universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Functional User Interfaces for Controlling and Monitoring the N2Sky System and its Services“

verfasst von / submitted by

Andrii Fedorenko

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2018 / Vienna 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 926
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. techn. Erich Schikuta

Acknowledgements

I would first like to thank my thesis advisor Mr. Univ.-Prof. tit. Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta. The door to Prof. Schikuta office was always open whenever I ran into a trouble spot or had a question about my research or writing.

I would also like to acknowledge to collaborators of the Workflow Systems and Technology research group at University of Vienna for helping me with the cloud environment.

Finally, I must express my very profound gratitude to my parents and to my friend Mersi Stafa for providing me with unfailing support and continuous encouragement through the process of researching and writing this thesis.

Abstract

The master thesis encapsulates multiple topics in artificial neural networks, language specification for neural networks, cloud computing, microservices, system monitoring, web and infrastructure design. It is based on the concept of the implementation of the ViNNSL (Vienna Neural Network Specification Language) and on the original idea of the platform for the artificial neural network simulation environment N2Sky.

This thesis presents N2Sky as a reimagining novel system, which provides a virtual collaboration platform to the computational intelligence community. N2Sky is a cloud-based platform and it implements the Neural Network as a Service approach, which allows users to share and exchange neural network knowledge resources.

The user-centered intuitive design is a fundamental requirement for N2Sky. The platform supports different types of stakeholders and can be used on any device. For arbitrary users, N2Sky is a learning platform, which provides existing neural network solutions to a given problem. For neural network engineers, the system allows the creating and training of neural networks from existing paradigms and for contributors, developing own neural network paradigms and publishing them for the community. For every user is assigned a personal and customizable dashboard, so that he has its own way to interact with the N2Sky application.

The main purpose of the master thesis is to develop a production ready application. N2Sky implements microservices architecture on infrastructure as well as on application level. To make the system stable, a customized monitoring and alerting system was created, which allows the system administrator to control the environment directly from the N2Sky user interface. Additionally, it was decided to implement ViNNSL template in order to support transparent neural network workflows across the entire application.

Zusammenfassung

Die Masterarbeit umfasst mehrere Themen in künstlichen neuronalen Netzen, Sprachspezifikation für neuronale Netze, Cloud Computing, Microservices, Systemüberwachung, Web- und Infrastrukturdesign. Es basiert auf dem Konzept der Implementierung der ViNNSL (Vienna Neural Network Specification Language) und auf der ursprünglichen Idee der Plattform für die künstliche neuronale Netzwerk-Simulationsumgebung N2Sky.

Diese These stellt N2Sky als ein neuartiges System vor, das der Computational Intelligence Community eine virtuelle Kollaborationsplattform bietet. N2Sky ist eine Cloud-basierte Plattform und implementiert den Neural Network-as-a-Service Ansatz, der es Benutzern ermöglicht, neuronale Netzwerkwissensressourcen zu teilen und auszutauschen.

Das benutzerzentrierte intuitive Design ist eine grundlegende Voraussetzung für N2Sky. Die Plattform unterstützt verschiedene Arten von Stakeholdern und kann auf jedem Gerät verwendet werden. Für beliebige Benutzer ist N2Sky eine Lernplattform, die vorhandene neuronale Netzwerklösungen für ein gegebenes Problem bereitstellt. Für neuronale Netzwerkingenieure ermöglicht das System das Erstellen und Trainieren von neuronalen Netzwerken aus bestehenden Paradigmen und für Contributors, Entwickeln eigener neuronaler Netzwerkparadigmen und Veröffentlichen dieser für die Intelligence Community. Jedem Benutzer wird ein persönliches und anpassbares Dashboard zugewiesen, sodass er mit der N2Sky-Anwendung interagieren kann.

Der Hauptzweck der Masterarbeit ist die Entwicklung ein produktionsbereiten System. N2Sky implementiert Microservices-Architektur sowohl auf Infrastruktur- als auch auf Anwendungsebene. Um das System stabil zu halten, wurde ein angepasstes überwachungsund Alarmierungssystem erstellt, mit dem der Systemadministrator die Umgebung direkt von der N2Sky-Benutzeroberfläche steuern kann. Darüber hinaus wurde beschlossen, das ViNNSL-Template zu implementieren, um transparente neurale Netzwerk-Workflows in der gesamten Anwendung zu unterstützen.

Contents

1. Introduction **11**

1.1. Motivation . . . 11

1.2. Terms and Definitions . . . 12

1.3. Related Work . . . 12

2. N2Sky Architecture **15**

2.1. Current Architecture Analysis . . . 15

2.1.1. Architecture Design . . . 15

2.1.2. Components . . . 15

2.1.3. Implementation . . . 17

2.1.4. Usability and User Experience . . . 18

2.2. Redesign Motivation . . . 18

2.2.1. Redesign Process . . . 18

2.2.2. Refactoring the User Interface . . . 21

2.2.3. Advanced Project Management . . . 22

2.2.4. User Roles . . . 24

2.2.5. User Main Functions . . . 25

2.3. Novel N2Sky Architecture . . . 26

2.3.1. Cloud Infrastructure . . . 26

2.3.2. Modular Frontend Application Design . . . 28

2.3.3. The Sample Workflow . . . 31

2.3.4. Technology Stack . . . 34

3. N2Sky Components **37**

3.1. N2Sky Frontend Application . . . 37

3.1.1. N2Sky Layouts Design . . . 37

3.1.2. N2Sky Fundamental Layout . . . 38

3.1.3. N2Sky Main Layout . . . 39

3.1.4. N2Sky Mobile Layout . . . 40

3.1.5. N2Sky Page Views . . . 41

3.1.6. N2Sky Dashboards . . . 44

3.1.7. Responsive Design . . . 45

3.1.8. User Interface Elements . . . 46

3.1.9. UI Elements in N2Sky . . . 48

3.1.10. UI Components in N2Sky . . . 51

3.2. N2Sky Services . . . 54

3.2.1. User Management Web Service. . . . 55

3.2.2. Model Repository Web Service. . . . 56

3.2.3. Cloud Management Web Service. . . . 58

3.3. Continuous Monitoring System 60
3.3.1. Monitoring Requirements 60
3.3.2. Applying Monitoring 61
3.3.3. Integration with N2Sky 65
3.3.4. Monitoring Dashlet Design 67
3.4. Alerting Management System 68
3.4.1. Alerting System Architecture 68
3.4.2. Alert Rules 70
3.4.3. Integration with N2Sky 71
3.4.4. Alerting System Design 71

4. Vienna Neural Network Specification Language 73
4.1. ViNNSL Development 73
4.2. ViNNSL Template 74
4.2.1. Metadata template 74
4.2.2. Environment 75
4.2.3. Neural Network Structure Template 76
4.2.4. Parameters Template 78
4.3. ViNNSL Generation 78
4.3.1. Generated Metadata 79
4.3.2. Generated Structure 79
4.4. ViNNSL Model 80

5. Requirements specification for Administration Module 83
5.1. General Definition 83
5.2. Affected Users 83
5.3. Administration Dashboard 83
5.4. Openstack Dashboard 85
5.4.1. OpenStack Nova Service 87
5.4.2. OpenStack Neutron Service 90
5.4.3. OpenStack Images Service 92
5.4.4. OpenStack Vitrage Service 94
5.5. Dashboard Settings 97
5.6. Monitoring System 98
5.6.1. Monitoring Charts Creation 98
5.6.2. Monitoring Charts Representation 99
5.7. Alert System 100
5.7.1. Alerting rule Creation 100
5.7.2. Alerts Representation 101

6. Requirements specification for Main Application Module 103
6.1. General Definition 103
6.2. Affected Users 103
6.3. N2Sky Dashboard 104
6.3.1. Available Tools 105
6.3.2. User Projects 106
6.3.3. The User Neural Networks and Models Overview 106
6.4. The User Project Dashboard 108
6.4.1. Available Tools 108
6.4.2. Neural Networks Grid 110
6.4.3. Saved Trained Models 110
6.5. Three Steps View 111
6.5.1. The Neural Network Description 111
6.5.2. The Neural Network Structure 112
6.5.3. The Neural Network Training 114
6.5.4. Training Results and the Model Evaluation 117
6.6. Neural Networks Repository 119
6.7. Models Repository 121

7. Tutorial 125
7.1. Using Network and Model Repositories 125
7.2. Creation of the Neural Network from the Existing Paradigm 128
7.3. Upload Own Neural Network Paradigm 131

8. Use Cases 133
8.1. N2Sky as a Learning Platform 133
8.2. Developing the Neural Network with N2Sky 134

9. Developer Guide 137
9.1. System Configuration 137
9.1.1. Prerequirements 137
9.1.2. Setting Up the Database 137
9.1.3. Setting Up the Cloud Web Service 138
9.1.4. Setting Up the Model Repository Service 139
9.1.5. Setting Up the User Management Service 139
9.1.6. Setting Up the N2Sky Frontend 140
9.1.7. Setting Up the Backpropogation Neural Network 140
9.1.8. Setting Up the Monitoring System 141
9.1.9. Setting Up Alert Management System 143
9.2. API Documentation 145
9.2.1. N2Sky Monitoring System API Documentation 145

9.2.2. Alerting Management System API Documentation 148
9.2.3. OpenStack API Documentation 149
9.2.4. User Management API Documentation 154
9.2.5. Model Management API Documentation 156

10. Conclusion and Future Work 159

A. ViNNSL Backpropagation Template 161

B. Generated ViNNSL 163

C. ViNNSL Model 167

D. Custom Image with the Monitoring System 169

1. Introduction

The master thesis has a tight connection with the infrastructure design for the N2Sky System proposed by Aliaksandr Adamenko [1]. The basis of the infrastructure is a microservices architecture. It is implemented and designed using concepts of decomposition, single responsibility and continuous integration, which allows to scale the system across the share-nothing infrastructure and gives more control over the computational resources. The architecture is designed to provide easier natural support for Cloud deployment with distributed computational resources.

Objectives: Develop the ready-to-use N2Sky System, which can be easily constructed from scratch and be maintainable on purpose. Implement scalable application, which can be used on any device. Introduce ViNNSL template in order to use this specification language across the whole application. Apply microservices architecture on the backend as well as on the frontend side. Organize monitoring and alerting system on N2Sky cloud and its instances. Make the live demo-prototype for testing purposes.

Non-objectives: The thesis contains information about the future production system. The current version is not fully ready and not tested in order to be released.

1.1. Motivation

Artificial neural networks have a long history. It is referred to cybernetics and connectionism. After the introduction of "deep learning", scientists started to use the full potential of neural networks [9].

The director of AI (Artificial Intelligence) at Tesla, Andrej Karpathy, described the main reasons why the AI was held back:

- *Compute* (the obvious one: Moore's Law, GPUs, ASICs)
- *Algorithms* (in a nice form, not just out there somewhere on the internet)
- *Infrastructure* (software under you - Linux, TCP/IP, Git, ROS, PR2, AWS, AMT, TensorFlow, etc.) [13]

These blockers have disappeared today, which gives engineers and scientists the possibility to investigate and create projects based on AI.

Nowadays, there is a dozen of different types of neural networks and existing models. The need to have a platform to consolidate them all is crucial. The N2Sky platform gives the opportunity to simulate owned or already existing neural networks without the need of buying an expensive environment. The platform where neural network engineers can publish their own neural network and observe its behavior or where other users try to train it and evaluate the trained model. N2Sky is this kind of platform, which gives the possibility not only to use the N2Sky cloud for their needs but also to try out and study neural network field without deep knowledge in this field.

1.2. Terms and Definitions

N2Sky is an artificial neural network simulation platform, which implements the Neural Network as a Service approach. It is a cloud-based system, which implements microservices architecture on infrastructure as well as on application level. The system was motived by delivering an intuitive tool for different stakeholders. The arbitrary user looking for existing neural network solutions, the neural network engineer creating a neural network based on existing neural network paradigms, the contributor implementing and publishing it on N2Sky repository. Since N2Sky is also a simulation platform, every stakeholder can perform training of the desired neural network as well as evaluate trained models. N2Sky today provides a virtual collaboration platform to the computational intelligence community.

ViNNSL template is an adopted for N2Sky application ViNNSL language [4] , which contains the neural network description, metadata, structure of the neural network and its connection. The ViNNSL template is used across the entire N2Sky application, on the frontend as a representation and on services side as a database model.

Micro application is an implementation of the microservices approach, which is normally used on the infrastructure side, but in this case it was used on the frontend application. The micro application is a small application, which is responsible for limited and consolidated workflows. The micro applications can be bundled with other applications and work as a whole so that the user has the feeling that he is using only one application.

1.3. Related Work

The basis for development and implementation of the N2Sky platform were various theoretical concepts which were reviewed and improved by multiple researchers.

The UK e-Science initiative [8] describes several goals to be reached by fostering new stimulating techniques:

- Enabling more effective and seamless collaboration of dispersed communities, both scientific and commercial.
- enable large-scale applications comprising of thousands of computers, large-scale pipelines etc.

- Transparent access to "high-end" resources from the desktop.
- Provide a uniform "look & feel" to a wide range of resources
- Location independence of computational resources as well as data.

Since large numbers of neural networks were developed, especially in the past few years, authors were motivated to create "everything about sharing" approach, which was implemented in N2Grid [30] and N2Cloud [12].

Unfortunately, both systems required deep knowledge in the neural network field and were almost not usable by the arbitrary users. The limited number of simulation and non-friendly user interface does not attract users, which leads to these common problems:

- *Complexity.* The tool was too complex and did not have an intuitive user interface.
- *Proprietary system* missing interconnection and data exchange to other software systems.
- *Lack of resources* for routine workflows.

The most related tool, which represents an expert system, was the CIML (Computational Intelligence and Machine Learning) [42]. The main idea of the tool was to create a web-based platform for machine learning, neural networks, and artificial intelligence, where any user could share and obtain resources. The project failed because its target was a too big diverse community and offered the all-in-one solution. Due to the complexity and lack of guidance, it failed.

After this research, the original N2Sky [29] was released. It was a cloud-based solution oriented only on neural networks. The idea was that any user could run the simulation on any device. The system supported multiple stakeholders with different roles and it also had a mobile web portal.

The original N2Sky supported ViNNSL language. ViNNSL is XML-based domain specific language, which was developed as a communication framework to support the service-oriented architecture based on neural network environments.

The decision was to take the good parts, refactor them, apply new and trending technologies in order to approach the production-ready system. In this master thesis, the novel N2Sky System was developed just by using the concepts, trial and failures from previous generations.

2. N2Sky Architecture

2.1. Current Architecture Analysis

The current N2Sky architecture is a monolithic standalone application, 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 which functionalities are still reusable. By doing that, new functionalities in a reasonable time will be gained.

2.1.1. Architecture Design

The current N2Sky design is based on RAVO architecture [14], 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 presented in figure 1.

Every service represents a 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 contains the User interfaces (UIs).

Everything as a service is considered to be a good idea but build upon it, the layered architecture could cause various issues. Since every layer is depending on the above one, there exists a possibility that if the services from the bottom layer are unreachable, then the application is not usable at all.

Besides the stand-alone-application design, which was implemented, another 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.

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

2.1.2. Components

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

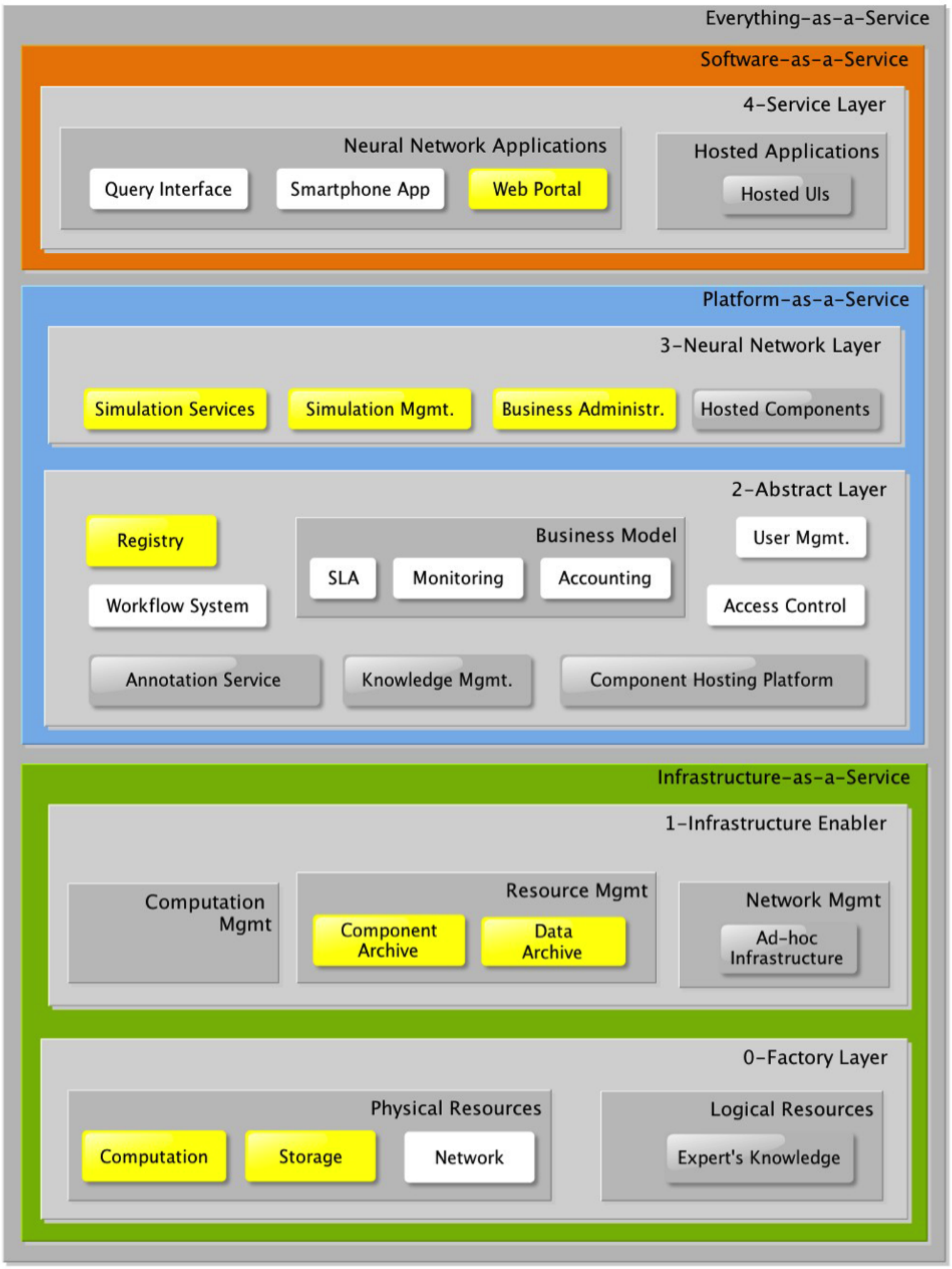


Figure 1: Current N2Sky architecure

- *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, 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, responsible for services management, it can slow down the whole process. All requests go throw this service. If the services are not accessible and high latency or waiting time occur, the UI as well as the API services would be unavailable.

Having the same services distributed on multiple servers would be great. Every service in the previous version of N2Sky is gigantic, that is why it is possible to divide services into microservices and incapsulate sensible components from public access.

2.1.3. Implementation

The previous version of N2Sky is fully written in Java using Spring framework, Java servlets and Tomcat server. The Java Spring framework is best used for enterprise applications, but N2Sky needs lots of computations so the high-end supercomputers are a necessary pre-requirement in this case.

