

#### 5.4.2 OpenStack Neutron Service

Neutron is an OpenStack networking service, which provides "network connectivity as a service". Neutron is fully integrated in OpenStack UI, which allows to manage networking directly from there. The networking service is based on quantum architecture, where API clients communicate with virtual switches through Quantum APU and Quantum plugin [6]. This service implements Neutron API, which is used by N2Sky Web UI.

Neutron Service is integrated in N2Sky and represents overview of networks, subnet pools and services providers as it shown in “Fig. 46”.

The server details view contains following components:

**Header.** Header of this view contains the name of the server, icon and status, which can be "Running" if instance is available and running on OpenStack environment, "Shutdown" if instance is available but not running and "Error" in case if an error occurred.

**Instance information grid .** Following grid represents summary information about server (instance) and consist of following grid items:

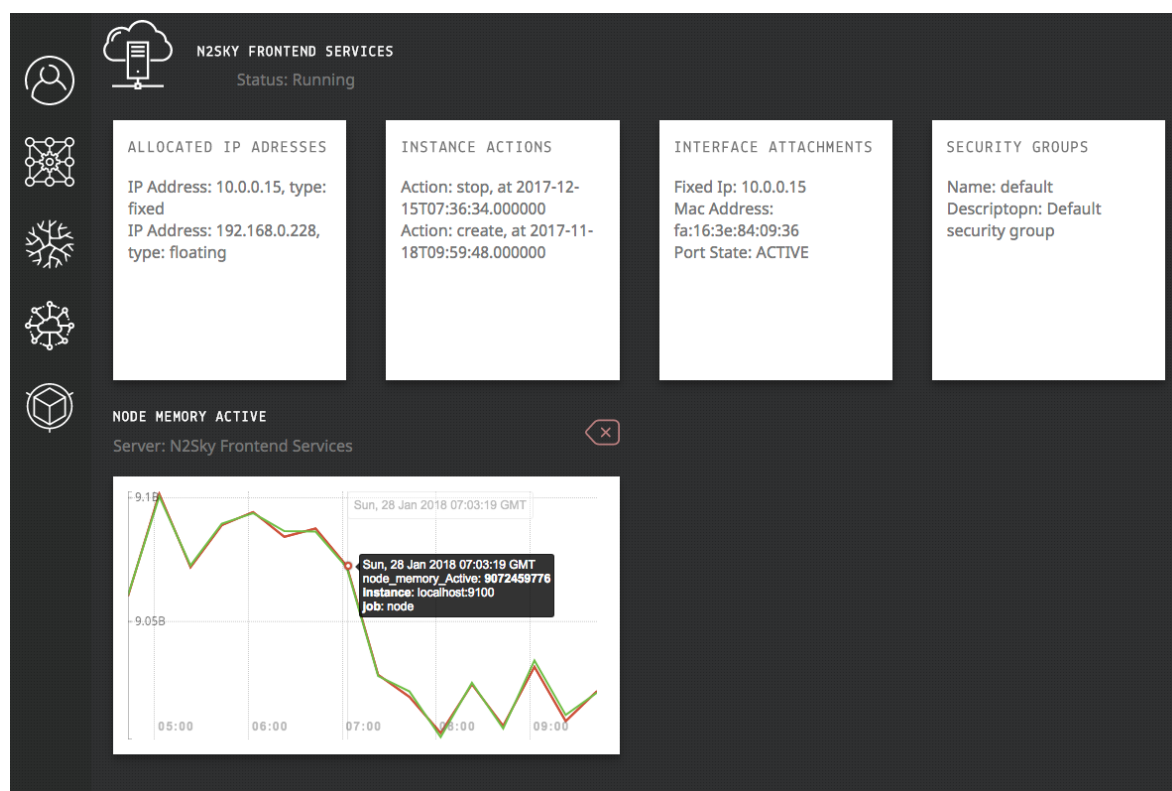


Figure 45: N2Sky OpenStack Dashboard. Server details view.

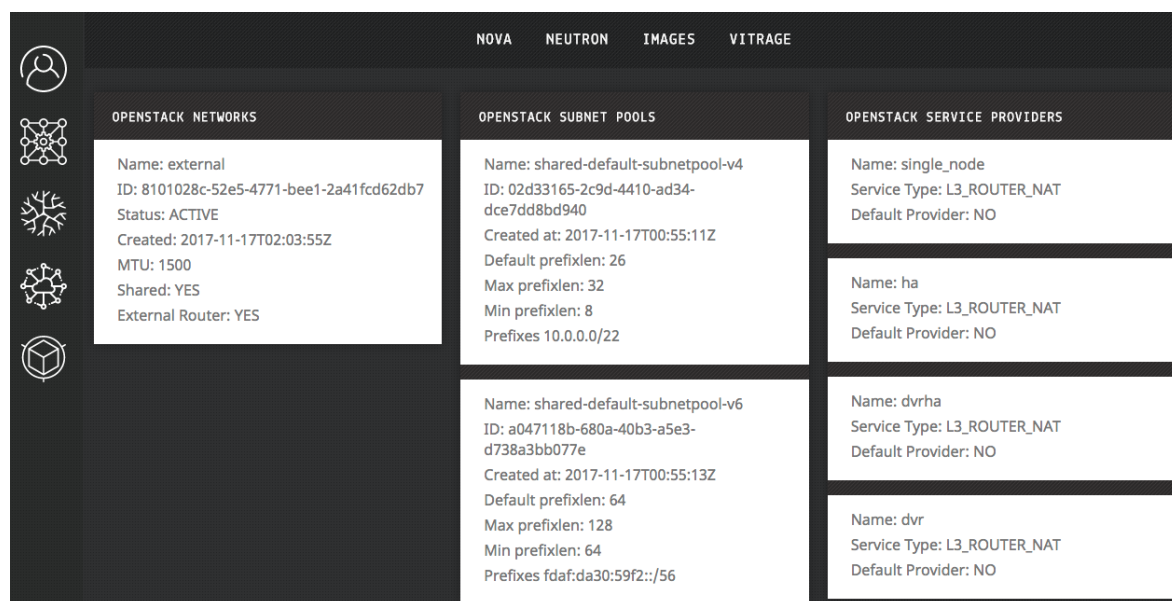


Figure 46: N2Sky OpenStack Dashboard. Neutron Service View

- Allocated IP Addresses is a list of assigned to instance IP addresses. Every IP address has a type either "fixed" or "floating".
- Instance actions is a list of all action and events, which happened with particular instance. Log information contains action type and timestamp of action.
- Interface attachments, which contains fixed IP address, mac address and port state ("ACTIVE" or "INACTIVE") of the instance.
- Security groups is a list of security groups which contains the name and description of security group .