Unfortunately, it is impossible to map the current architecture into the microservices approach. The only solution is making replicas of the application, but then the sessions and data consistency have to be tracked in each one of them.

The mobile application of N2Sky is built in raw HTML5 and JavaScript. Today, looking on this mobile application, is almost impossible to maintain it. Nevertheless, the mobile version is also wrapped into a layered architecture, while the implementation part 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 mobiles with a small screen resolution.

The whole system is deployed on Amazon Web Services (AWS), which is a good thing 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

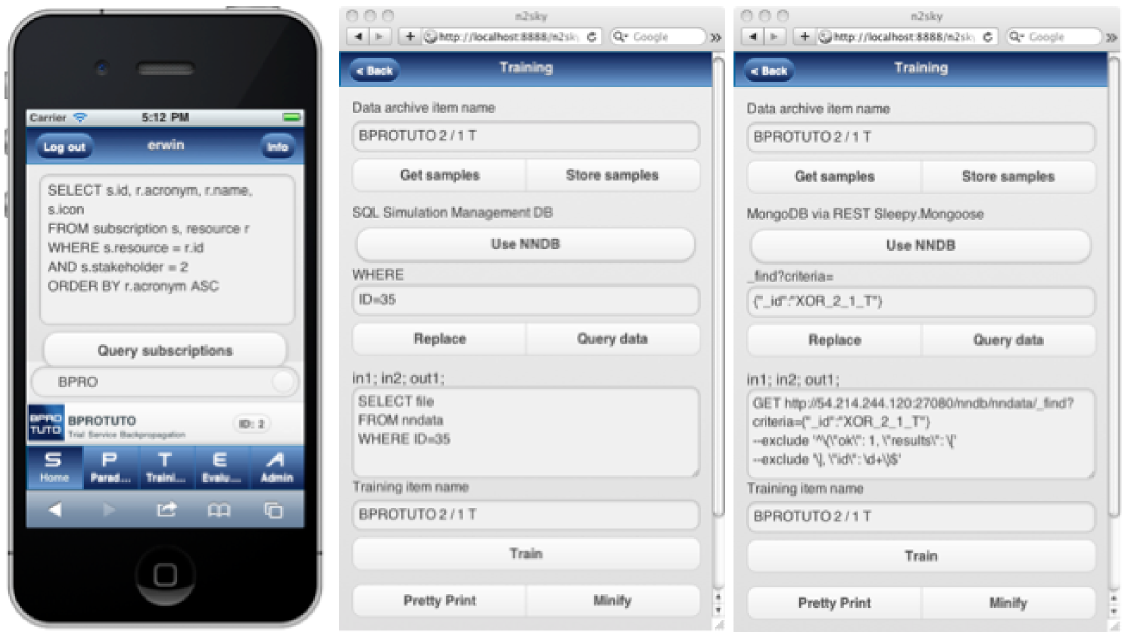
2.2. Redesign Motivation

Application redesign is a project, which takes a lot of work and time. But at some point, every designer has faced a refactoring project. It is strongly connected to the user experience. Bad user experience will make users stop using an application and leave negative feedback for the application.

2.2.1. Redesign Process

There is available data, information and user experience of the previous version of N2Sky to work with. During the redesign, it was already known who the users are and what they are trying to achieve. Using such information, the possibility to create future goals for user interface and user experience is high.

Finding problems. There are multiple problems with the application. One of the most crucial is that the user interface is not intuitively understandable as it shown in figure 2.



```
SELECT s.id, r.acronym, r.name,
s.icon
FROM subscription s, resource r
WHERE s.resource = r.id
AND s.stakeholder = 2
ORDER BY r.acronym ASC
```

BPRO

BPROTUTO ID: 2
Trial Service Backpropagation

Data archive item name: BPROTUTO 2 / 1 T

SQL Simulation Management DB

WHERE: ID=35

in1; in2; out1;

```
SELECT file
FROM nndata
WHERE ID=35
```

Training item name: BPROTUTO 2 / 1 T

Data archive item name: BPROTUTO 2 / 1 T

MongoDB via REST Sleepy.Mongoose

_find?criteria=: {"_id":"XOR_2_1_T"}

in1; in2; out1;

```
GET http://54.214.244.120:27080/nndb/nndata/_find?
criteria={"_id":"XOR_2_1_T"}
--exclude '^\{\"ok\": 1, \"results\": \['
--exclude '\], \"id\": \d+\}$'
```

Training item name: BPROTUTO 2 / 1 T

Figure 2: Current N2Sky User Interface

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

User interviews and Questionnaire. User insights are very important. They help with the understanding of problems nature. In the case that a user finds something confusing, the interviewer needs to dig deeper to stress the importance of particular insights. Unfortunately, there were no analytic data or any reviews regarding UI and UX. With a small group of colleges, the current UI of N2Sky was reviewed. The following questions were derived (Q1-Q5):

- Q1: "How can N2Sky help you with the developing of your neural network?"
- Q2: "What was the most difficult part of creating a new model?"
- Q3: "Did you face any problems during spawning of your neural network? If yes, 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 the using of N2Sky?"

There were five students interviewed which were grouped together. The summarised answers (A1-A5) according to questions (Q1-Q5) are shown below:

- A1: N2Sky gives the 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: A user faces difficulties during the creation of a new neural network model. The user interfaces contains technical jargon and it is not intuitively understandable.
- A3: Spawning a neural network was a pretty clear process, but it was unclear if the neural network was ready to be used or not.
- A4: Logging of the training data is very useful. The neural network owner can see how his network behaves with different training and testing data.
- A5: A user would want to perform testing and training on already existing neural networks.

Current application design mapping. After studying the answers, highlighting weak parts of the application was easier. This approach will show a big picture of the current application design:

- An arbitrary user needs to know multiple technologies and programming languages just to simply reuse existing neural networks.
- Too much information on every view. The purpose of the view is overloaded. Each view has too much functionality, which makes the user loose focus.
- The application works relatively slow. Even if processes behind might be happening, the user does not know it.

Important is to face the problems, but that 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" [31]. That is why it was decided to consider the problems of current N2Sky design and reuse working ideas in a refactored system

Application Maintenance. N2Sky was the monolithic standalone application, which included all services in one and was deployed as a whole. The application was not distributed. In case one of the services did not work correctly, the whole application was unusable. Originally the previous version of N2Sky was fully written in Java. There were hundreds of classes, providers, and services in one project. The developer would spend hours to maintain this kind of project. Finding an issue in a big application is always a challenge. Small changes are causes for subsequence changes. If the software breaks after such a change, then it will add high effort for fixing 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 re-engineer the code and is more likely to result in defects due to not understanding the side effects" [19]. He categorizes this kind of code as "Smell" code. Unfortunately, seen from the maintaining perspective, N2Sky had all the above mentioned problems.

That is why N2Sky has shifted from a monolithic system to a container based system with an independent micro service which located in the cloud environment. The frontend and its services are lightweight and easy maintainable. If something happens to go wrong, the developer will know exactly where the problem is. The independence of services makes it almost impossible to break something during fixing it. Additionally, there are monitoring and alerting systems which support developers during maintenance. Before, a user could not tell it the application was working correctly or was it even running. Users could get a bad experience while using an application in case of an improperly working environment. Now, it is possible not only to notify an application administrator about problems but also to predict potential threads.

2.2.2. Refactoring the User Interface

User Interface (UI), an abbreviation of the 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 were invented by hlJakob Nielsen. It was decided to apply the following ten general principals of interaction design to N2Sky:

- *Simple and natural dialogue.* N2Sky will have a simple UI which is understandable for any user, even if the user is not an expert. Every icon, navigation or action button will be self-describing. The application will follow the slogan "less is more". No more overloaded views. The idea of the 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 user's language.* Developing N2Sky was concentrated on user perspective. There is no technical jargon for the arbitrary user.
- *Minimize user memory load.* There is no multiple options, functions or menus in one view. There is also no multiple ways to do the same thing. N2Sky teaches the user how to make things done with one existing and convenient workflow.
- *Consistency.* N2Sky has similar layouts, fonts, colors, icons types, structures and organization through the entire application. The user should get the same visual experience in every view.
- *Feedback.* Every action, process or even error will be notified. The user will know exactly what is happening with the system with clear and understandable messages.
- *Clearly marked exits.* Every push-to-action button has a short and clear caption.
- *Shortcuts.* N2Sky has multiple user types. One of the types is the expert user, which is the advanced user in neural network and artificial intelligence topics. For this kind of user is provided more technical jargon, but this UI is separated from arbitrary users UI.
- *Good error messages.* Every notification is understandable and includes an error message if it happens to occur. Every message has a simple description.
- *Prevent errors.* In N2Sky is implemented the logical structure of UI components. There are constraints which help the user in the workflow. For example, the 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 an 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 relationship with the UX follows since this is also one of the key concepts 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 the product. It deepens into such aspects 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 [11]. Putting both of UI and UX in comparison, to all appearances the user interface is the target of 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. UI will concentrate on the appearance and design of the product, rather than the functionality. The intent of it relies on 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 the visually comprehensibility meaning which colors or font type shall be used for a better perceptibility of the product [34]. 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. 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.

2.2.3. Advanced Project Management

In order to make N2Sky competitive on the market, a modern approach in the 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 future planning.

FRS is a document that defines functionality, which the application or some parts of it must perform [37]. N2Sky is a sociotechnical system, meaning that it strongly interacts with humans.

FRS for N2Sky is described in the natural language with formal methods in order to establish specifications between development process and end-user consuming. Every N2Sky module has FRS, which is explicit and points on the systems functionality. A good FRS must be unambiguous, consistent and correct [10].

The formal or semi-formal methods serve for analyzing and validating FRS. The main purpose of that is to limit interpretation errors. The problem occurs when FRS is fully written in natural language when designers do not have required technical knowledge in order to use other languages. One of the typical solutions is that it defects detection techniques, which require effort from the designer [39].

In N2Sky, some methods and techniques were applied in order to make specification as clean and clear as possible. Despite that, in FRS will always be parts which are leading to misunderstanding [10].

FRS is part of the engineering phase. Following qualities for N2Sky were defined and applied during this phase [16] as shown in figure 3:

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

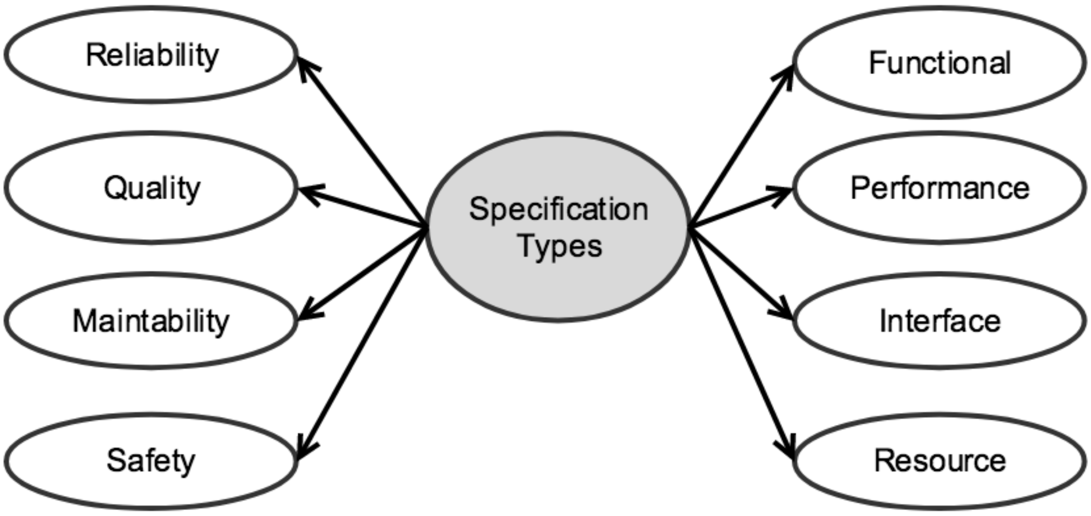


Figure 3: Requirement Specification Qualities

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 his own way of interaction with the application.

Every user has some specific area within the works. For example, authorised users do not need to know the current environmental monitoring information. These restrictions were the motivation to create some user roles in order to restrict or grand some functionality of N2Sky as shown in figure 4.

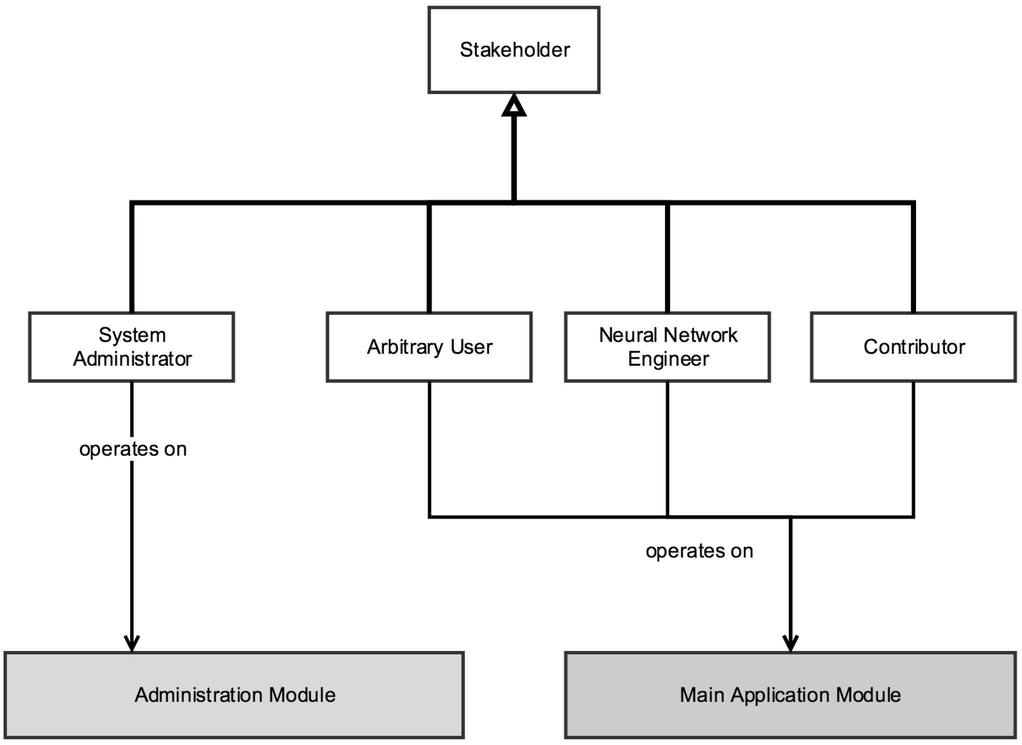


Figure 4: User roles hierarchy and modules where they are operating on (marked grey)

- *Arbitrary User.* The Arbitrary User is 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 of 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 networks. The arbitrary user can be converted to a Neural Network Engineer user if he has enough knowledge for it.

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

- *Neural Network Engineer.* The Neural Network Engineer is an arbitrary user. The Neural Network Engineer has 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 his own neural network instances from existing neural network paradigms. 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 full access to the application including environmental 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 the more detailed overview of user roles it is possible to define permission and main functions. Permissions describe a users allowed page view. If the user has access to a particular page view, the main user function can be defined. The main user function characterizes allowed behavior on a particular page view.

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

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

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

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

As it was mentioned before, the System Administrator as well as Moderator has access to the N2Sky Main Application Module, but they do not have the main function over that. On the other hand, all user roles, which are extending consumers can contribute to the N2Sky Main Application Module, except Consumer itself. The 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 stakeholders 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 backend 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 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 is located the load balancer, which includes shared cloud resources. Since the N2Sky environment needs to support thousands of Docker containers [23] simultaneously, only one container orchestration like Cloudify [33] with a microservices architecture could perform this. Cloudify is a container orchestration tool, which provides communication between cloud platforms and manages container deployment and execution. Docker container is an operating-system-level virtualization. Containers technology is used across entire system from frontend and its services to monitoring and alert management system.

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

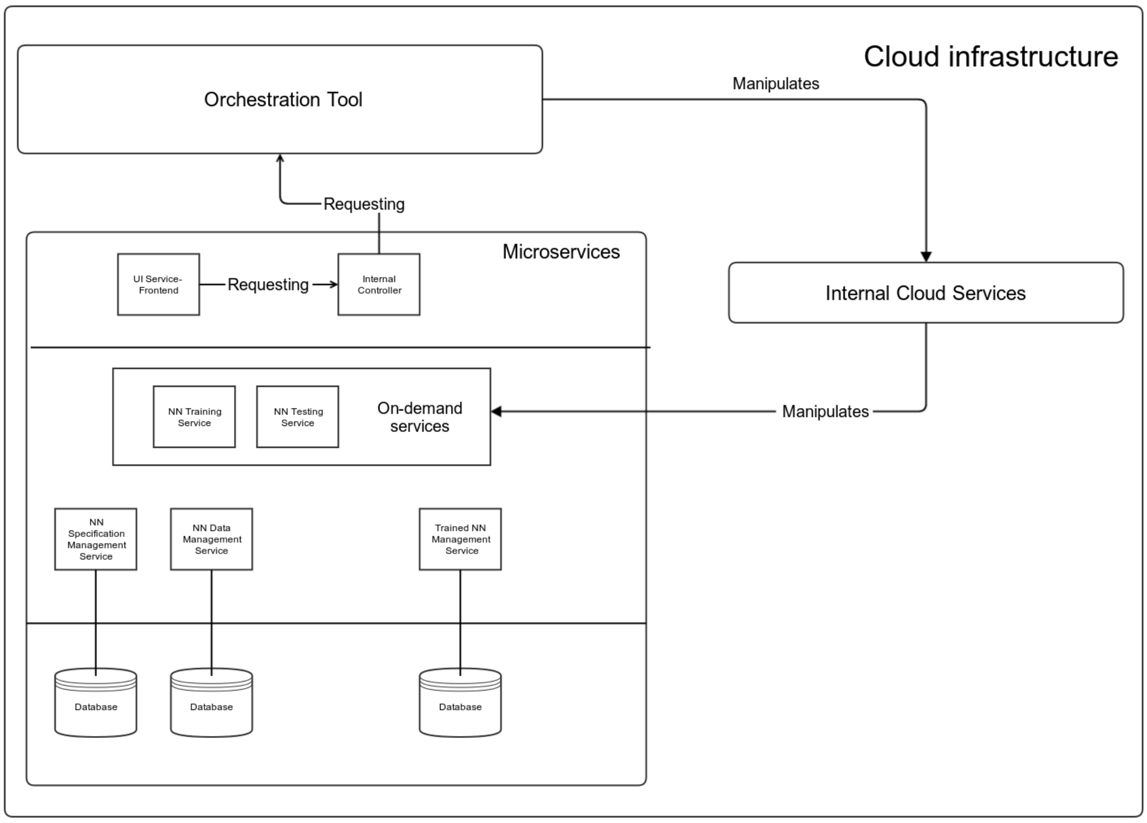


Figure 5: Novel N2Sky Architecture

- *Orchestration Tool* is the Cloudify framework, which communicates with a cloud infrastructure, namely creates and manages 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 containers.
- *UI Service-Frontend* is a modular application. The user can request container management over the UI, which will be redirected to Internal Controller.
- *On-demand services* are not exposed publicly, 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*, is located right on top of the database and provides the API for adding external data sources.

- *Trained Neural Network Management Service* is a service, which is responsible for evaluating trained models.