**Monitoring.** The grid of monitoring charts of particular server. Can be removed directly from the view or added via navigation menu or Administration Dashboard.

The grid represents following components besides navigation bar:

**OpenStack Networks.** The list of available networks on OpenStack cloud. Every network contains following information:

1. Name of the network
2. ID of the network
3. Status, which can be "ACTIVE" network is available and "INACTIVE" is it not available or an error occurred.
4. Created is timestamp of creation of the network
5. MTU (Network Service Uses), which are based on physical network in order to calculate MTU of virtual network components.
6. Shared. "YES" for shared access and "NO" for not shared accordingly.
7. External router. "YES" for external router is attached and "NO" for not attached accordingly.

**OpenStack Subnet Pools.** This service provides information about available subnetworks and includes following information on N2Sky application:

1. Name of the subnetwork
2. ID of the subnetwork
3. Created at is the timestamp of subnetwork creation
4. Default number prefixlen
5. Max number of prefixlen
6. Min number of prefixlen
7. Prefixlen, which represents a pool prefix

**OpenStack Service Providers.** The list of created and available service providers with customised configuration, which contains:

1. Name of service provider
2. Service type
3. Default provider. "YES" if exist, "NO" if not accordantly.

### 5.4.3 OpenStack Images Service

In OpenStack is possible to upload customised images of the Virtual Machine to the provider. N2Sky uses Images Service only for representation of images information. It also possible to download image, but it is not possible to upload customised image due expensive process, because of the size of images. It is possible to use template provider with pre-configured virtual machine [22].

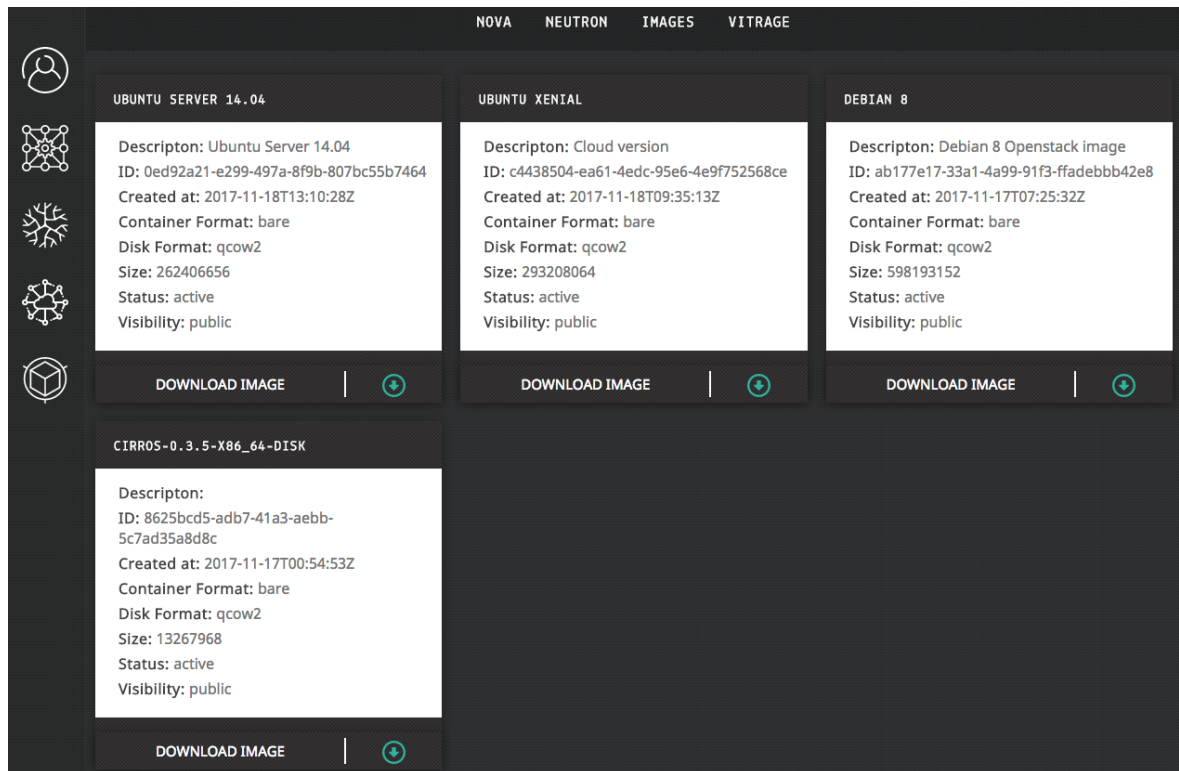


Figure 47: N2Sky OpenStack Dashboard. Images Service View

The Image Service view, which shown in “Fig. 47” is representing a grid of available images. Every grid item contains following information:

1. Name of the image
2. Description of the image
3. ID of the image

4. Created at, which represents the timestamp of the creation date of the image
5. Container format
6. Disk format namely format of the image (qcow2 is a typical format for OpenStack instance)
7. Size of the image in bytes
8. Status of the image. Active if image is successfully upload and ready to use, inactive if it not
9. Visibility. Public for everyone or group name for particular group of OpenStack users
10. Download image button, which create one more thread an initialise downloading procedure

There are custom OpenStack image templates and configuration created for N2Sky system. Every image has to contain following configuration:

- Preinstalled Docker CLI
- N2Sky Monitoring System instance
- N2Sky Alert Management System rules configuration

N2Sky templates and configuration has to support following operating systems:

- Ubuntu 14.04 / 16.04
- Debian 8
- Centos

#### 5.4.4 OpenStack Vitrage Service

Vitrage is the OpenStack RCA (Root Cause Analysis) service. Is a build in monitoring system, which helps to analyse and organise OpenStack alarms and events [3]. In N2Sky platform Vitrage service is used only for representation of templates and resources, because N2Sky has its own Monitoring System, which can be propagate in all OpenStack instances.

Two column grid view which is shown in “Fig. 48” represents following grid items:

**OpenStack Templates.** List of OpenStack templates with following data:

1. Name of the template
2. ID of the template

The screenshot displays the N2Sky OpenStack Dashboard's Vitrage Service View. The interface is organized into two primary sections: 'OPENSTACK TEMPLATES' on the left and 'OPENSTACK RESOURCES' on the right. A sidebar on the far left contains navigation icons, and a top navigation bar includes tabs for 'NOVA', 'NEUTRON', 'IMAGES', and 'VITRAGE'.

**OPENSTACK TEMPLATES**

- execute\_mistral**  
ID: cecffe48-9599-47fc-99a1-7a45ced2c6be  
Status Details: Template validation is OK  
Status: pass  
Date: 2017-11-17T00:57:01Z
- host\_aodh\_alarm**  
ID: 9e737984-f280-4a55-8a3b-e9128e847c1a  
Status Details: Template validation is OK  
Status: pass  
Date: 2017-11-17T00:57:01Z
- corrupted\_template**  
ID: 0d30e575-7720-46d4-ab22-e11d0ccb4bd5  
Status Details: template\_id does not appear in the definition block. template id: alarm\_on\_host  
Status: failed  
Date: 2017-11-17T00:57:01Z
- first\_deduced\_alarm\_ever\_nagios**  
ID: c91b69c3-dc69-47a9-b3e2-a35599559bfe  
Status Details: Template validation is OK

**OPENSTACK RESOURCES**

- nova**  
ID: nova  
State: available  
Vitrage aggregated state: AVAILABLE  
Vitrage Category: RESOURCE  
Vitrage id: b896dddc-a43f-463f-a7f8-4fbff3e3ae32  
Vitrage Operational State: OK  
Vitrage Type: nova.zone
- ubuntu.local**  
ID: ubuntu.local  
State: available  
Vitrage aggregated state: AVAILABLE  
Vitrage Category: RESOURCE  
Vitrage id: 3b36873c-ed59-44ce-bafd-cff29d4189f2  
Vitrage Operational State: OK  
Vitrage Type: nova.host
- openstack.cluster**  
ID: OpenStack Cluster  
State: available  
Vitrage aggregated state: AVAILABLE  
Vitrage Category: RESOURCE  
Vitrage id: e1e7e8d7-5fd0-429c-81a0-f91842e924fb  
Vitrage Operational State: OK  
Vitrage Type: openstack.cluster

Figure 48: N2Sky OpenStack Dashboard. Vitrage Service View



3. Status Details, which shows if the template validated
4. Date, which shows timestamp of template creation
5. Template details button, which redirect in detailed information about chosen template

**OpenStack Resources.** List of available OpenStack resources, which contains following data:

1. Name of the resource
2. ID of the resource
3. State of the resource, can be available or not available
4. Vitrage aggregated state, can be available or not available
5. Vitrage Category, can be resource, alarm or other customised category
6. Vitrage ID
7. Vitrage Operational State
8. Vitrage Type

When user click on template details button he will be redirected on particular Vitrage template as it shown in “Fig. 49”. From here user can observe following information:

**Template Entities** . List of entities like alarm or resource. This grid item contains following data

1. Template ID (mandatory)
2. Name of the template (optional)
3. Template category (optional)

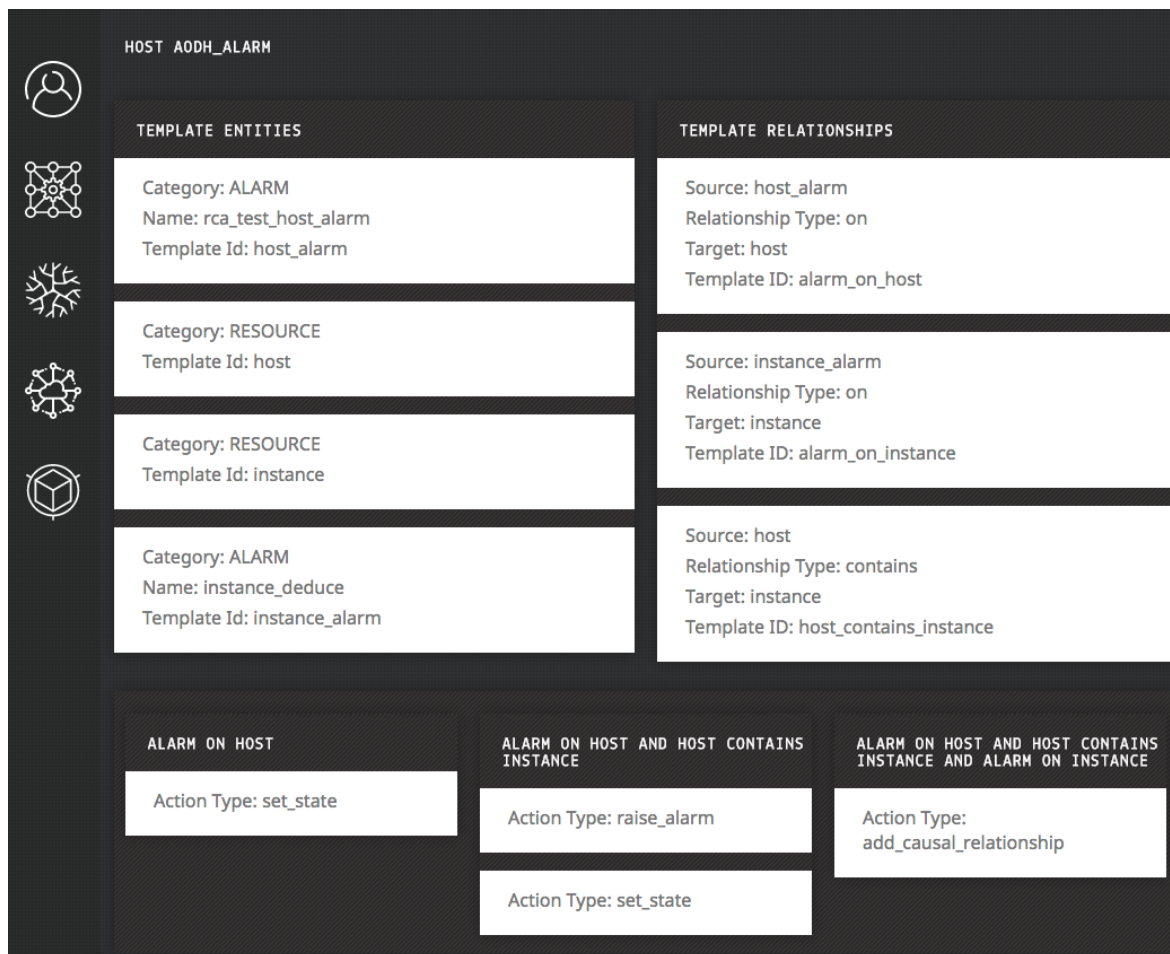
**Template Relationships** . List of relationships between templates, which reference from source to target.