2.3.2. Modular Frontend Application Design

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

Microservices in the 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 knowledge about other parts of the application.
- *Diversity of technologies.* The monolithic approach makes the whole application stick to one framework. Microservices allow to use any technology without the need to rewrite the application.
- *Independent deployment.* Every application has release periods. Every release is accompanied with a redeployment procedure. There is no need to redeploy the whole application, but just only the required components.

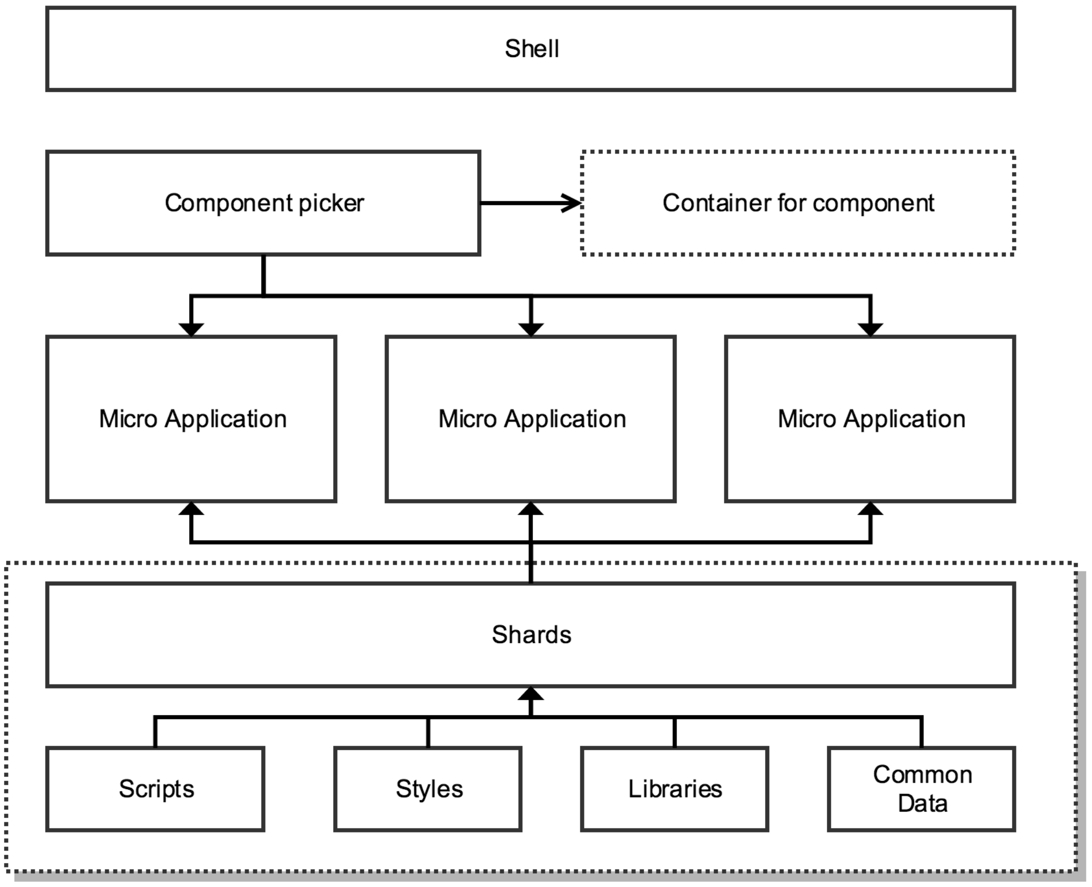


Figure 6: Microservices approach in frontend application

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

- *Shell.* is a top-level component, which wraps the Components picker and Container for the component. It contains the 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.

The central concept of the application is to support the Software as a Service (SaaS) and Platform as a Service (PaaS) distributions [36]. N2Sky consists of two modules: administration module and main application module as shown in figure 7.

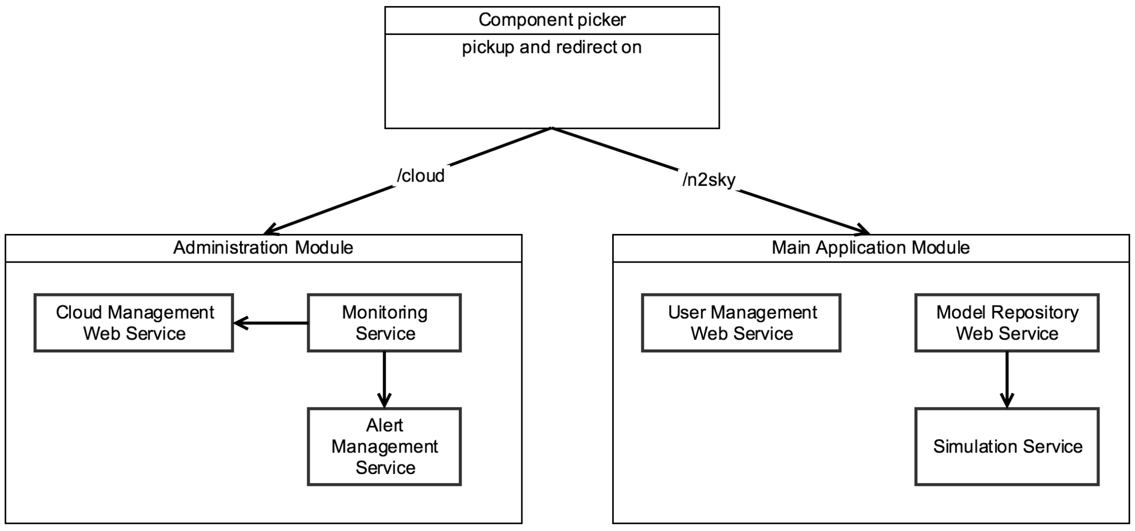


Figure 7: N2Sky frontend application and its services

- *N2Sky component picker.* When the user is located in to the N2Sky web portal, first he will be dealing with the component picker. The component picker is a small service, which redirects the user depending on the 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 networks. N2Sky allows deploying own networks 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. It is the 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 the Software as a Service distribution, every web service can work independently. This means that the stakeholders can use N2Sky via web portal as well as use N2Sky API. N2Sky API allows stakeholders to:

- Authorise in the System
- Create new neural from existing paradigm
- Deploy own neural networks 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 use cloud service as a service, the user has to install it on his own cloud environment. This approach supports the Platform as a Service distribution. Cloud services are only available for system administrator and granted users.

The sample workflow overview, which shown in figure 8 represents microservices architecture in action:

1. *Contributor User*
   a) The Contributor user authorizes 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 describes his own neural network paradigm using the ViNNSL template and deploying it in the N2Sky cloud.
   c) The Contributor user performs 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 the available API.
   e) The Contributor user will be awaiting until other N2Sky users use his neural network paradigm in order to monitor the behavior of the neural network.
   f) The user modifies, redeploys and retrains his neural network after the first results.

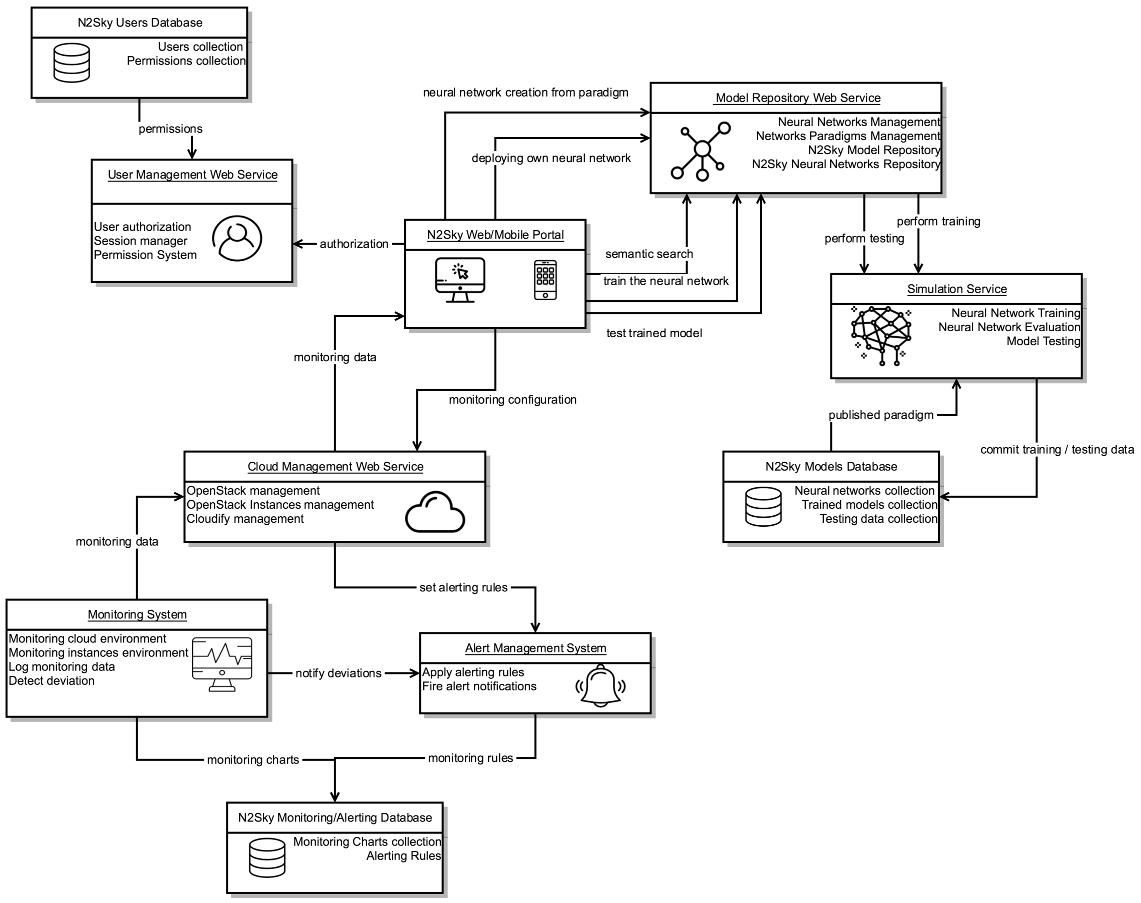


Figure 8: The sample workflow

2. *Neural Network Engineer User*
   a) The neural network engineer user authorizes 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 paradigms using Model Repository Web Service.
   c) The user performs training against his newly created neural network using the N2Sky platform.
   d) If the user is satisfied with a trained model, he can perform testing using the N2Sky platform.
   e) The neural network engineer user can publish his neural networks and trained models in order to make it available for other N2Sky users.
3. *Arbitrary User*
   a) The arbitrary user authorizes 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 the 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 copies existing neural network and trained models into his project.
   d) The user performs training from the N2Sky platform against copied neural network with the default input parameters data.
   e) The user evaluates trained neural network models with the default parameters.
4. *System Administrator*
   a) The system administrator authorizes the N2Sky portal using his browser on desktop PC or mobile device. He will be redirected to the administration dashboard.
   b) The user observes the cloud environment.
   c) The system administrator creates the new monitoring chart with specific metrics and adds it to the administration dashboard.
   d) The user creates alert against newly created monitoring.
   e) The user is notified by the Alert Management System, that a specific event occurs.

2.3.4. Technology Stack

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

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

- *Benefits.* Easy to develop in Java. The developer does not have to think about JavaScript functionality. There a dozen of predefined components like buttons, input fields, frames etc. Customisation is also possible.
- *Obstructions.* The deployment process is a blockage process. There is no "hot redeployment" available. Even if the developer might save time from the build components in Java, he will lose more time by continuous redeployment.

  Java applications need servers which supports JVM, meaning that the server should have much more memory than some other frameworks, which is written in 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. The React-Native extension for this framework allows wrapping the whole application in a mobile as well as in a desktop application.
- *Obstructions.* Difficulties supporting big projects. JSX is not a type-safe programming language. Exception handling also needs to be done by the 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 the IONIC framework.
- *Obstructions.* AngularJS compiles TypeScript code into the JavaScript core and sometimes the compilation fails because TypeScript is a new language and it is not fully adapted for browsers.

Vaadin does not fit the N2Sky needs, but AngularJS and ReactJS could both perfectly fit. These frameworks are written in JavaScript and have big corporations behind:

AngularJS was developed by Google and ReactJS was developed by Facebook. Since N2Sky has to support a fully cross-platform architecture, it was decided to choose ReactJS. With this framework, N2Sky has great potential to be a multi-platform application in the future.

Furthermore, the backend has the microservices architecture to support scalability. By choosing the JavaScript framework for frontend, it makes sense to use the same in backend as well, 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 the administration dashboard, the original dashboard was no more needed. Every backend and frontend service is deployed on OpenStack instances. Every instance is absolutely scalable, which allows finding the best environment for every service.

Each 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 exists a Monitoring Management System. The basis of this system is Prometheus monitoring system, which gives full access to all information on any installed server. Prometheus has an open API, which is used by N2Sky. Using Prometheus, there were charts and graphs developed and integrated into N2Sky. The Alert Management System, which is a part of N2Sky, is also using the Prometheus API in order to detect the deviation and notify if an event occurred.

As a database, it was decided to choose the NoSQL one. There were two NoSQL databases under consideration ElasticSearch and MongoDB. ElasticSearch supports indexes. It is possible to configure this index, so that is impossible to insert something which is not mapped by the index. MongoDB does not use the index approach, but a MongoDB client supports the schema. It was decided to use this MongoDB schema and map the ViNNSL 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 of a whole unreachable system, every service can be restored with a Jenkins Profiles.

3. N2Sky Components

In the concept of N2Sky, lies a minimalistic and modern design. Dimmed tones of the UI components are used to make the end-user to feel like he is using a 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 to navigate easier through the application. It is important to have common components and elements and do not mix up together. Every component and every GUI element should have the self-describing purpose. Building user interface components and elements is almost the same as developing an UML diagram. It is possible to group common parts and maximize reusability [26].

It is important to consider the multiple end-users, devices, platforms as well as environments that will be used. Through a heterogeneous context, only needed elements have to be displayed. It is necessary to make view prototypes to determine if the particular components are comfortable and easy to use [20].

3.1.1. N2Sky Layouts Design

Design of N2Sky layouts was concentrated on principles of User interface design which were described in [27]:

Organise. All UI elements and components have to be ordered and not be chaotic. Randomise position or not understandable logic can confuse the user.

Consistency. There are few types of consistency:

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

Screen Layout. There are few screen layouts:

- Grid layout, which organizes all components into blocks, like menus, navigation bars etc.
- Standardise the screen layout, which is mostly used on screens with restrictions.

- Group related elements, which is usable for smaller screens.

In N2Sky Grid, the layout is used since it is easy to apply responsive design and reorganize grid items.

Navigability. Navigation always needs to be on user focus or at least effortless to find. How it is supposed to work and where or how to use its elements have also an essential importance.

Economize. Following rules have to be applied:

- Simplicity
- Clarity
- Distinctiveness
- Emphasis

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

3.1.2. N2Sky Fundamental Layout

The N2Sky Fundamental Layout is represented by basic styling of the N2Sky application as is shown in figure 9.

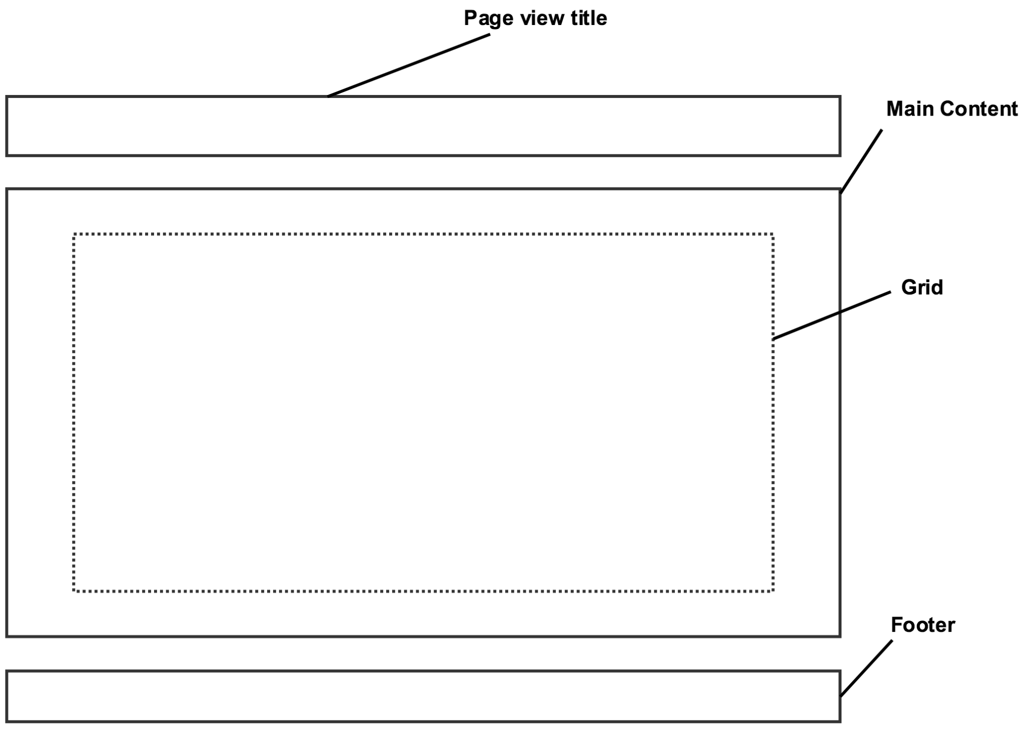


Figure 9: N2Sky Fundamental Layout

This layout applies styling like:

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

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 a centered component, which represent the primary context. This component is always in focus by the user and can contain one or more grid components.
- Grid in its basic layout represents the positioning of elements, which can be displayed inside it. In the basic layout, in grid components it is possible to add any elements.
- Footer is a component which goes across the whole application with a static data. Normally, it contains terms and conditions.

3.1.3. N2Sky Main Layout

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

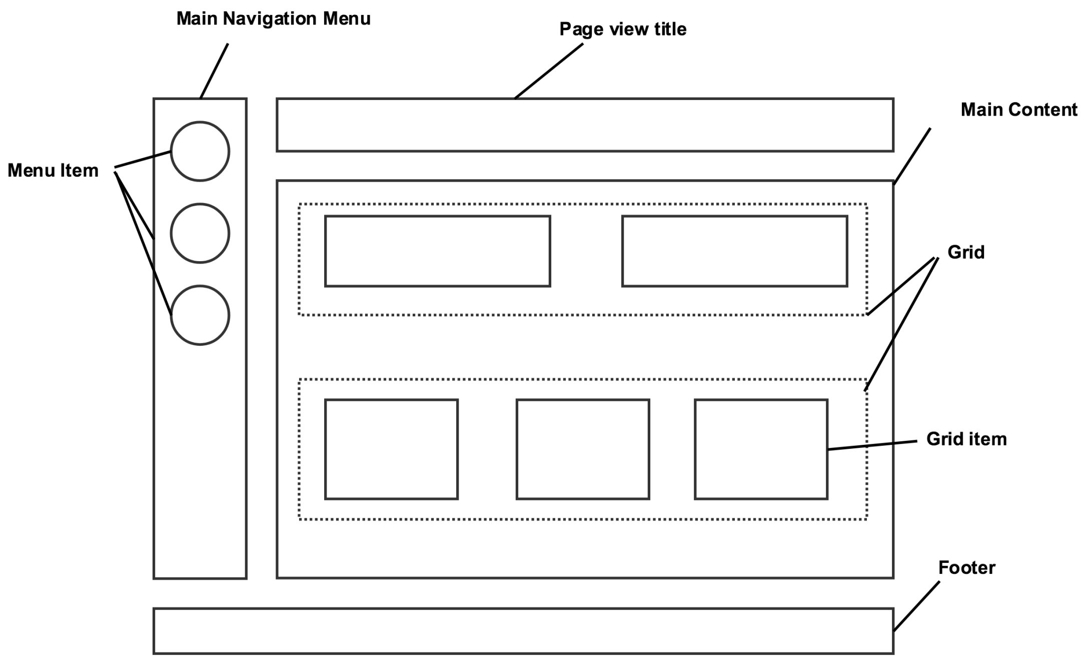


Figure 10: N2Sky Main Layout

- Main Navigation Menu, which is displayed as a vertical bar. This component has two states: on mouse over which it will be extended and menu items with caption text and icons will appear; on mouse away from the component only menu items icon will be displayed.
- Menu items are icons with caption components. Which items will be shown or hidden depends on the users permissions.
- Grid items are blocking components, which can have multiple sizes but fixed within one grid. The context of grid items can be customized.

3.1.4. N2Sky Mobile Layout

N2Sky supports mobile devices. For mobile devices, the mobile layout was developed as is shown in figure 11.

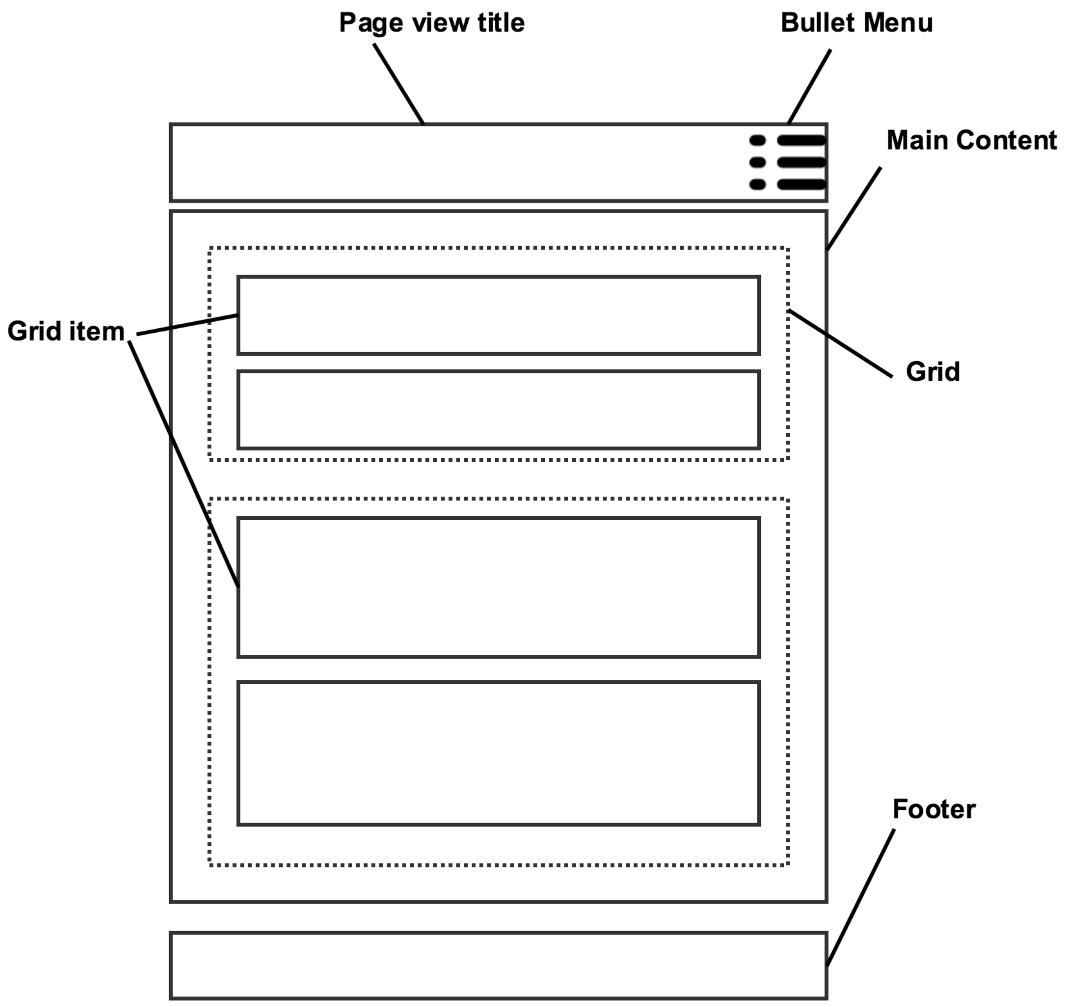


Figure 11: N2Sky Mobile Layout

The context of all components remains 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 the page view title component is the Bullet Menu located. The normal desktop view is hidden and instead the bullet menu will perform the same functionality. On click, the menu will appear as an overlay of the current view.

3.1.5. N2Sky Page Views

As it was mentioned in subsubsection 2.3.4 "Technology Stack", the N2Sky frontend application is developed on the ReactJS framework. An important part of the ReactJS framework is the React-Router, which contains all page views of the application and redirects it according to the URL path:

```
<Provider store={store}>
```

```
        <Router history={browserHistory}>
                <Route path="/" component={Auth}/>
                <Route path="/signup" component={Reg}/>
                <Route component={AbstractDashboardLayout}>
                        <Route path="/cloud" component={DashboardsOverview}/>
                        <Route path="/user/profile" component={UserProfile}/>
                        <Route path="/openstack"
                                component={OpenStackMainDashboard}/>
                        <Route path="/openstack/project/:id"
                                 component={OpenStackProjectDashboard}/>
                        <Route path="/openstack/server/:projectid/:serverid"
                                component={ServerDetailsDashboard}/>
                        <Route path="/openstack/vitrage/:templateId"
                                component={VitrageDetailsView}/>
                        <Route path="/alert" component={AlertDashboard}/>
                </Route>
                <Route component={AbstractDashboardLayout}>
                        <Route path="/n2sky" component={N2SkyDashboard}/>
                        <Route path="/n2sky/available"
                                 components={AvailableNetworksOverview}/>
                        <Route path="/n2sky/models"
                                components={ModelsRepository}/>
                        <Route path="/n2sky/paradigm/create/:projectid"
                                components={AddNNFromParadigm} readOnly={false}/>
                        <Route path="/n2sky/paradigm/nn/:id"
                                components={AddNNFromParadigm} readOnly={true}/>
                        <Route path="/n2sky/network/:id"
                                component={NetworkDetails}/>
                        <Route path="/n2sky/network/:id/test/:model_id"
                                 component={NetworkTestDetails}/>
                        <Route path="/n2sky/project/:id"
                                component={ProjectDashboard}/>
                </Route>
        </Router>
</Provider>
```

Listing 1: React Router

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

- Path, which is 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 two modules: Administration module and main application module. Both of these modules are 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 is the main control center of the OpenStack services.

  OpenStack Project, path: "/openstack/project/:id". Route to the OpenStack project, which contains information about servers, flavours, images etc. of the 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 is 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 the Alert Management System. This view contains information about alerting rules and alerting events.

- **Main Application Module:**

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

  Registration View, path: "/signup". Registration View contains the 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 projects.

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

  Neural Network Create View, path: "/n2sky/paradigm/create/:projectid". This view represents the workflow of the creation of neural networks from existing paradigms. The newly created neural network will be saved in users project, hence this project id is required.

Neural Network Details View, path: "/n2sky/network/:id". The details of the created neural network. It can be a logged in users neural network as well as the neural network of another user. User permissions show visibility levels. The networks id is required.

Neural Network Test View, path: "/n2sky/network/:id/test/:model_id". From this view, the user can evaluate neural networks with 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 a particular project. The project id is required.

3.1.6. N2Sky Dashboards

The purpose of the dashboard is to embed business intelligent (BI) objects into a single page in order to make an overview of highlighted titles of BI objects [25].

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

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

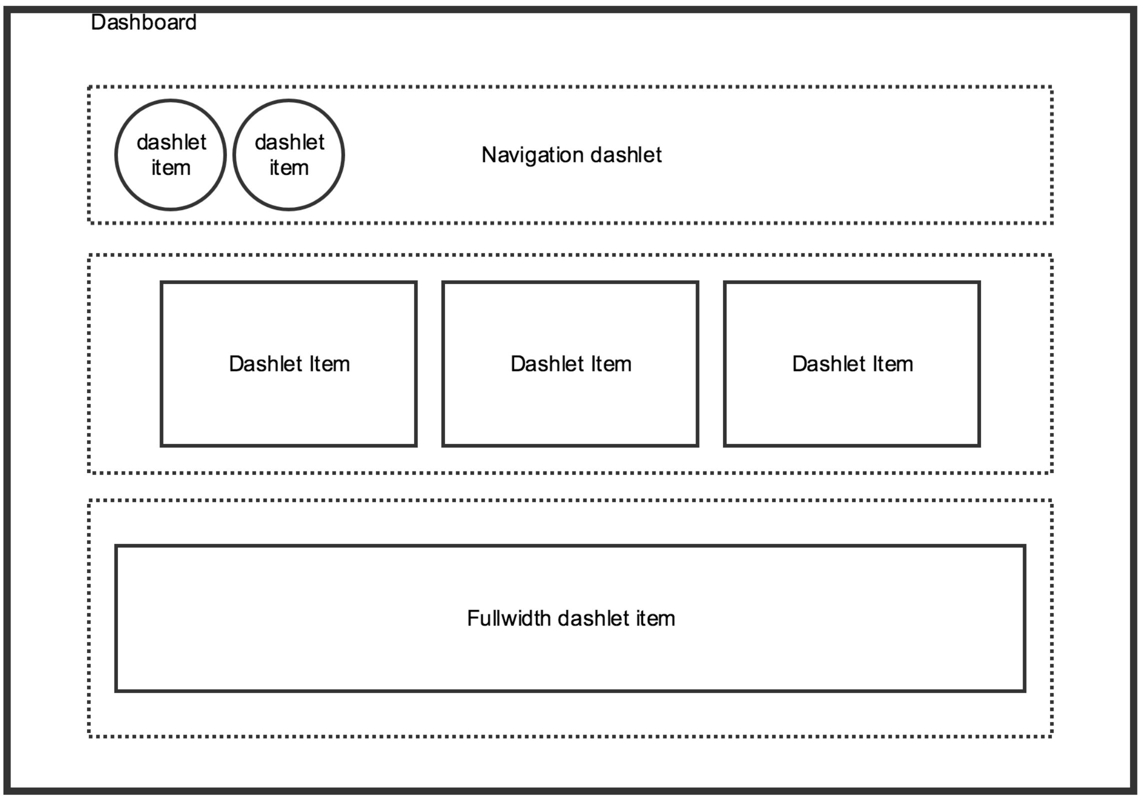


Figure 12: N2Sky Dashboard template

In the figure 12 is shown the typical N2Sky dashboard template. A dashboard in N2Sky has a grid structure. Every dashlet has one or more items. Each item can contain any UI component, but it has to be fit 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 the development of N2Sky, a confronted challenge was when choosing from different usable fonts, which would be displayed all the same and be readable across multiple devices. The typeface [38] 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 [41]. Even such standard fonts like "Arial" and "Times New Roman" appeared unreadable on mobile devices as shown in figure 13.

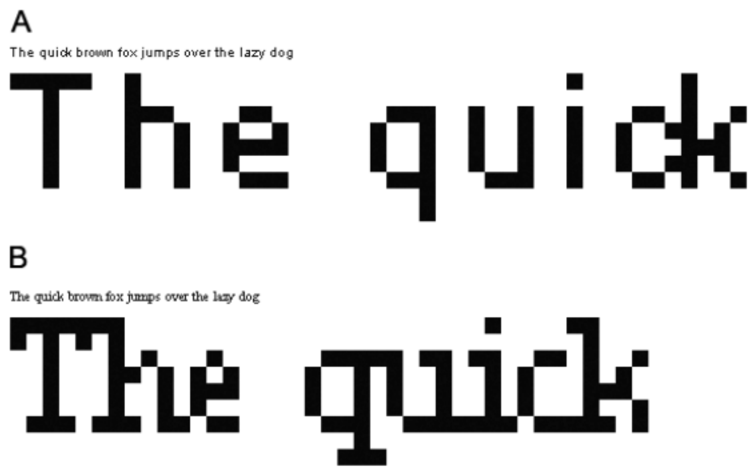

A
The quick brown fox jumps over the lazy dog
The quick
B
The quick brown fox jumps over the lazy dog
The quick

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 freedom to developers and designers. Today is possible to develop an application simultaneously for mobile devices, desktop computers, and web browsers. Problems were though faced when the mobile and tablet devices manufacturers started to produce devices with different resolutions. Today, the versioning of mobile screen resolution came to over 1000 devices, but N2Sky was concentrating only on major market players as shown in figure 14 [24].

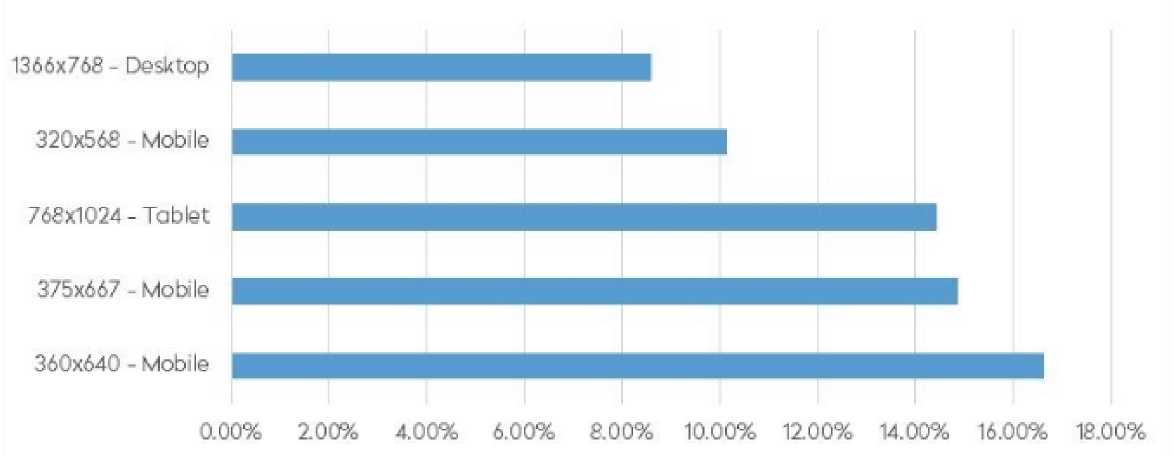


Figure 14: Average screen resolution of mobile, tablet and desktop devices on 2017

N2Sky is focused on mobile and tablet devices because of the extensive trend nowadays.

3.1.8. User Interface Elements

It is possible to divide all UI elements into groups [17]:

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 include also some particular request from and to users. Following UI elements are the parts of navigational components:

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

Informational Components. These components are the addition to workflows. It can help the user to perform actions or inform him that the action will occur or has already occurred. Informational Components contain following UI elements:

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

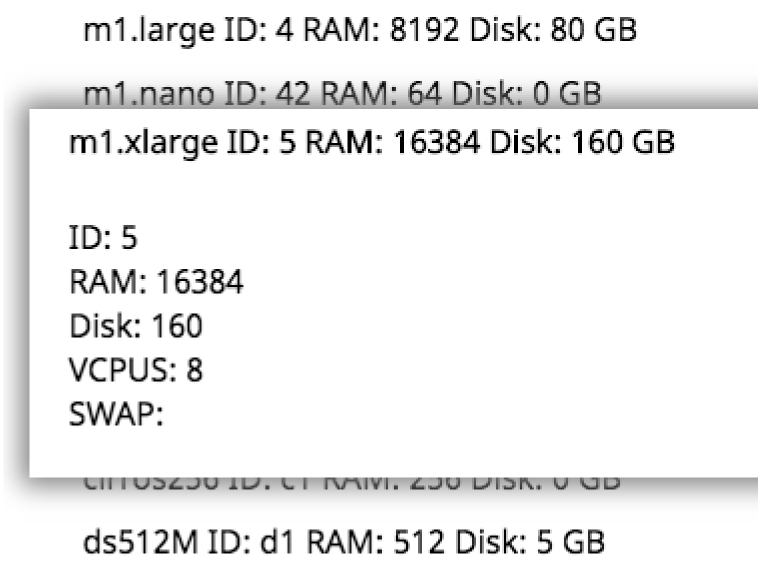
Containers. Containers are components, which are hiding additional information or not focused information, where the user needs to perform the action in order to see it.

- Accordion
- Semi-hidden 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. Each of the UI elements is absolutely customised. Following UI elements were created:

Accordion Accordion is a list of items, which is accessible on mouse click. The N2Sky accordion works more or less like a modal window. With this kind of functionality, other data, which surround the accordion will not be stretched or squeezed but appear on top of elements as shown in figure 15:


m1.large ID: 4 RAM: 8192 Disk: 80 GB
m1.nano ID: 42 RAM: 64 Disk: 0 GB
m1.xlarge ID: 5 RAM: 16384 Disk: 160 GB

ID: 5
RAM: 16384
Disk: 160
VCPUS: 8
SWAP:

ds512M ID: d1 RAM: 512 Disk: 5 GB

Figure 15: Customised N2Sky accordion UI element

Buttons. The idea behind it was to make buttons more interactive and understandable to use as displayed in 16. Buttons contain a caption and icon in SVG format in order to support high quality image in all devices.

Buttons are using on hover animation. When the mouse moves over a button, then the icon slides in the middle to show that the action can be performed as shown in figure 17.

Icons. In N2Sky, all icons are in Scalable Vector Graphics (SVG) format. With SVG the icons do not loose their quality in any device [6]. Since it is a vector graphic, it is easy to edit an icon with programming language code. The N2Sky Logo, which is represented in figure 18, also uses the SVG format.

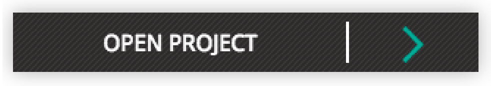


Figure 16: Customised N2Sky button UI element

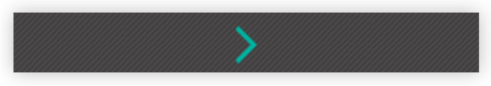


Figure 17: Customised N2Sky button UI element animation

Importing code in some graphical vector editor in order change color, path (vector graphic itself), metadata etc is also possible. The following code demonstrates the N2Sky Logo in SVG format. The whole vector path was shortened.

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

...