1. Source entity
2. Target entity
3. Relationship type
4. Template ID

**Alarm On Host.** The action type of the alarm

**Alarm on host and host contains instance.** Optional item, visible only of host contains instance.

**Alarm on host and host contains instance and alarm on instance.** Optional item, visible only of host contains instance and alarm on instance.



The image shows a screenshot of the N2Sky OpenStack Dashboard, specifically the 'Template Details View' for a template named 'HOST AODH\_ALARM'. The interface is dark-themed with white text and icons. On the left, there is a vertical sidebar with five icons: a person, a network, a tree, a gear, and a cube. The main content area is divided into several sections. At the top, the title 'HOST AODH\_ALARM' is displayed. Below this, there are two main columns: 'TEMPLATE ENTITIES' and 'TEMPLATE RELATIONSHIPS'. The 'TEMPLATE ENTITIES' column contains four entries, each with a category, name, and template ID. The 'TEMPLATE RELATIONSHIPS' column contains three entries, each with a source, relationship type, target, and template ID. At the bottom, there are three columns representing different alarm types: 'ALARM ON HOST', 'ALARM ON HOST AND HOST CONTAINS INSTANCE', and 'ALARM ON HOST AND HOST CONTAINS INSTANCE AND ALARM ON INSTANCE'. Each column contains one or more entries with an 'Action Type'.

| HOST AODH_ALARM   |  |  |
|---|--|--|
| TEMPLATE ENTITIES   | TEMPLATE RELATIONSHIPS   |  |
| Category: ALARM<br>Name: rca_test_host_alarm<br>Template Id: host_alarm | Source: host_alarm<br>Relationship Type: on<br>Target: host<br>Template ID: alarm_on_host              |  |
| Category: RESOURCE<br>Template Id: host                                 | Source: instance_alarm<br>Relationship Type: on<br>Target: instance<br>Template ID: alarm_on_instance  |  |
| Category: RESOURCE<br>Template Id: instance                             | Source: host<br>Relationship Type: contains<br>Target: instance<br>Template ID: host_contains_instance |  |
| Category: ALARM<br>Name: instance_deduce<br>Template Id: instance_alarm |  |  |
| ALARM ON HOST   | ALARM ON HOST AND HOST CONTAINS INSTANCE   | ALARM ON HOST AND HOST CONTAINS INSTANCE AND ALARM ON INSTANCE |
| Action Type: set_state  | Action Type: raise_alarm   | Action Type: add_causal_relationship                           |
|   | Action Type: set_state   |  |

Figure 49: N2Sky OpenStack Dashboard. Template Details View

## 5.5 Cloudify Dashboard

TODO ????

## 5.6 Dashboard Settings

As it was mentioned before in Administration Dashboard chapter subsection 5.3, the dashboard has the administration tools. One of this tools is Dashboard Settings tool. When user click on this tool the modal popup window will open and available configuration options will appear as it shown in “Fig. 49”.

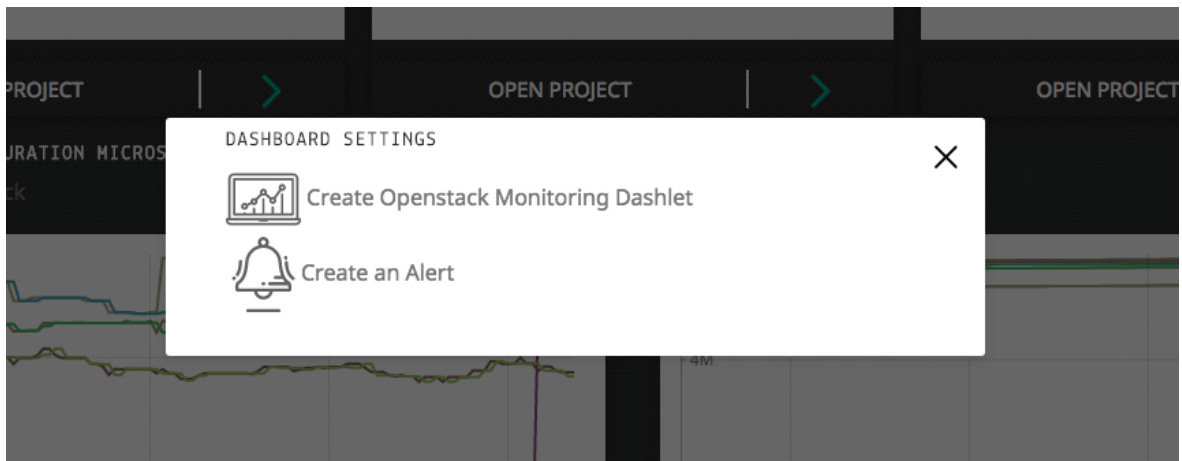


Figure 50: N2Sky Administration Dashboard. Dashboard Settings

Another one possibility to initiate settings popup is to open it from main navigation menu, which is available across entire application on the left screen of the page view.

Dashboard Settings popup contains two setting for now:

- Create OpenStack Monitoring Dashlet, which create one more modal popup upon existing and initialise creation monitoring chart, which is described in chapter "Monitoring System" in subsection 5.7.
- Create an Alert, which also create one more modal popup upon existing and initialise creation alerting rule, which is described in chapter "Alert System" in subsection 5.8.

The popup will be automatically closed only after action performed, if action is not performed user can leave the popup manually.

## 5.7 Monitoring System

When Administration Dashboard in general shows overview of the OpenStack and Cloudify, the monitoring system goes through every dashboard an apply monitoring

charts on user request. There are two main parts, which has to be mentioned in FRS: displaying of monitoring charts and management of metrics.

### 5.7.1 Monitoring charts creation

After user initialise creation of monitoring metrics at it was described in subsection 5.6 "Dashboard Settings", a new modal window will be opened as it shown in "Fig. 51".

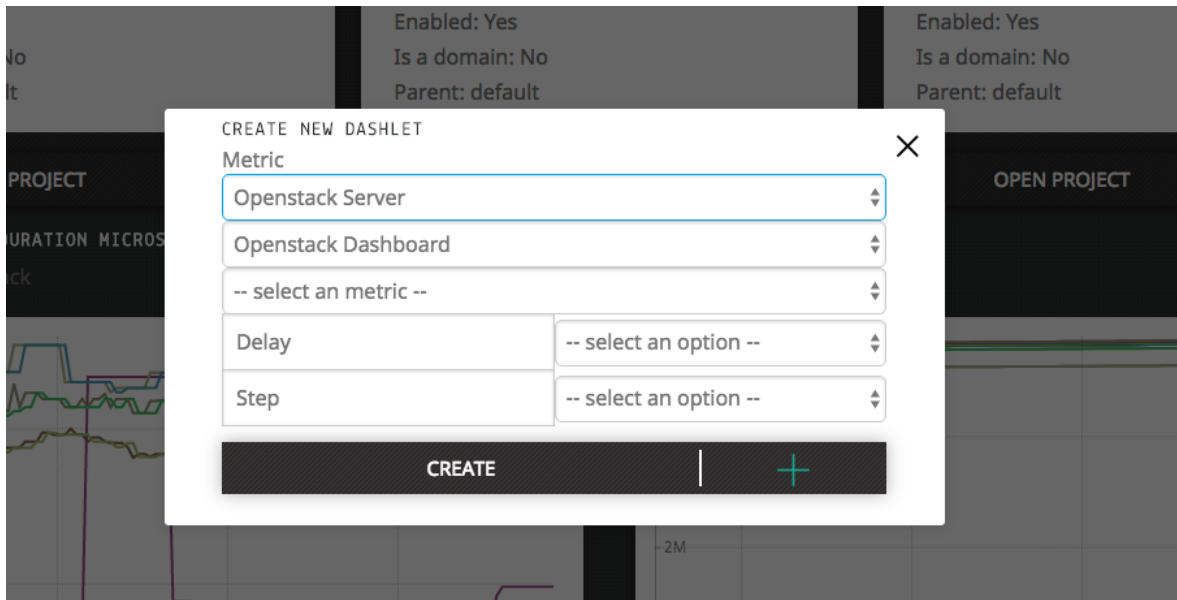


Figure 51: N2Sky Administration Dashboard. Monitoring charts creation modal popup

The popup window has following elements:

1. "CREATE NEW DASHLET" title of the popup
2. Combobox with servers, where monitoring system instance is installed. User can choose all available instances, even if they are not running currently. It is also possible to choose OpenStack cloud server itself, because monitoring system instance is preinstalled there.
3. Combobox with views. User can choose where combobox will be added. There are few places where chart can be displayed:
  - a) Administration Dashboard
  - b) OpenStack Dashboard
  - c) Cloudify Dashboard
  - d) Servers (OpenStack instances) View.
4. Combobox with metrics. Normally it is more then hundred of available metrics. It is important to mention that every operating system has different naming of metrics.

5. Delay input. This input shows the delayed time of metric.
6. Step input. This input shows the line chart step of chosen metric
7. Timing combobox. This combobox contains following time options:
  - a) seconds
  - b) minutes
  - c) hours
  - d) days
  - e) weeks
8. Create button, which trigger metrics creation. The view will be reloaded and monitoring chart will appear on this view.

After adding a new monitoring chart, the view will be reloaded and the chart will appear on the view.