-18 74 -40 36 -22 67 -40 70 -40 11 0 253 -152 269 -169 11 -12 15 -27 12
-36 26 -103 -14 -32 -19 -60 -35 -62 -35 -3 0 -34 -18 -70 -40 -36 -22 -68
-40 -70 -40 -3 0 -22 -11 -43 -23 -142 -90 -172 -96 -201 -39 -20 40 -47 168
-55 268 l-5 61 122 22 c67 13 156 30 197 38 85 16 106 30 95 63 -7 21 -11 22
```

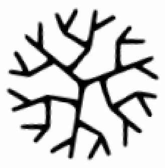


Figure 18: Customised N2Sky logo in SVG format

```
-69 16 -33 -4 -88 -10 -121 -15 -178 -26 -178 -26 -170 -1 4 13 21 46 37 74
57 100 63 116 52 134 -6 9 -22 17 -34 17 -24 0 -47 -33 -130 -180 -75 -132
-74 -130 -61 -206 6 -38 16 -102 21 -141 6 -40 15 -98 20 -129 6 -30 10 -72
10 -93 10 -38 -162 4 c-151 3 -167 5 -212 28 -82 43 -86 57 -86 320 10 226 38
34 c21 19 53 47 72 61 19 14 49 39 67 55 17 16 54 47 82 69 60 49 79 79 65
103 -18 28 -47 25 -84 -10 -20 -18 -52 -45 -72 -60 -20 -15 -42 -34 -50 -41
-23 -23 -92 -67 -105 -67 -10 0 -13 27 -13 104 0 57 -3 111 -6 120 -7 19 -45
21 -62 4z"/>
    </g>
</svg>
```

Listing 2: SVG example

Notification messages. Notification messages are part of the information component. In N2Sky, there are used two types of notification messages:

- Warning messages, which announce that something goes wrong as shown in “Fig. 19”. It can be an error in the application or an user workflow error.
- Informational messages give information about the event which will occur or already occurred as shown in “Fig. 20”.

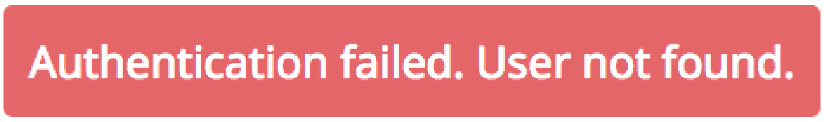

Authentication failed. User not found.

Figure 19: Customised N2Sky warning message UI element

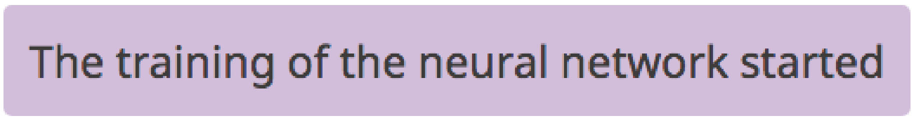

The training of the neural network started

Figure 20: Customised N2Sky informational message UI element

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

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

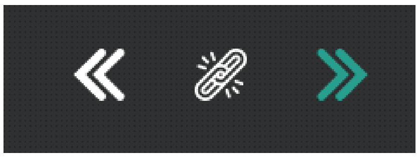
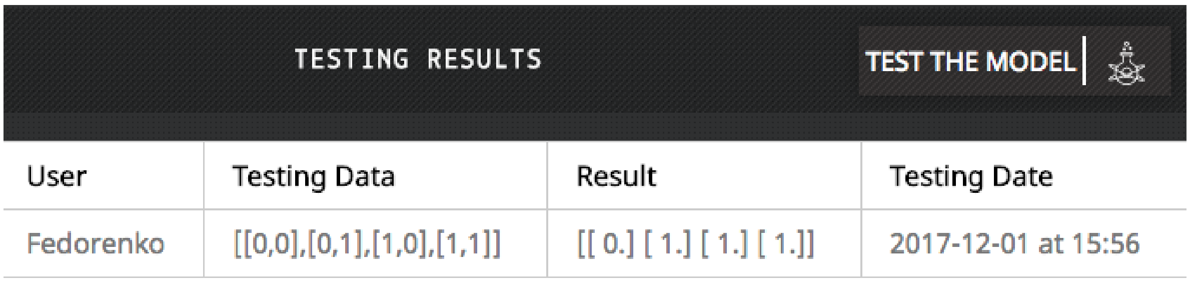


Figure 21: Customised N2Sky navigation UI element



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, the 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 shown in “Fig. 23”.

Grid item contains following UI elements:

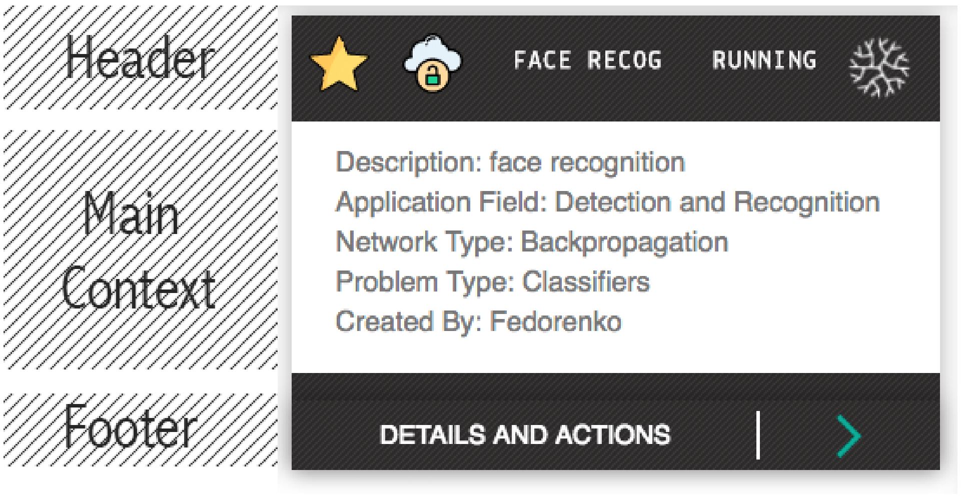
Header. The first component on which the user focuses, that is why it should be short but highlighted. Following UI elements are included:

- 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 customized. It is possible to put list of items, plain text or even image.

Footer. Optional element and contains only the button UI element.

Main navigation menu. Every N2Sky view uses a main navigation menu. The menu is injected in an abstract view, which is extended by all other view components. The menu has menu items, which contain a caption and an icon. As it was mentioned before, N2Sky has two modules: administration module and main application



Description: face recognition
Application Field: Detection and Recognition
Network Type: Backpropagation
Problem Type: Classifiers
Created By: Fedorenko

Figure 23: Responsive N2Sky Grid Item UI comonent

module. Both modules are represented in the menu and the menu items visibility depends on the logged-in users permission. The menu for arbitrary user is shown in figure 24 and has following menu items:

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

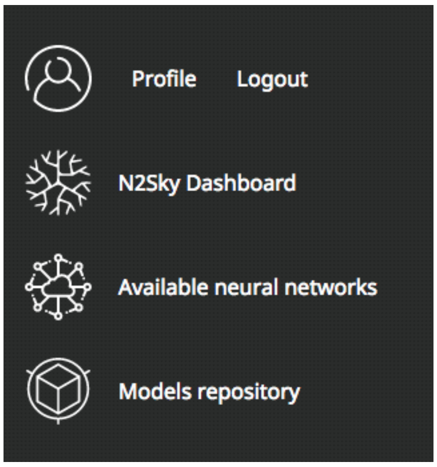


Figure 24: N2Sky Main Navigation Menu for arbitrary user

If the end-user is a system administrator, an additional menu form administration module will appear as demonstrated in figure 25. This menu contains dropdown submenus:

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

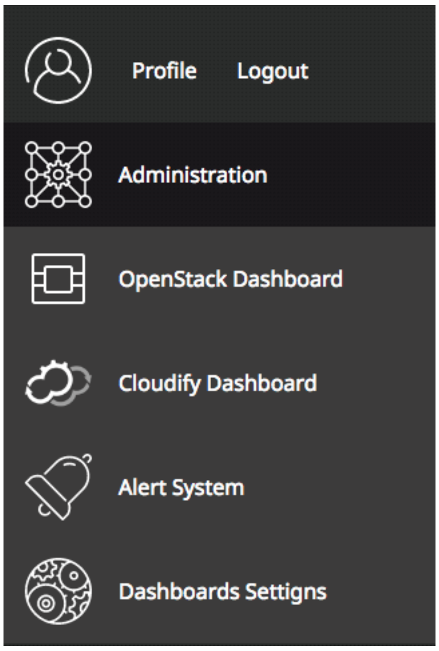
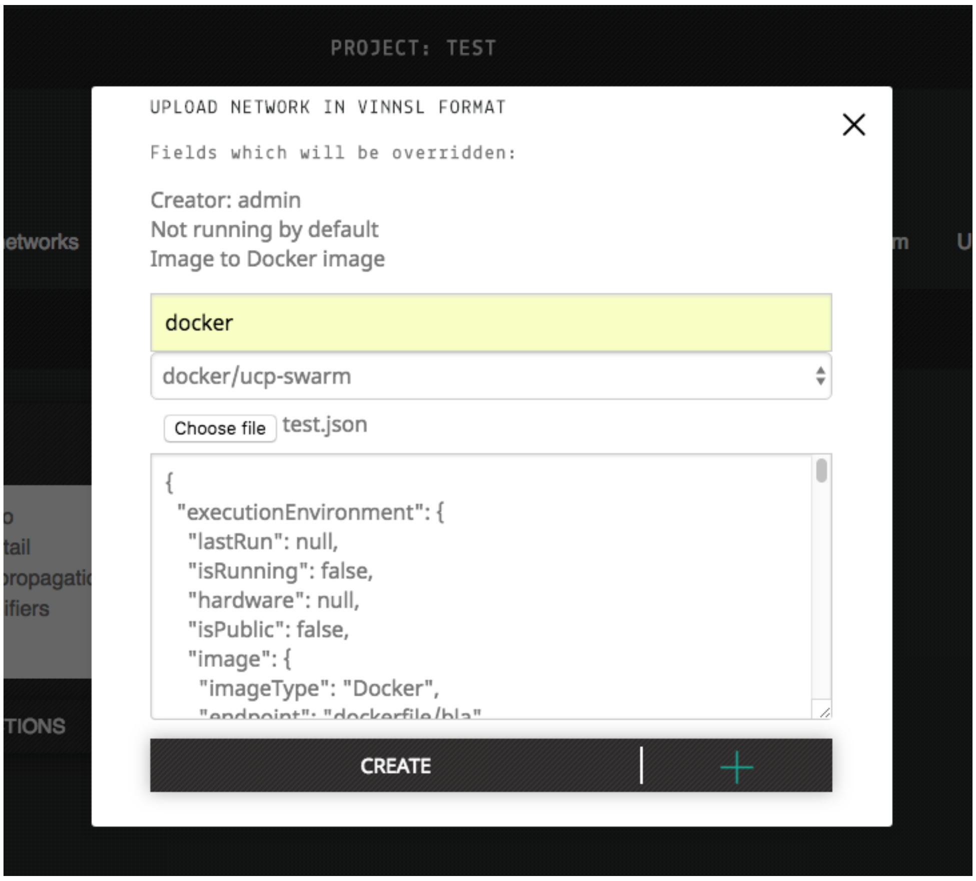


Figure 25: N2Sky Main Navigation Menu for system administrator

Modal windows. N2Sky uses modal windows almost in every view as shown in figure 26. Modals are responsive and touch screen friendly. It has 3 elements:

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