### 5.7.2 Monitoring charts representation

Every monitoring chart is a grid item of two column responsive grid as it shown in “Fig. 52”

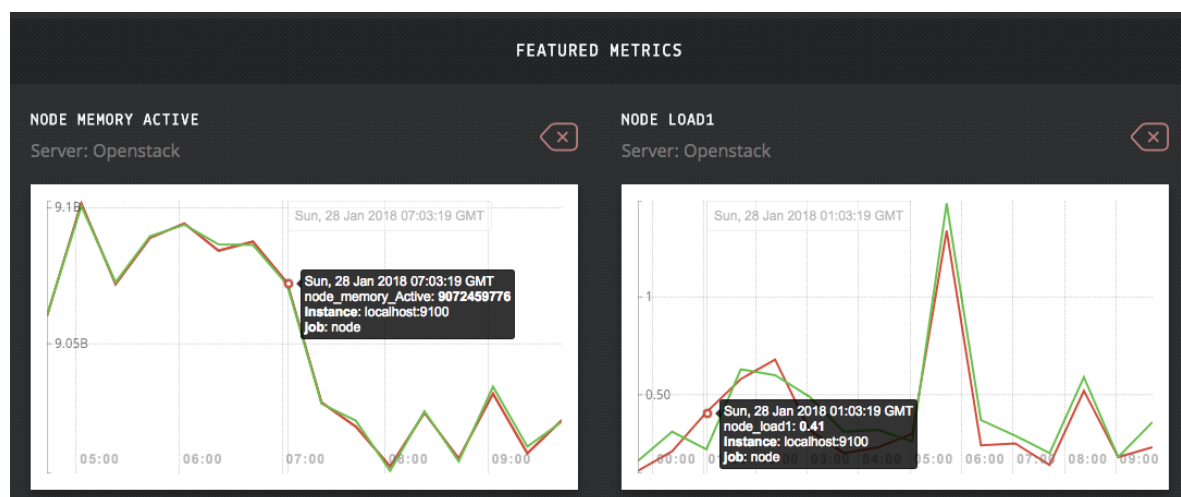


Figure 52: N2Sky Administration Dashboard. Monitoring charts representation

The grid has to be responsive in order to support mobile devices. When mobile device is detected the grid should have only one column and grid items have to be aligned vertically.

The header of monitoring chart has to contain the name of the metrics and the server (instance) from where this metrics comes.

Every monitoring chart has at list one line graph with a random colour. If there are few line graphs in the chart, then lines has to have a different colour. A chart contains on x-Axis the timestamp and on y-Axis the velocity. The timestamp in line graph has its own toolkit with additional information about the chart and contains following data:

- Timestamp of event
- Type of metrics and Id of timestamp
- Instance, where monitoring system is running
- The name of the job, which is responsible for particular event

## 5.8 Alert System

Alert System hat it is own page view. It is possible to redirect to this view from main navigation menu as well as from administration dashboard. The url of the alert page is:

1                                      <host>/alert

### 5.8.1 Alerting rule creation

It is possible to alerting rule directly from alert page view. The button "Create Alert" initiate the modal popup window as it shown in “Fig. 53”.

The modal window has following elements:

1. Modal window title is "Create new alert".
2. Combobox with available servers with installed monitoring system.
3. Combobox with a page view, where alert can be shown (optional).
4. Combobox with a available metrics.
5. Combobox wth alerting rule severity level. Three values are possible: "page" for information, "waring" and "critical".
6. Summary information about alerting rule.
7. Full description about alerting rule.
8. Expression which will be executed agains chosen metric.
9. Period, which shows how often will alerting rule validated.

The screenshot displays the 'CREATE NEW ALERT' modal form overlaid on the N2Sky Administration Dashboard. The form contains the following fields and controls:

- Alert Name:** A text input field.
- Openstack Server:** A dropdown menu with 'Openstack Server' selected.
- Overview Dashboard:** A dropdown menu with 'Overview Dashboard' selected.
- select an metric --:** A dropdown menu.
- select a severity --:** A dropdown menu.
- Summary:** A text input field.
- Description:** A text input field.
- Expression:** A text input field.
- Period:** A text input field.
- CREATE:** A button at the bottom left of the modal.
- +** A green plus icon button at the bottom right of the modal.

The background shows a list of alerts with details such as 'Instance localhost:9100', 'localhost:9100', and 'localhost:9090'.

Figure 53: N2Sky Administration Dashboard. Alerting rule creation

This screenshot is identical to Figure 53, showing the 'CREATE NEW ALERT' modal form in the N2Sky Administration Dashboard. The form includes fields for Alert Name, Openstack Server, Overview Dashboard, metric selection, severity selection, Summary, Description, Expression, and Period, along with 'CREATE' and '+' buttons.

Figure 54: N2Sky Administration Dashboard. Alerts representation

### 5.8.2 Alerts representation

The alerts are displayed three column grid as it shown in “Fig. 54”

The view contains:

**Header.** Header is navigation bar with a title "Fired alerts" and the button. The button has caption "Create Alert" and on click will initiate creating alert rule popup as it described in subsection 5.8.1 "Alerting rule creation".

**Alerts Grid** Contains 3 columns of alerts. When alert end date is earlier then current date, then alert will not be shown. Only active alerts are visible. Detailed information about alert content is described in subsection 3.4 "Alerting Management System".

## 6 Requirements specification for Main Application Module

### 6.1 General Definition

Main Application Module (MAM) is an application, which responsible for management N2Sky system. It embeds:

- Model Repository
- Neural Network Repository
- N2Sky Main Dashboard.

TODO

### 6.2 Affected users

The MAM has one User Interface but users can use it diverse. Only users, which main function is within MAM can operate this module on their own purpose. Following types of users have this kind of main function:

**Consumer.** The main function of consumer is to learn about neural networks or try out his knowledge in this field. Detailed description is in subsection 2.2.4.



## 6.3 N2Sky Dashboard

## 6.4 Neural Networks Repository

## 6.5 Models Repository

# 7 Tutorial

# 8 User Cases

# 9 Developer Guide

## 9.1 System configuration

### 9.1.1 Setting up the database

N2Sky uses non-relational database MongoDB, which is running in Docker container.

First of all the developer has to create the mongo database container:

```
1 docker run -d -p 27017:27017 --name database \
2     -v ~/dataMongo:/data/db mongo
```

It will start the mongo database Docker container on port 27017. It is possible to remap ports with the flag "-p".

The second step is to configure N2Sky database:

*Login into container:*

```
1 docker exec -it database mongo
```

*Create database superuser:*

```
1
2 db.createUser(
3     { user: 'admin', pwd: 'password',
4       roles: [
5         { role: "userAdminAnyDatabase", db: "admin" }
6       ] });
```

*Create database:*

```
1 use n2sky;
```

*Create database n2sky user:*

```
1
2 db.createUser({user:"n2sky", pwd:"password", roles:['dbOwner']})
```

*Exit from container and enable authentication:*

```
1
2 docker rm -f database
```

```

3
4     docker run -d -p 27017:27017 --name database \
5     -v ~/dataMongo:/data/db mongo mongod --auth

```

### 9.1.2 Setting up the Cloud Web Service

*Clone the project from the repository*

```
1 git clone https://github.com/latyaodessa/n2sky-services.git
```

*Go to service Dockerfile*

```
1 cd n2sky-services/services
```

*Build the Docker image*

Dockerfile:

```

1 FROM node:7
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 RUN npm install -g forever
6 COPY . /app
7 CMD forever -c 'node --harmony' server.js
8 EXPOSE 8080

```

Command:

```
1 docker build -t cloud:1 .
```

*Run the service Docker container*

```
1 docker run -d -p 9595:8080 --name cloud cloud:1
```

Important to check the host and the port of the database. If something goes wrong and the port or the host is changed in n2sky project developer can edit it. The rest api endpoint host located in *service/config/database.js*

### 9.1.3 Setting up the Model Repository Service

*Clone the project from the repository*

```
1 git clone https://github.com/CN2Sky/modelRepository.git
```

*Go to service Dockerfile*

```
1 cd modelRepository
```

*Build the Docker image*

Dockerfile:

```

1 FROM node:7
2 WORKDIR /app
3 COPY package.json .

```

```

4 RUN npm install
5 RUN npm install -g forever
6 COPY . /app
7 CMD forever -c 'node --harmony' server.js
8 EXPOSE 8080

```

Command:

```
1 docker build -t model:1 .
```

*Run the service Docker container*

```
1 docker run -d -p 9092:8080 --name model model:1
```

### 9.1.4 Setting up the User Management Service

*Clone the project from the repository*

```
1 git clone https://github.com/CN2Sky/user-management.git
```

*Go to service Dockerfile*

```
1 cd user-management
```

*Build the Docker image*

Dockerfile:

```

1 FROM node:7
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 RUN npm install -g forever
6 COPY . /app
7 CMD forever -c 'node --harmony' server.js
8 EXPOSE 8080

```

Command:

```
1 docker build -t user:1 .
```

*Run the service Docker container*

```
1 docker run -d -p 9091:8080 --name user user:1
```

### 9.1.5 Setting up the N2Sky frontend

*Clone the project from the repository*

```
1 git clone https://github.com/latyaodessa/n2sky-services.git
```

*Go to service Dockerfile*

```
1 cd n2sky-services/frontend
```

*Build the Docker image Dockerfile:*

```

1 FROM node:7
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 COPY . /app
6 CMD npm run dev
7 EXPOSE 9593

```

Command:

```
1 docker build -t frontend:1 .
```

*Run the service Docker container*

```
1 docker run -d -p 9593:9593 --name frontend frontend:1
```

### 9.1.6 Setting up the Monitoring System

In the global configuration is possible to setup scrape interval and evaluation interval.  
global:

```

1  scrape_interval:      15s
2  evaluation_interval: 15s

```

Prometheus has to reference on Alert Manager, where messages will be published.

```

1 alerting:
2   alertmanagers:
3     - static_configs:
4       - targets:
5         - localhost:9093

```

Every machine where Prometheus is installed can has its own alerting rules. In general alerting rules are located in the root folder of Prometheus.

```

1 rule_files:
2   - "alert.rules"
3   - "node.rules"
4   - "test.rules"

```

Since there is a need to get more specific data, in N2Sky was decided to user Node Exporter Module. The reference on this module has to be added into configuration

```

1 - job_name: 'node'
2   scrape_interval: 5s
3   target_groups:
4 -   targets: ['localhost:9100']

```

Node Exporter Module has no configuration file. Prometheus listen the modules and scrap the data with a defined interval.

For deploying alert manager Docker containers technology is used. **TODO**

### 9.1.7 Setting up Alert Management System

All configuration of alert manager are written in YAML file. On the beginning SMTP email sender should be configured. This would be used to sending notifications.

```
1 global:
2   smtp_smarthost: 'localhost:25'
3   smtp_from: 'alertmanager@example.org'
4   smtp_auth_username: 'alertmanager'
5   smtp_auth_password: 'password'
```

It is possible to define multiple Email templates and configure which template need to be loaded on which severe level. In configuration the path to templates need to be defined.

```
1 templates:
2 - '/etc/alertmanager/template/*.tmpl'
```

When alerts are consumed they need to be converted using Email template and fired to the particular route. Every route has a receiver.

```
1 route:
2 group_by: ['alertname', 'cluster', 'service']
3 group_wait: 30s
4 group_interval: 5m
5 repeat_interval: 3h
6 receiver: team-X-mails
```

**group\_by** Group by label. This way ensures that multiple alerts from difference cluster can be received

**group\_wait** Ensures that multiple alerts can be fired shortly after particular group is received.

**group\_interval** Interval between alert batches.

**Receiver** Unique name of receiver which is defined in configuration.

Receiver it is a group of matching by regular expression events.

```
1 routes:
2 - match_re:
3     service: ^(foo1|foo2|baz)
4     receiver: team-X-mails
```

Receiver can be defined by user configuration, it is an email where is alert notification will be send.

```
1 receivers:
2 - name: 'team-X-mails'
3   email_configs:
4     - to: 'team-X+alerts@example.org'
```

**How to write alerting rules** The alerting rules are supporting simple query language, which looks very similar to Sequel Query Language. There is multiple possibilities how work with a alerting rules. The query language allows to use an expression and as a result to check an attribute of time series.

```

1 ALERT HighLatency
2 IF api_http_request_latencies_second{quantile="0.7"} > 1
3 FOR 5m
4 LABELS { severity="critical"}
5 ANNOTATIONS { summary = "High latency detected ", description = "over limit? }