```
{
  "executionEnvironment": {
    "lastRun": null,
    "isRunning": false,
    "hardware": null,
    "isPublic": false,
    "image": {
      "imageType": "Docker",
```

Figure 26: N2Sky Modal Window UI component

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

3.2. N2Sky Services

N2Sky implements the microservices architecture. It has three main web services as shown in figure 8:

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

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

The above mentioned web services and also the processes and services that they encapsulate are in detail described:

3.2.1. User Management Web Service.

This web service is responsible for permissions and user management. It has its own database. The user can authorize the system and get a session token. Every token is a unique collection of numbers and Latin letters. The token is assigned to the authorized user and will be saved until the user is active. If the user is not active in the next 3 hours, the session token will disappear. If the user still has an active browser session, the authorization token will be revalidated. If the user is trying to make an illegal request or appears to have an overly active behaviour, the authorization token will be revoked and the system administrator will be notified of this incident.

Every user encapsulates permission levels. There two different types of permissions:

- Administrator permission. The user has fully granted permission through the entire application.
- Regular permission. The user has access only for his own as well as publicly available resources.

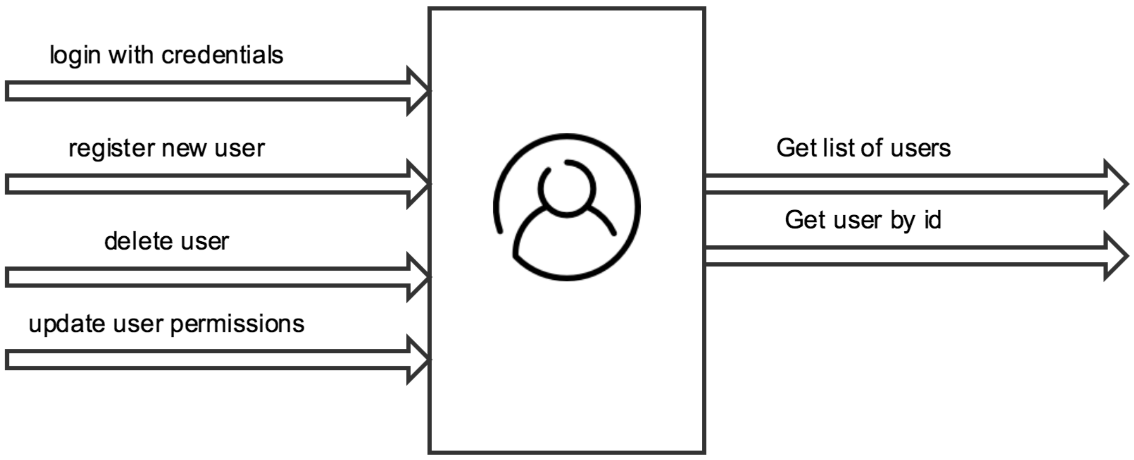


Figure 27: N2Sky User Management Web Service

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

- 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 can be found in subsection 9.2

3.2.2. Model Repository Web Service.

Model Repository Web Service is the core service of N2Sky. The authorized user can create a new project, add their neural network from a chosen paradigm or deploy his own one. Every newly created project is assigned to one user and can not be shared. Only the system administrator can look up into other users projects. This functionality is also limited by User Management Web Service.

Using this service, the user can create a neural network from proposed paradigms as well as upload his own neural network in the ViNNSL format [4]. This functionality is exposed via the service so that each user can use it either from N2Sky web portal or via HTTP request directly on web service.

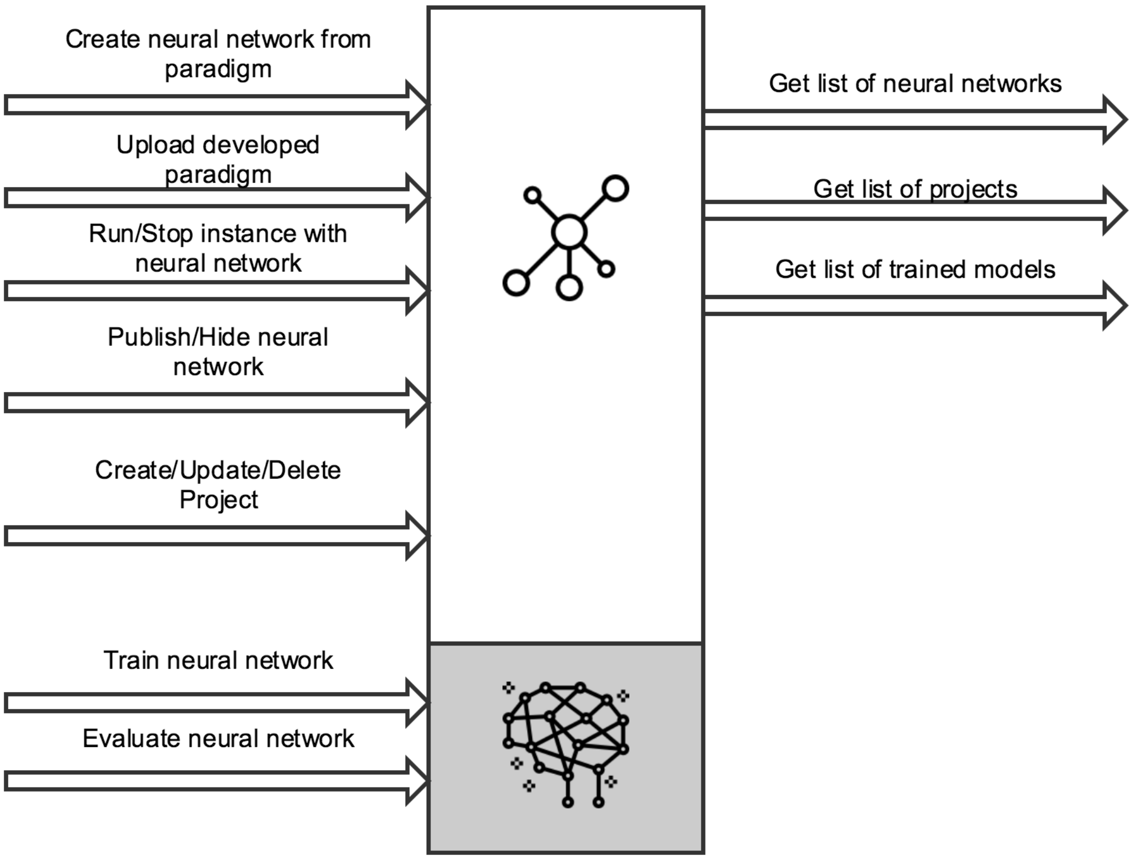

Create neural network from paradigm
Upload developed paradigm
Run/Stop instance with neural network
Publish/Hide neural network
Create/Update/Delete Project
Train neural network
Evaluate neural network
Get list of neural networks
Get list of projects
Get list of trained models

Figure 28: N2Sky Model Repository Web Service

As shown in figure 28 Model Repository Web Service has following accessible services:

- Create neural network from paradigms
- Upload developed neural network paradigms
- Run/Stop instances of neural networks
- Publish/Hide neural networks
- Create/Update/Delete Project with neural networks
- Get list of neural networks
- Get list of projects
- Get list of trained models

- Train neural networks (accessible only for the model management service itself)
- Evaluate neural networks (accessible only for the model management service itself)

Detailed information about web service API and API documentation can be found in subsection 9.2

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

Training service. This service provides the neural network training functionality. It is not possible to perform training by making direct requests on the service endpoint.

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

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

If the user performs training on his own neural network he can also log results 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 a trained neural network model. Since there is absolute freedom by neural network structure customization, the testing process can be inefficient and could take 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 the system administrator. This service can manage the OpenStack environment and Cloudify container management system. The monitoring management system service is installed in every OpenStack instance as well as on the OpenStack itself. It has its own metric configurations and alerting rules. The monitoring service is only exposed via Cloud management web service.

Cloud management service supports the 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.

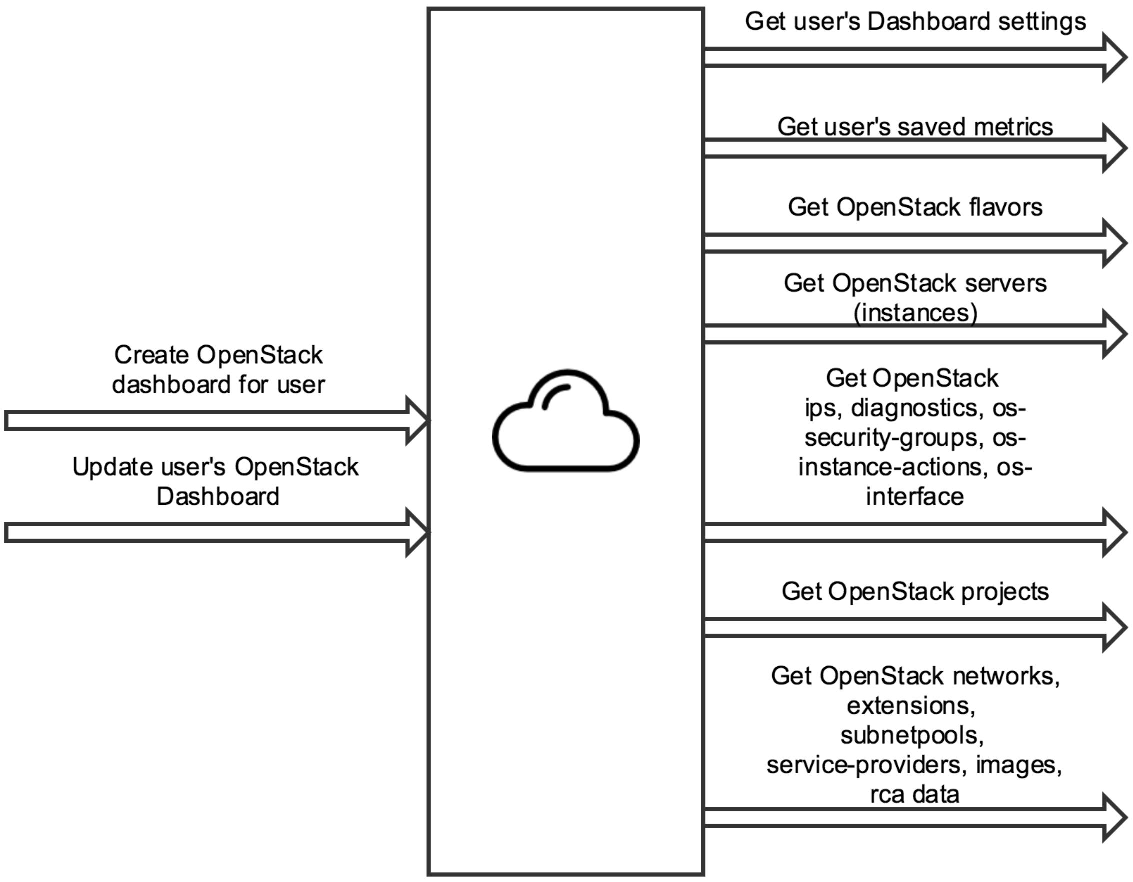

Get user's Dashboard settings
Get user's saved metrics
Get OpenStack flavors
Get OpenStack servers (instances)
Get OpenStack ips, diagnostics, os-security-groups, os-instance-actions, os-interface
Get OpenStack projects
Get OpenStack networks, extensions, subnetpools, service-providers, images, rca data
Create OpenStack dashboard for user
Update user's OpenStack Dashboard

Figure 29: N2Sky Cloud Management Web Service

As shown in figure 29 Cloud Management Web Service has following accessible services:

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

Detailed information about web service API and API documentation can be found in subsection 9.2

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 a problem occurs. A system administrator could use a monitoring tool such as Nagios for OpenStack in order to get information about each servers state. Monitoring tools can give information about the system, network, and infrastructure [3]. The purpose of the monitoring tool is to detect issues and inform the system administrator about the occurred problem. This tool cannot solve the problem, but it can give information on what might have failed in this process.

3.3.1. Monitoring Requirements

The base monitoring system is a readable and understandable representation of the graph. Graphs allow the user to identify monitored objects and recognise their metrics. A good monitoring graph gives meaningful description but also helps quickly in detecting and determining issues via representation. This kind of graph should serve as a motivation while solving problems. Some simple rules can be followed for a precise graph representation:

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. The metrics set should be small, one metric represent one object. There is no need to put multiple objects which do 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 high frequency, it is better to use a multiline area. A stacked area might be better on longer time series but with a bigger metric set respectively.

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

Data hierarchy. It is necessary 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 give the actual graph representation. The nested level is a multiple line kind of metrics. All mentioned data should be visible and accessible.

Clarity. Designing a chart is significant. There are multiple devices with different screen resolution. Too many lines on limited space will make the chart unreadable. If there are lots of charts on one page, users will be confused on what the meaning of that actually is. A good practice would be 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 user-oriented. Today, a simple and clean appearance of the application is relevant when choosing an appropriate design when a large number of charts are available.

Control and managing. It should be possible for any user to manipulate a graph. Either to change time series or remove metric, customization of the graph makes the whole application more attractive.

3.3.2. Applying Monitoring

To build continues monitoring system, there is a need to use a toolkit with an active ecosystem. While searching for a proper toolkit one should keep in mind some specific requirements that are going to be used 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 time series to repository.
- Scalable.

After researching, it was decided on the Prometheus Monitoring Toolkit [21]. This tool supports all demanded requirements. One of the most interesting features is that Prometheus can be used on any UNIX environment. Prometheus will match exactly our needs since the multiple instances of OpenStack which can have different operating systems. Originally, Prometheus was created by SoundCloud team in 2012. The core of monitoring application is the Prometheus server, which collects time series data from the moment it was executed on the environment as shown in figure 30. All components are written in Go language and support multiple modules for monitoring different environment metrics. To understand the nature of Prometheus it is necessary to explain its architecture.

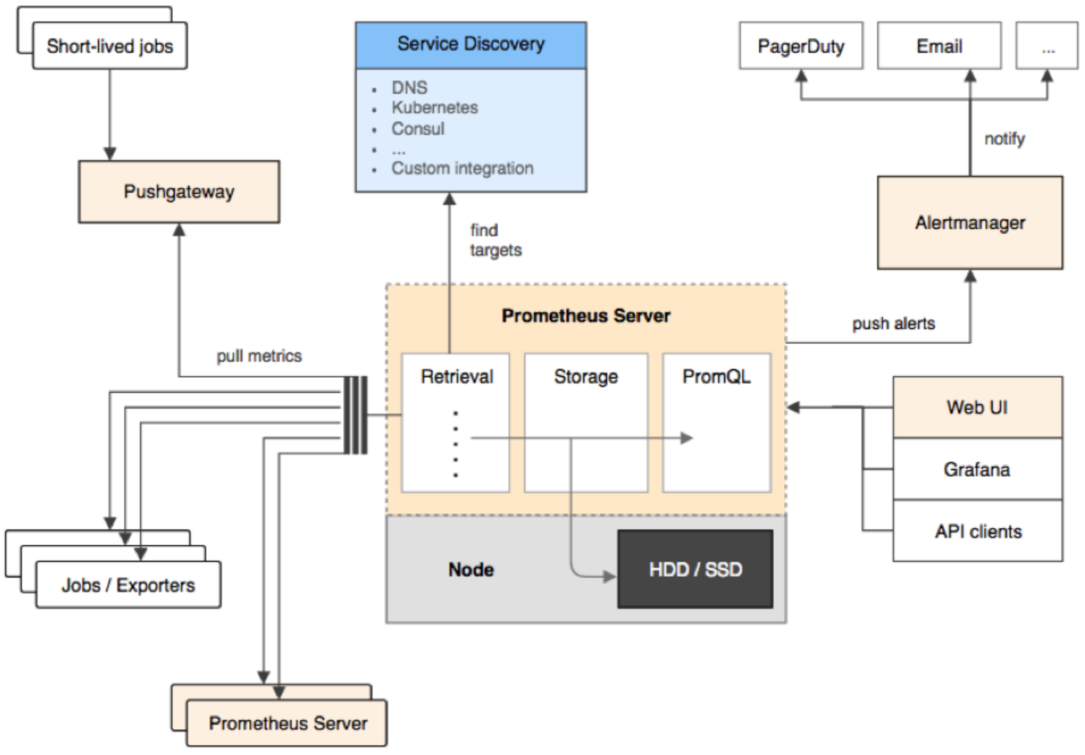


Figure 30: Prometheus monitoring architecture

The core Prometheus server pulls all metrics from jobs which are instrumented if the service is unavailable. For instrumentation, it can be pulled from 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 customized dashboards. The common components of Prometheus architecture:

The Prometheus server. This is the basic element in the whole architecture. The server includes services which collect, store and retrieve nodes. The principle is scrapping or pulling. This means that the data fetched with some interval can be configured and stored accordingly as a time series. Prometheus supports different modules and each module represents a specific node. The nodes expose these ports that Prometheus uses for retrieving the data. For example, in N2Sky is being used the Node Exporter Module which gives the possibility to collect almost all essential data like CPU, RAM, HDD/SSD etc.

Push gateway. There are nodes, which are not exposing endpoints. In this case, the data collection through this 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.

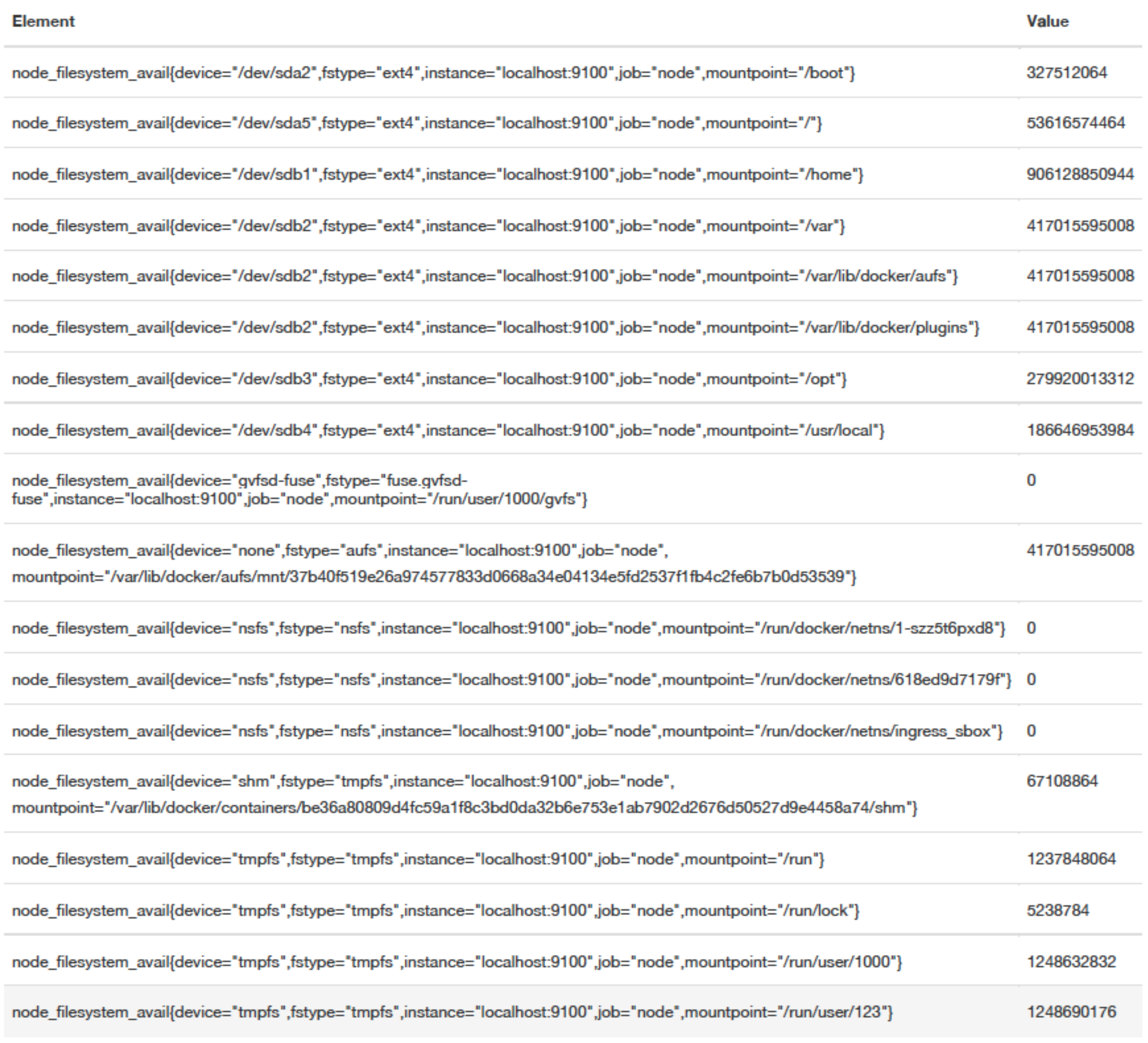
Alert Manager. Monitoring consists of multiple metrics and each metric can be analyzed. It is possible to subscribe to particular metric in order to detect metric behaviour, namely metric deviations. The Alert Management System used for firing events. It is possible to receive alert notification over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

Metric notation Following example represent metric notation:

```
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 operating system has a different metric naming. N2Sky will propose the list of available metrics. In curly brackets is defined the type of request, an endpoint and the expected HTTP response status.

After executing this query, the data will be retrieved from logs and represented as a time series as shown in figure 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="nsfs",fstype="nsfs",instance="localhost:9100",job="node",mountpoint="/run/docker/netns/1-szz5t6pxd8"}	0
node_filesystem_avail{device="nsfs",fstype="nsfs",instance="localhost:9100",job="node",mountpoint="/run/docker/netns/618ed9d7179f"}	0
node_filesystem_avail{device="nsfs",fstype="nsfs",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 of our monitoring system is scalability. One of the greatest features of Prometheus is that even if the environment is going to be overloaded, it will generate the same amount of metrics anyway. Hence the amount of events it still is independent of the amount of generated time series. When mentioning requirements, the possibility exists to build joint metrics, namely build multiple time series using this kind of metric:

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

Gauge The gauge is a metric, which also represents a simple numerical value like a 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 represents observations. It is stored in a bucket, which can be pulled. Any bucket can be configured depending on the need. It can be the sum of values or count of events, which are observed.

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

3.3.3. Integration with N2Sky

Prometheus supports query language, which is the key feature of this tool. The Prometheus query language, or promql, is expressive. With Prometheus, a self-described metric name can be chosen. Prometheus converts all metrics so that every human can understand what eacg particular metric means. The previous example with a metric "node_filesystem_avail" is a good one to show how it actually works. This metric shows the folders on root and available memory on each.

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

Listing 3: Prometheus query

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

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

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

Listing 4: Prometheus short query

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

For building a proper monitoring dashboard, it is important to provide customisation. Prometheus supports time duration:

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

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

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

Listing 5: Prometheus alerting rule

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

In N2Sky, was developed such a monitoring service that uses the micro services approach like an 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 adds complexity. It was decided that the user should give only the time interval and step. In an example when the user wants to see the CPU load for the last hour with a step 30 seconds. It makes the creation of metric more intuitive, no more range like "from", "to" and other types of ranges. All this can be solved with one simple request. The second part is the storing of metrics. Instead of every time building a query, the monitoring service will save the requested by user metric. In this case, every user will get his own customised metric. The service uses Mongo DB for storing of the metric configuration. Every collection has its own schema. When a user makes a request to save a metric, the schema has to be filled with the requested by user data.

The benefit of the microservices architecture is the independence of each service. The good part of it is that when one of the services is not be accessible, the rest of them would still be up and running. The problem is that in order to make the application running in the correct way, all services are required. It is important to detect events and rollbacks or transactions in case if an error or failure occurs. N2Sky has its own monitoring and alerting system, which notifies the system administrator about failures. The problem is how to handle events, which are transactionally dependent from service to service in case of an occurring error and how an event is handled when still in transaction. There is two main kind of events [5]:

- *Intrinsic Events.* Process steps starting and finishing generate events intrinsic to the process model. This kind of events contains the process step or failure.
- *Context Events.* Events stemming from the context of the process.

It is impossible to maintain each event and process an event stream in the N2Sky application. If other developers forget to put the event into the stream, it will unnoticeable. That is why in N2Sky are events intrinsic, which log every transaction from one service to another. In case of an error, the alerting event will be fired with a log data which contain the step and the error itself.

3.3.4. Monitoring Dashlet Design

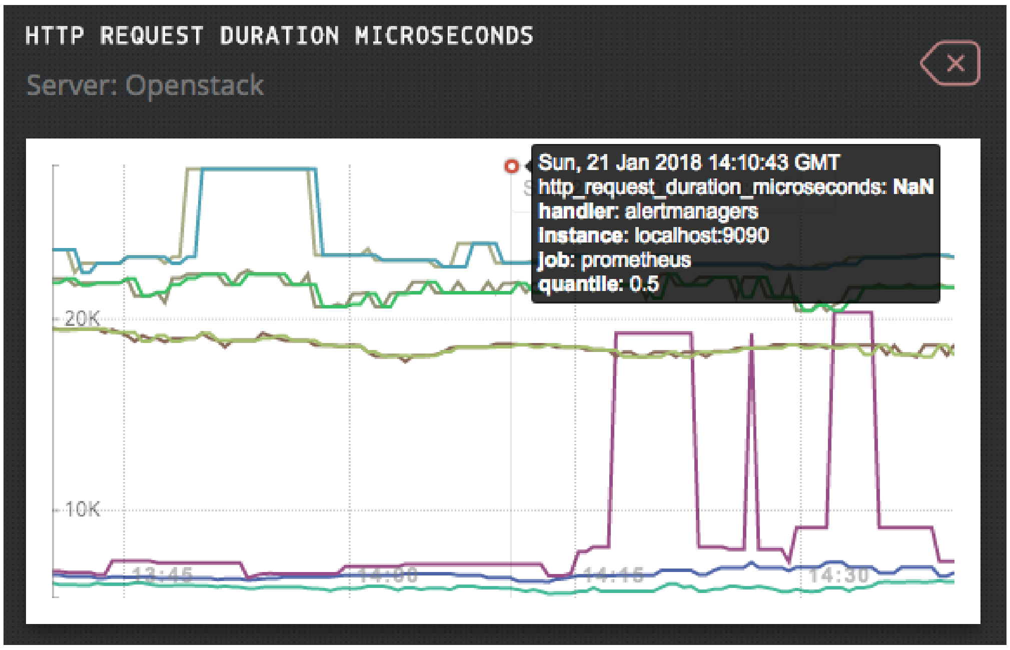
Since multiple machine and services are being monitored, there was a need to create a dedicated dashboard design. At first, the to-be-monitored environments would be:

Openstack Machine. It is a 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 parts of application design is to maximize reusability of the components.


HTTP REQUEST DURATION MICROSECONDS

Server: Openstack

Sun, 21 Jan 2018 14:10:43 GMT
http_request_duration_microseconds: NaN
handler: alertmanagers
instance: localhost:9090
job: prometheus
quantile: 0.5

Figure 32: N2Sky Monitoring Dashlet

The figure 32 shows a HTTP request directions in milliseconds metrics. Each dashlet item contains:

- Title, which represents a readable and self-described metric name.
- Server, which shows to which instance this metric belongs.
- Chart, which is the monitoring data itself.
- Tooltip, which appears on metric mouse over. Tooltip shows following data:
  - Date of the particular position.
  - Name of metric.

- Instance environment.
- Handler is an alert manager listener (reference to subsection 3.4).
- Cron job.
- Quintile.

3.4. Alerting Management System

Today, the most trending subject in the monitoring area is a prediction and automated detection. It makes the system-administration free from 24/7 managing, maintaining, and monitoring system. For monitoring, the Prometheus tool is the used one. Since this tool is constantly saving data logs about the system, it is possible to reuse these 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 application. The idea behind is to have only one Alert Manager and have monitoring tools on multiple machines. If the machine goes down or even Prometheus itself, the Alert Manager can catch and deliver this event. To understand how the Alert Manager works, it is essential to understand the architecture of the whole system as displayed in figure 33.

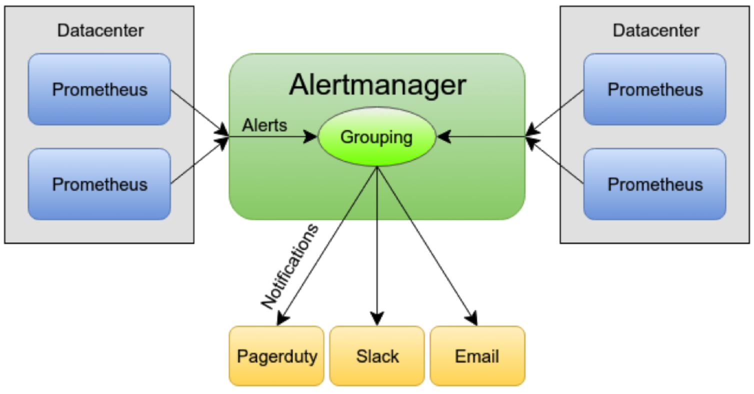


Figure 33: Alerting System Architecture

This architecture is a typical messaging platform. The Messaging Service sends messages to multiple clients. It is implemented in the Producer-Consumer Pattern. In the Alert Manager Architecture, the role of the producer is taken by Prometheus Datacenter. Alert Manager consumes the messages. The consumer knows nothing about the producer but just subscribes to the event. With this approach, multiple producers can be attached [32].

Alerts can be collected in groups by the datacenter. If an event occurs on multiple machines, it can be packed into one notification and accordingly fired . In the Prometheus configuration, two things should be set up: reference on Alert Manager and Alert Rules. When the Alert Manager consumes an event, it just dispatches it via notification as displayed in figure 34.

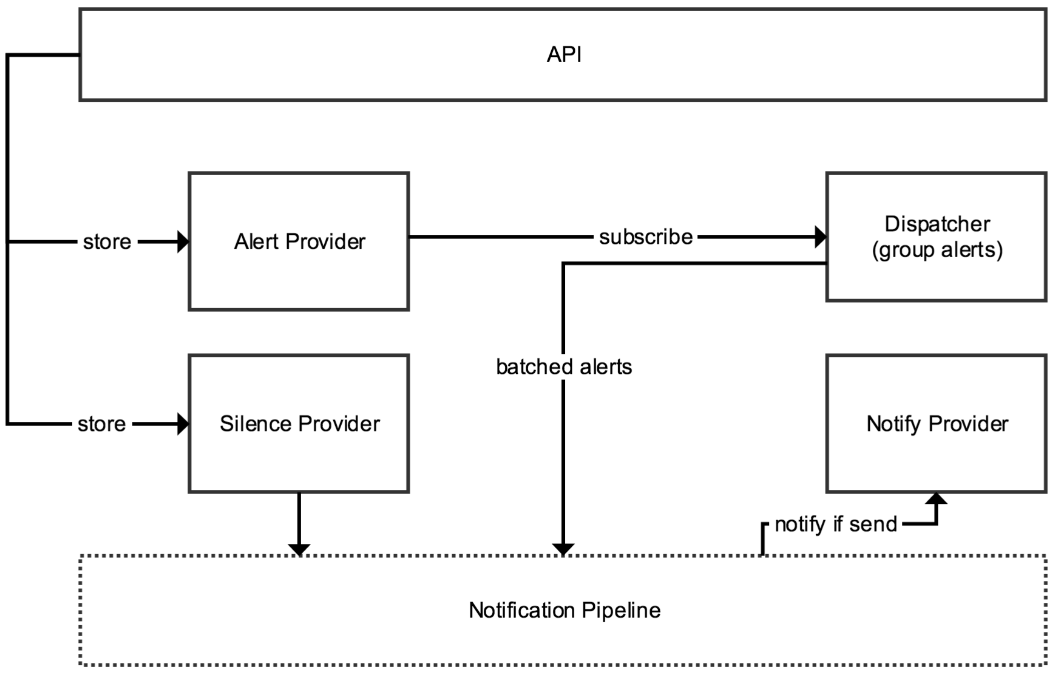


Figure 34: Communication within Alerting Management System

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

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

As a response the event information is sent [22].

```
[
 {
   "labels": {
     "<label>": "<value>"
   },
   "annotations": {
     "<label>": "<value>",
   },
   "startsAt": "<date>",
   "endsAt": "<date>"
   "generatorURL": "<url>"
```

```
    }
]
```

Listing 6: Querying alert

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

Silences. Silences are commands, which mute alerts for specific time. It can be configured via web interface.

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

Notification Pipeline. A pipeline which consists on channels, routers and filters. It takes rules from Alert Provider and Silence Provider and alerts from Dispatcher. If a notification is ready, it can be send.

3.4.2. Alert Rules

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

```
groups:
- name: test
  rules:
  - alert: Error
    expr: job:request_latency_seconds:mean5m
    {job="loclhost:5000"} > 0.5
    for: 5m
    labels:
      severity: page
    annotations:
      summary: latency
```

Listing 7: Alert error response

The following example of alerting rules shows the typical alerting rule. The most important part is an expression, which is applied to the Monitoring System. More details about the creation of alerting rules are located in the Development Guide chapter "Setting up Alert Management System" subsubsection 9.1.9.

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

Recording rules allow to pre-compute frequently needed expressions or expressions, which are resource or time-consuming. These rules are saving the 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 an alerting rule, it is possible to define an alert, i.e deviation by particular expression from the Prometheus Tool and its exports (modules). It allows the building of an alert even on the combined query.

In case that the Alert Manager is not available, all alerts are saved into the buffer. As soon as the Alert Manager will be online, all events will be fired sequentially.

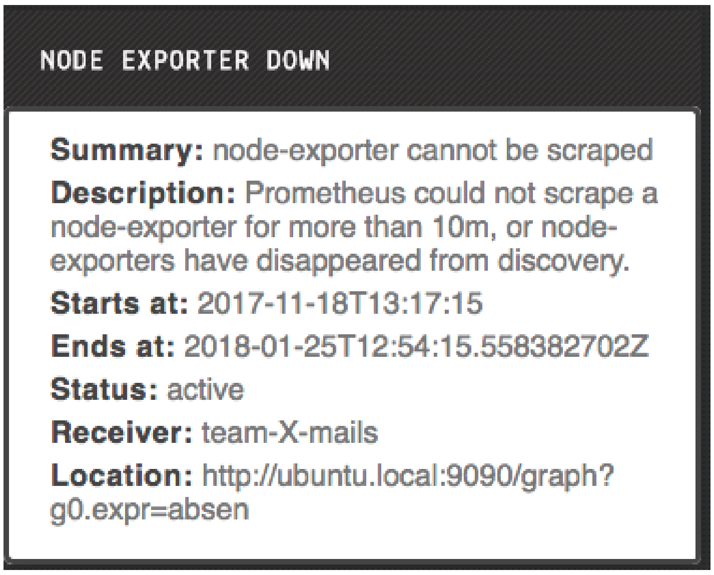
3.4.3. Integration with N2Sky

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

Alerting Client fire alerts depend on configured rules. The rules can be created via the user interface. The list of events N2Sky receives via its service it is fetched information from the Alert System.

3.4.4. Alerting System Design

Alerting System UI kept the same design as in the entire application. The N2Sky UI components were reused and no new component was created. The main component is a grid item which contains information about an event, which occurs as shown in figure 35.


NODE EXPORTER DOWN

Summary: node-exporter cannot be scraped
Description: Prometheus could not scrape a node-exporter for more than 10m, or node-exporters have disappeared from discovery.
Starts at: 2017-11-18T13:17:15
Ends at: 2018-01-25T12:54:15.558382702Z
Status: active
Receiver: team-X-mails
Location: http://ubuntu.local:9090/graph?g0.expr=absen

Figure 35: N2Sky Alerting Management System Event representation.

The following information is shown:

Title. The name of alert.

Summary. It is a short description of the alert. If the summary is too long, it will be cut.

Description. The whole description of the alert.

Starts at. Timestamp when the event occurred.

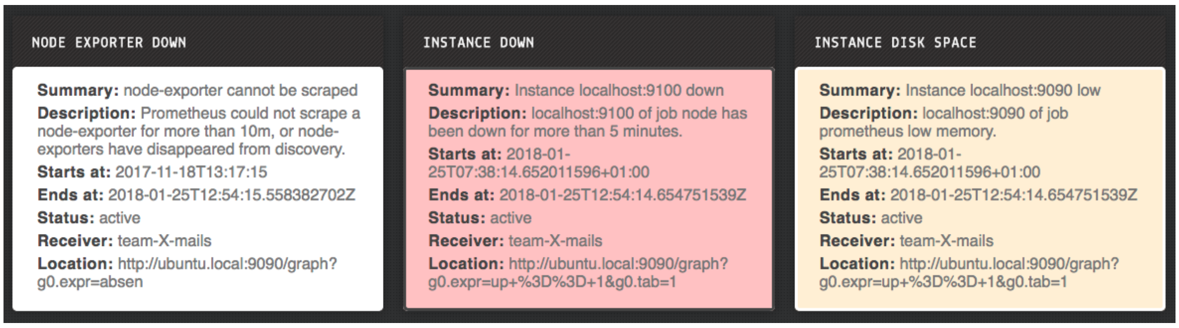
Ends at. Timestamp when the event is no more valid. If the timestamp is a current day and time, the problem is not yet fixed.

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

Receiver. The group of receivers. Normally, the group contains multiple receivers emails.

Location. The endpoint of the server where the monitoring system is installed.

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



NODE EXPORTER DOWN

Summary: node-exporter cannot be scraped
Description: Prometheus could not scrape a node-exporter for more than 10m, or node-exporters have disappeared from discovery.
Starts at: 2017-11-18T13:17:15
Ends at: 2018-01-25T12:54:15.558382702Z
Status: active
Receiver: team-X-mails
Location: http://ubuntu.local:9090/graph?g0.expr=absen

INSTANCE DOWN

Summary: Instance localhost:9100 down
Description: localhost:9100 of job node has been down for more than 5 minutes.
Starts at: 2018-01-25T07:38:14.652011596+01:00
Ends at: 2018-01-25T12:54:14.654751539Z
Status: active
Receiver: team-X-mails
Location: http://ubuntu.local:9090/graph?g0.expr=up+%3D%3D+1&g0.tab=1

INSTANCE DISK SPACE

Summary: Instance localhost:9090 low
Description: localhost:9090 of job prometheus low memory.
Starts at: 2018-01-25T07:38:14.652011596+01:00
Ends at: 2018-01-25T12:54:14.654751539Z
Status: active
Receiver: team-X-mails
Location: http://ubuntu.local:9090/graph?g0.expr=up+%3D%3D+1&g0.tab=1

Figure 36: N2Sky Alerting Management System Severity Level

There are three types of severity levels:

- Critical, which shows that a crucial event occurs. For example, if the server goes down, it is a critical severity level. In N2Sky, the UI is red.
- Warning, which shows that something went wrong. For example, not enough disk space on the server. In N2Sky, the UI is orange.
- Page, which shows information. For example, lots of requests occurred. In N2Sky, this UI is white.

4. Vienna Neural Network Specification Language

In this chapter will be described the history of the ViNNSL language, its development, and new features, which are applied in the current N2Sky System.

4.1. ViNNSL Development

Vienna Neural Network Specification Language (ViNNSL) is the language, which is used for the description of the semantic and behaviour of the neural network paradigm. ViNNSL gives the possibility to create dynamic services, which can differentiate the behaviour. The ViNNSL has been seen as a semantic language standard. The language uses schemas, which help users to describe attributes like service capabilities, semantic, functions, and parameters [4].

ViNNSL consist of five parts:

- Description Schema
- Definition Schema
- Data Schema
- Result Schema

This approach with multiple schemas was used in Neural Network Cube (N2Grid) System. This system is a web-based neural network simulator used by students and researchers [30]. Today, N2Sky is the follower of the N2Grid ideas.

In 2015, the ViNNSL 2.0 was released, which was an extension of the previous version [28]. Basically, almost nothing changed, only additional fields were added:

- *Creator*
- *Problem Domain*
- *Propagation Type*
- *Learning Type*
- *Application Field*
- *Network Type*
- *Problem Type*
- *Execution Environment* The idea was proposed by the concept of the N2Grid [30]. It is possible to define the environment, which is needed for particular neural network instances.

- *Execution Type.* Neural networks will be executed either sequential or parallel. Additionally, the neural networks architecture and metadata could be defined.
- *Result Schema* became part of the ViNNSL Description. Earlier, it was a separated schema, but the data schema and instance schema still remained the same.
- *Parameters* The user could define input as well as output parameters. The training file and the file type could also be determined in the schema.

4.2. ViNNSL Template

After studying ViNNSL with its extension, it was decided to simplify all schemas and most importantly adapt it to the current N2Sky System.

The idea behind is to use the ViNNSL template, which has additional fields from ViNNSL 2.0. The fields can be fully processed and be removed after creating the neural network. No more extra objects or fields in other application. ViNNSL template allows users to apply the ViNNSL language across the whole N2Sky platform. The detailed information is shown in Appendix A

4.2.1. Metadata template

As before mentioned, the ViNNSL 2.0 contains metadata about the user and neural network.

```
...
        "problemDomain": {
                "propagationType": {
                        "value": null,
                        "possibleValues": ["feedforward"],
                        "learningType": {
                                "value": null,
                                "possibleValues": ["definedconstructed",
                                "trained", "supervised", "usupervised", "linear"]
                        }
                },
                "applicationField": {
                        "value": null,
                        "possibleValues": ["AccFin", "HealthMed", "Marketing",
                         "Retail", "Insur", "Telecom", "Operations", "EMS"]
                },
                "problemType": {
                        "value": null,
                        "possibleValues": ["Classifiers", "Approximators",
                         "Memory", "Optimisation", "Clustering"]
                },
                "networkType": "Backpropagation"
```

```
            }
...
```

Listing 8: ViNNSL template metadata

From the code above, it is possible to spot additional values:

- *value* is the default value of the object.
- *possibleValues* parameter, which contains an array of possible value. The user can choose one of the value.

This approach not only helps to add templating into the current ViNNSL schema, but also adds some rules upon it:

- If the default value is set and there are no more values under "possibleValues" parameter, then the value is impossible to change.
- If there is no value for "possibleValues" and "value" fields, then the user can type a customized value.
- If the "value" has some parameter and "possibleValues" field is also not empty, it means the "value" parameter is the default value and it can be changed by "possibleValues".

4.2.2. Environment

In previous versions of the ViNNSL language was possible to define the execution environment. This possibility is still available, but some additional values were added in order to apply the templating approach.

In the current version of N2Sky, the user can deploy neural network on the N2Sky cloud as well as on the private cloud. The "endpoints" parameter was added.

```
...
    "endpoints": [{
        "name": "train",
        "endpoint": null
    }, {
        "name": "test",
        "endpoint": null
    }]
...
```

Listing 9: ViNNSL template environment

Since it is an array parameter, it gives room for adding new endpoints in case that it will be needed in future versions of N2Sky.

4.2.3. Neural Network Structure Template

The structure of the neural network was also changed. The template looks pretty empty, but from that is possible to generate any type of structure based on ViNNSL language.

```
...
"structure": {
                "inputLayer": {
                        "result": {
                                "nodesId": []
                        },
                        "config": {
                                "dimensions": {
                                        "min": 1,
                                        "max": 1
                                },
                                "size": {
                                        "min": 960,
                                        "max": 960
                                }
                        }
                },
                "hiddenLayer": {
                        "result": {
                                "dimensions": [{
                                        "id": null,
                                        "nodesId": []
                                }]
                        },
                        "config": {
                                "dimensions": {
                                        "min": 1,
                                        "max": 1
                                },
                                "size": {
                                        "min": 960,
                                        "max": 960
                                }
                        }
                },
                "outputLayer": {
                        "result": {
                                "nodesId": []
                        },
                        "config": {
                                "dimensions": {
                                        "min": 1,
                                        "max": 1
                                },
                                "size": {
                                        "min": 960,
                                        "max": 960
                                }
                        }
                },
...
```

Listing 10: ViNNSL template structure

The structure contains input, hidden and output layer. The hidden layer has "dimensions" parameter, which allows creating multiple hidden layers.

Every layer has "config" property, which shows how many dimensions can be used. This config is scalable and additional configuration can be added.

Furthermore, every layer has a number of nodes, namely the neurons id.

In the structure, it is important to define the connections between nodes.

```
...
                "connections": {
                        "fullyConnected": {
                                "isConnected": null
                        },
                        "shortcuts": {
                                "isConnected": null,
                                "connections": [{
                                        "from": null,
                                        "to": null,
                                        "isFullConnected": null
                                }]
                        }
                }
        }
...
```

Listing 11: ViNNSL template connections

There are few types of connections:

- *fullyConnected* parameter, which has boolean "isConnected" in case that the neural network is fully connected or not.
- *shortcuts* is a parameter, which also has boolean flag parameter. If there are any types of the shortcuts, it will be set to true. Inside of this parameter, the array field "connections" has "from" and "to" nodes id parameter. In case that the full connection remains the same "isFullConnected" will be set to true.

4.2.4. Parameters Template

The templating was also applied to the input parameters. The structure of the template is pretty similar to metadata template.

```
...
"parameters": {
                "input": [
                        {
                                "parameter": "activationFunction",
                                "defaultValue": "sigmoid",
                                "possibleValues": ["sigmoid", "relu", "softmax"]
                        },
                        {
                                "parameter": "activationFunctionHidden",
                                "defaultValue": "relu",
                                "possibleValues": ["sigmoid", "relu"]
                        },
...
```

Listing 12: ViNNSL template input parameters

Additionally, in this template the "parameter" field is used. This field defines an actual input parameter and has to be unique.

The output parameter is fixed and can not be changed. This parameter will be used for trained model evaluation in case that the user does not want to type his own testing data.

The training file information is also defined under parameters and can be changed only by the neural network owner.

4.3. ViNNSL Generation

After the neural network contributor user fills out the ViNNSL template, the user can generate the ViNNSL Description schema. This schema will be used for neural network training. The generation happens on the N2Sky platform and the ViNNSL Description file can be totally different, but the rules will be followed. The user now has a possibility to customize the structure, connection, metadata and parameters of the neural network. Due to restricted permissions of the ViNNSL template, it is impossible to mistype the schema.

The contributor user can also use ViNNSL in order to fill out the structure, connection, metadata and parameters of the neural network. If the user does it manually, without the N2Sky "3-step-view" generator, he has to upload his ViNNSL description in the N2Sky platform and the neural network will be automatically created.

4.3.1. Generated Metadata

In the generated metadata, the information is filled out from N2Sky platform.

All unused fields will be removed automatically. The typical parameters, like "possibleValues", will be removed everywhere. The training parameters will remain the same.

```
...
        "problemDomain": {
                "problemType": "Classifiers",
                "networkType": "Backpropagation",
                "applicationField": [
                        "AccFin"
                ],
                "propagationType": {
                        "propType": "feedforward",
                        "learningType": "supervised"
                }
        },
        "creator": {
                "name": "fedorenko",
                "contact": "andriifedorenko@gmail.com"
        },
        "metadata": {
                "name": "XOR Test",
                "description": "test",
                "paradigm": "Backpropagation",
...
```

Listing 13: Generated ViNNSL model

The detailed information is shown in Appendix B

4.3.2. Generated Structure

The structure is the most complicated part of the ViNNSL generation if the user does it manually without N2Sky generator.

Every node will receive the id, which will be parsed by N2Sky.