```

Following notations should be considered be creation of alerting rules:

- All queries staring with "ALERT" namespace. After it follows name of alert in this case it is "HighLatency".
- "IF" is a condition "api\_http\_request\_latencies\_second", which based on Prometheus Tool expression. Set of time series with this expression has one parameter it is "quantile". Reading condition as a whole can be translated in a human language like this: "Send a alert if latency request per second bigger then 0.7".
- "FOR" it is period of time how often this condition should be checked.
- "LABELS" shows a severity level. There are 3 types of severity:
  - Critical
  - Warning
  - Page
- Every severity level can be defined on developer needs.
- "ANNOTATIONS" shows a readable for human comments. There are two sub sections: summary, which shows a short description of the event and description where detailed information about deviation can be written

For deploying alert manager Docker containers technology is used. **TODO**

## 9.2 Continuous integration

## 9.3 API Documentation

### 9.3.1 N2Sky Monitoring System API Documentation

## Literaturverzeichnis

- [1] Systems and software engineering – vocabulary, 2010.
- [2] Typeface — Wikipedia, the free encyclopedia, 2016. [Online; accessed 20-December-2016].
- [3] What is Vitrage? — Wikipedia, the Free Encyclopedia, 2016.
- [4] Anusas-amornkul, T., and Sangrat, S. Linux server monitoring and self-healing system using nagios. In *Mobile Web and Intelligent Information Systems* (Cham, 2017), M. Younas, I. Awan, and I. Holubova, Eds., Springer International Publishing, pp. 290–302.
- [5] Cagle, K. *The Foundations of SVG*. Springer London, London, 2005, pp. 21–62.
- [6] Denton, J. *Learning OpenStack Networking (Neutron)*. Packt Publishing, 2014.
- [7] Goubali, O., Girard, P., Guittet, L., Bignon, A., Kesraoui, D., Berruet, P., and Bouillon, J.-F. Designing functional specifications for complex systems. In *Human-Computer Interaction. Theory, Design, Development and Practice* (Cham, 2016), M. Kurosu, Ed., Springer International Publishing, pp. 166–177.
- [8] Khalil, W. Reference architecture for virtual organization. *PhD thesis, University of Vienna 2012* (2012).
- [9] Li, F.-L., Horkoff, J., Borgida, A., Guizzardi, G., Liu, L., and Mylopoulos, J. From stakeholder requirements to formal specifications through refinement. In *Requirements Engineering: Foundation for Software Quality* (Cham, 2015), S. A. Fricker and K. Schneider, Eds., Springer International Publishing.
- [10] Macías, J. A., and Paternò, F. Intelligent support for end-user web interface customization. In *Engineering Interactive Systems* (Berlin, Heidelberg, 2008), J. Gulliksen, M. B. Harning, P. Palanque, G. C. van der Veer, and J. Wesson, Eds., Springer Berlin Heidelberg, pp. 303–320.
- [11] Markelov, A. *OpenStack Compute*. Apress, Berkeley, CA, 2016, pp. 65–86.
- [12] Martinez, J., Sottet, J.-S., Frey, A. G., Ziadi, T., Bissyandé, T., Vanderdonckt, J., Klein, J., and Le Traon, Y. *Variability Management and Assessment for User Interface Design*. Springer International Publishing, Cham, 2017, pp. 81–106.
- [13] Matt T. Proud, J. V. Overview. what is prometheus? — prometheus, 2017. [Online; accessed 2017].
- [14] Matt T. Proud, J. V. Sending alerts — prometheus, 2017. [Online; accessed 2017].
- [15] Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal 2014*, 239 (2014), 2.
- [16] Morgan, D. How mobile web traffic has affected screen resolution — spindogs, 2017. [Online; accessed 21-September-2017].

- [17] Paternò, F., Santoro, C., and Scordia, A. Preserving rich user interface state in web applications across various platforms. In *Engineering Interactive Systems* (Berlin, Heidelberg, 2008), P. Forbrig and F. Paternò, Eds., Springer Berlin Heidelberg, pp. 255–262.
- [18] Pleuß, A. Modeling the user interface of multimedia applications. In *Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2005), L. Briand and C. Williams, Eds., Springer Berlin Heidelberg, pp. 676–690.
- [19] Regula Stopper, R. S. Graphical User Interface layout and design, 2012.
- [20] Tai, S., and Rouvellou, I. Strategies for integrating messaging and distributed object transactions. In *Middleware 2000* (Berlin, Heidelberg, 2000), J. Sventek and G. Coulson, Eds., Springer Berlin Heidelberg, pp. 308–330.
- [21] Technologies, G. Clodify. [online], <http://docs.getcloudify.org/3.4.1/intro/what-is-cloudify/>, last visited January 2018, 2017.
- [22] Vakanas, L., Sotiriadis, S., and Petrakis, E. G. M. Implementing the cloud software to data approach for openstack environments. In *Adaptive Resource Management and Scheduling for Cloud Computing* (Cham, 2015), F. Pop and M. Potop-Butucaru, Eds., Springer International Publishing, pp. 103–118.
- [23] Walraven, S., Truyen, E., and Joosen, W. Comparing paas offerings in light of saas development. *Computing* 96, 8 (Aug 2014), 669–724.
- [24] Weyers, B., Bowen, J., Dix, A., and Palanque, P. The handbook of formal methods in human-computer interaction. In *The Handbook of Formal Methods in Human-Computer Interaction* (Cham, 2017), Springer International Publishing.
- [25] Zeng, Y., Gao, J., and Wu, C. Responsive web design and its use by an e-commerce website. In *Cross-Cultural Design* (Cham, 2014), P. L. P. Rau, Ed., Springer International Publishing, pp. 509–519.



## **Anhang**