- *Input layer as well as output layer* have the same structure of the node ids.

```
...
                "inputLayer": {
                        "amount": 3,
                        "nodesId": [
                                "1-input",
                                "2-input",
                                "3-input"
                        ]
                },
                "outputLayer": {
                        "amount": 1,
                        "nodesId": [
                                "1-output"
                        ]
                },
...
```

Listing 14: ViNNSL generated layers

The amount of the nodes will be added. The nodes id has the following structure:

1. The unique id of the node, which also defines the order
2. The name of the layer

- *The hidden layer* can be multidimensional, that is why the id of the layer, as well as the id of nodes, are required.

```
...
                "hiddenLayer": [
                        {
                                "id": "1-hidden-layer",
                                "amount": 4,
                                "nodesId": [
                                        "1-node-1-hidden-layer",
                                        "2-node-1-hidden-layer",
                                        "3-node-1-hidden-layer",
                                        "4-node-1-hidden-layer"
                                ]
                        },
...
```

Listing 15: ViNNSL generated hidden layers

The id of the layer is defined by the first number, which is also the order of the layer.

The id of the nodes is defined by the following rules:

1. The unique id of the node, which also defines the order
2. The id of the layer
3. The name of the layer

4.4. ViNNSL Model

The ViNNSL Model derives from the ViNNL Description trained model. Basically, the input parameters were taken from the ViNNSL Description and default values were replaced with the trained data. Additionally, it was decided to put the following parameters:

- *rawModel* is the raw training model, which can be used for the model evaluation.
- *logs.* The logs are appending during training. When the training is done, the logs will be removed from the neural network and added as a parameter to the model.
- *test* is the array of performed tests. The test results are saved in the model in order to stop spreading multiple schemas.

```
...
        {
                "createdOn": "2018-02-24T12:24:38.198Z",
                "result": "[[0.]\n [1.]\n [1.]\n [1.]]",
                "testing_data": "[[0,0],[0,1],[1,0],[1,1]]",
                "user": "admin"
            }
 ...
```

Listing 16: ViNNSL trained model testing

  The test has following data:
  - *createdOn* timestamp of the testing
  - *testing_data* with which the model was tested
  - *result* actual result of the testing
  - *user* the user, who performed test.
- *paramters* the input parameters, where default values were replaced with the actual values.

Detailed example of the ViNNSL model is located in Appendix C

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 the Monitoring System and Alert Management System. AMC is integrated into N2Sky cloud platform in order to support the 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 have full knowledge about the domain with corespondent permissions or domain owner, can have 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
- Control OpenStack dashboard and its services
- Control Cloudify dashboard and its services
- Manage monitoring charts
- Manage alerts and alerting rules
- Manage the OpenStack instances (servers)
- Has an access to dashboards of other users

5.3. Administration Dashboard

Administration Dashboard represents the overview of administration tools and important metrics, which are displayed in figure 37.

After System Administrator authorise in the system, he will be automatically redirected to the Administration Dashboard. By default, URL link is:

```
<host>/cloud
```

Administration tools have to be in focus of the main dashboard view as shown in figure 38. The collection of elements has to be under the title bar element, which is named "ADMINISTRATION TOOLS". Every element of administration tool contains an icon in SVG [6] format and caption under it. Following administration tools have to be represented:

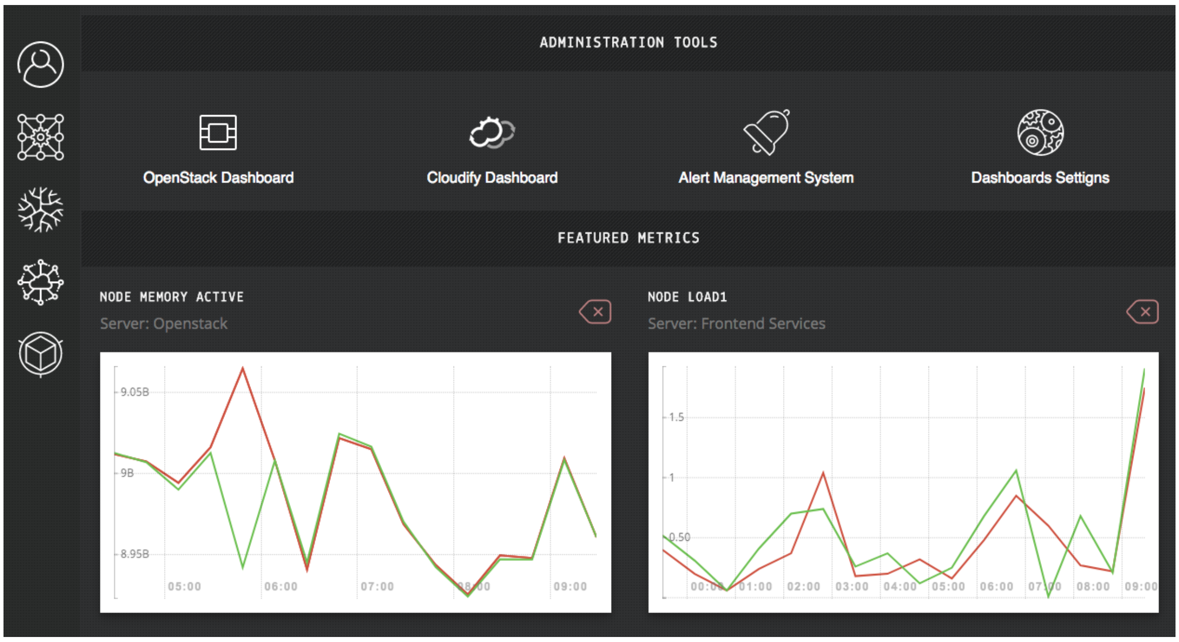


Figure 37: N2Sky Administration Dashboard.

- OpenStack Dashboard. Link to view:

```
<host>/openstack
```

- Cloudify Dashboard. Link to view:

```
<host>/cloudify
```

- Alert Management System. Link to view:

```
<host>/alert
```

- Dashboards Settings. Modal popup window without redirection

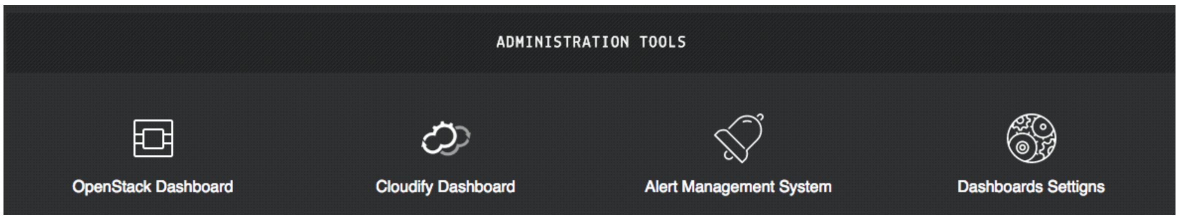


Figure 38: N2Sky Administration Dashboard. Administration tools component.

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

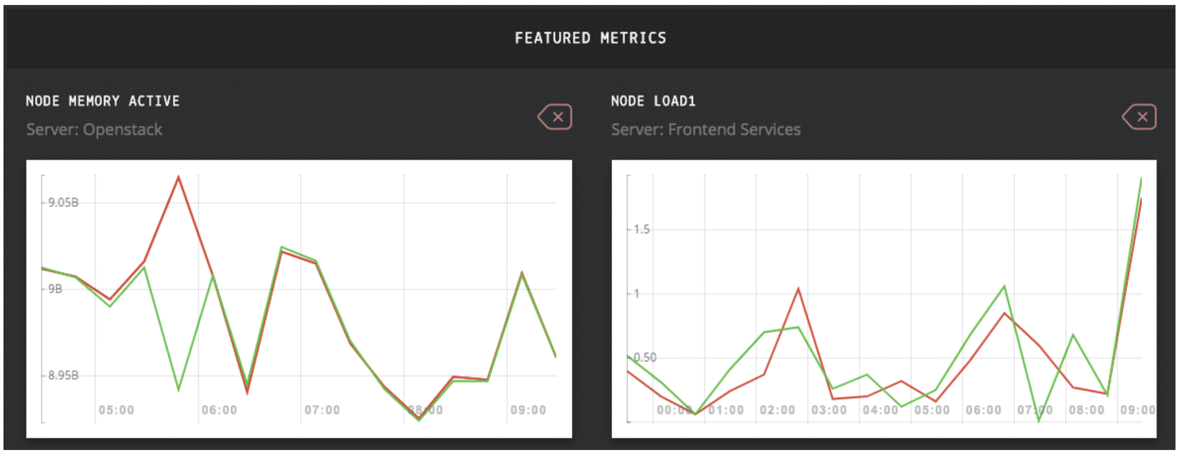
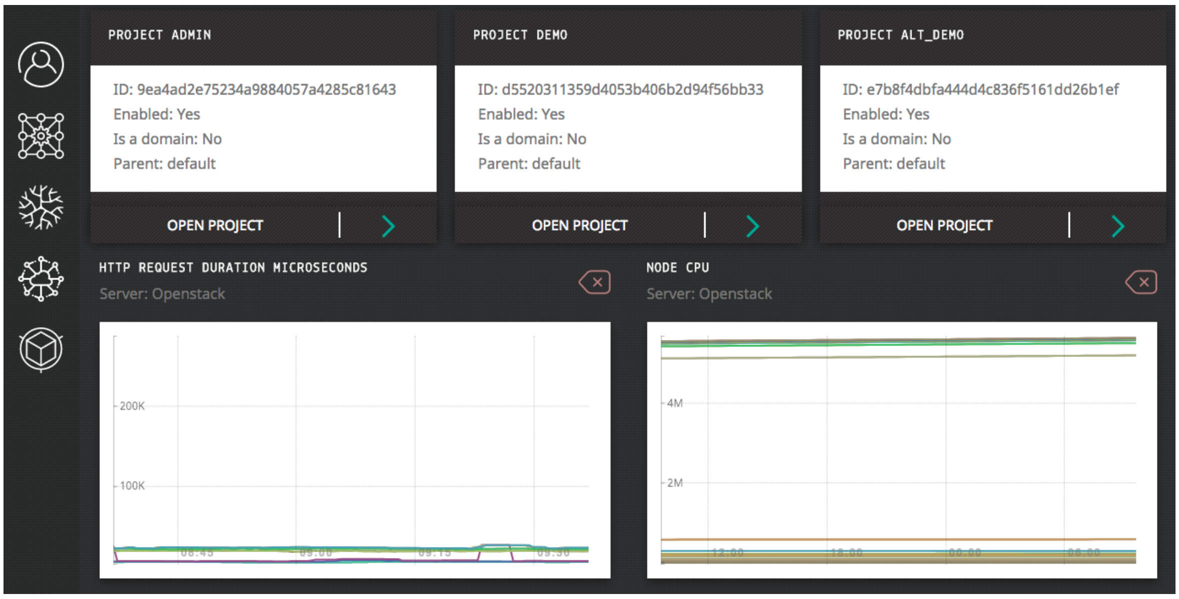


Figure 39: N2Sky Administration Dashboard. Featured Metrics component.

The user decides by himself which monitoring charts will be shown. The configuration is located in "Dashboard Settings" administration tool element. More details about the creation of monitoring charts are located in subsection 5.6.

5.4. Openstack Dashboard

OpenStack Dashboard is the dashboard for managing OpenStack and its services. It also displays the monitoring of OpenStack and its instances. It looks similar to the Administration Dashboard, except that it is the only oriented OpenStack and there are no additional tools as shown in figure 40.



PROJECT ADMIN
ID: 9ea4ad2e75234a9884057a4285c81643
Enabled: Yes
Is a domain: No
Parent: default
OPEN PROJECT

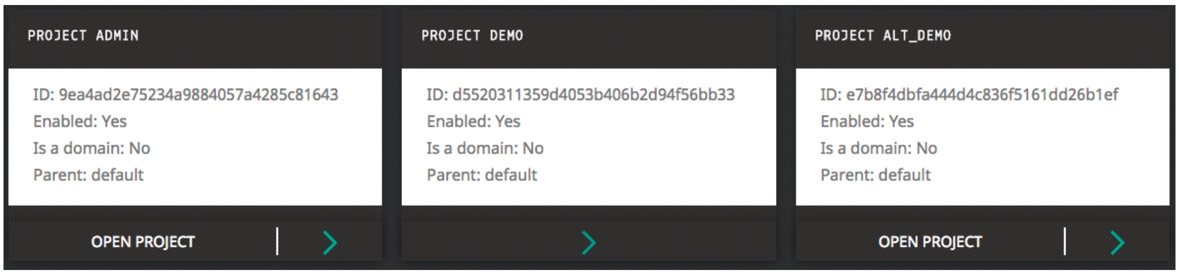
PROJECT DEMO
ID: d5520311359d4053b406b2d94f56bb33
Enabled: Yes
Is a domain: No
Parent: default
OPEN PROJECT

PROJECT ALT_DEMO
ID: e7b8f4dbfa444d4c836f5161dd26b1ef
Enabled: Yes
Is a domain: No
Parent: default
OPEN PROJECT

Figure 40: N2Sky OpenStack Dashboard.

This dashboard contains two blocks:

Project grid. This block contains projects, which are available on OpenStack as shown in figure 41.



PROJECT ADMIN
ID: 9ea4ad2e75234a9884057a4285c81643
Enabled: Yes
Is a domain: No
Parent: default
OPEN PROJECT

PROJECT DEMO
ID: d5520311359d4053b406b2d94f56bb33
Enabled: Yes
Is a domain: No
Parent: default

PROJECT ALT_DEMO
ID: e7b8f4dbfa444d4c836f5161dd26b1ef
Enabled: Yes
Is a domain: No
Parent: default
OPEN PROJECT

Figure 41: N2Sky OpenStack Dashboard. Projects grid

Every grid item represents a brief overview of the OpenStack Project:

- Name of the project, which is the header of the grid item.
- ID of the 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 shows if there any parent project which the current project is linked to.

- Button "Open Project", which redirects to the project details.

Monitoring Grid. Similar to the Administration Dashboard, the monitoring grid can be customized by users. Detailed information about customization is written in subsection 5.6.

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

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

Where <id> is the OpenStack project ID.

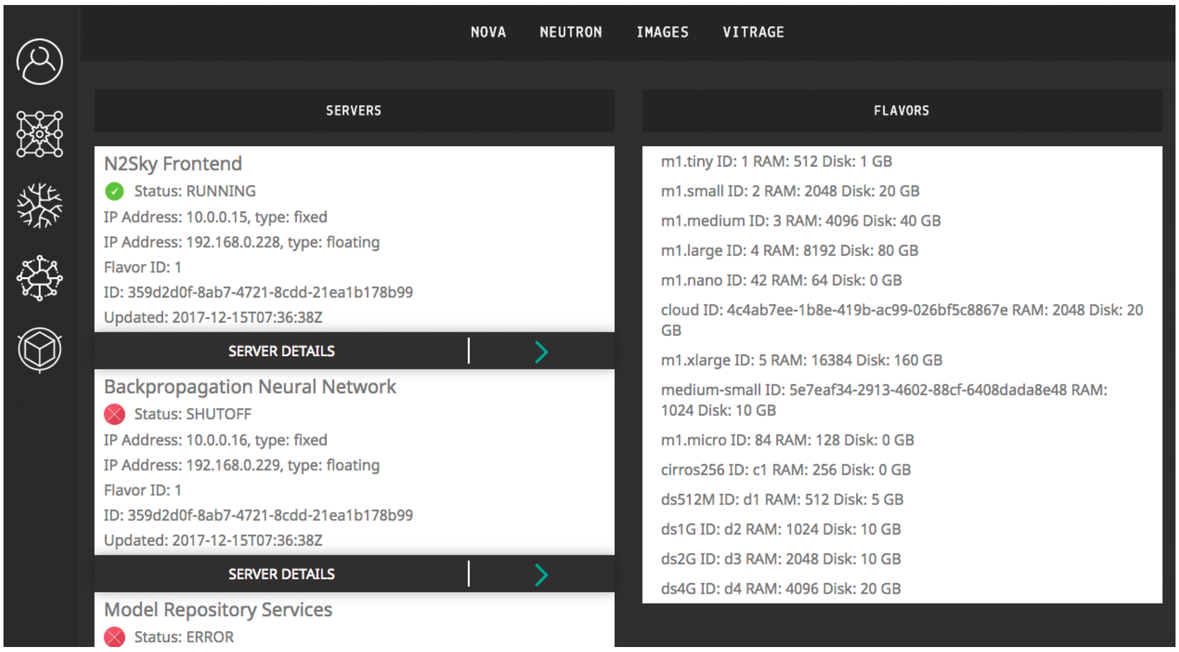
5.4.1. OpenStack Nova Service

Nova is a compute service of OpenStack. It gives an overview and manages all existing virtual machines (servers or instances) [18].

Following tools are used by this compute service:

- Horizon. Web UI for managing OpenStack projects. It is not used in the 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. N2Sky does not use it since OpenStack Client can provide all needed commands for nova service.

The OpenStack project view has detailed description about the server and its instances as shown in figure 42.



N2Sky Frontend
✓ Status: RUNNING
IP Address: 10.0.0.15, type: fixed
IP Address: 192.168.0.228, type: floating
Flavor ID: 1
ID: 359d2d0f-8ab7-4721-8cdd-21ea1b178b99
Updated: 2017-12-15T07:36:38Z

Backpropagation Neural Network
✗ Status: SHUTOFF
IP Address: 10.0.0.16, type: fixed
IP Address: 192.168.0.229, type: floating
Flavor ID: 1
ID: 359d2d0f-8ab7-4721-8cdd-21ea1b178b99
Updated: 2017-12-15T07:36:38Z

Model Repository Services
✗ Status: ERROR

m1.tiny ID: 1 RAM: 512 Disk: 1 GB
m1.small ID: 2 RAM: 2048 Disk: 20 GB
m1.medium ID: 3 RAM: 4096 Disk: 40 GB
m1.large ID: 4 RAM: 8192 Disk: 80 GB
m1.nano ID: 42 RAM: 64 Disk: 0 GB
cloud ID: 4c4ab7ee-1b8e-419b-ac99-026bf5c8867e RAM: 2048 Disk: 20 GB
m1.xlarge ID: 5 RAM: 16384 Disk: 160 GB
medium-small ID: 5e7eaf34-2913-4602-88cf-6408dada8e48 RAM: 1024 Disk: 10 GB
m1.micro ID: 84 RAM: 128 Disk: 0 GB
cirros256 ID: c1 RAM: 256 Disk: 0 GB
ds512M ID: d1 RAM: 512 Disk: 5 GB
ds1G ID: d2 RAM: 1024 Disk: 10 GB
ds2G ID: d3 RAM: 2048 Disk: 10 GB
ds4G ID: d4 RAM: 4096 Disk: 20 GB

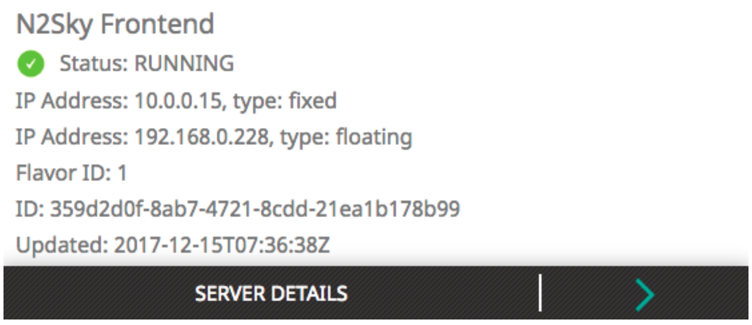
Figure 42: N2Sky OpenStack Dashboard. Project view

This view contains following components:

Navigation Bar. Navigation over OpenStack Services namely:

- NOVA
- NEUTRON
- IMAGES
- VITRAGE

Servers. The servers item grid is located on the left side of the screen and has a collection of servers (OpenStack instances) as shown in figure 43.



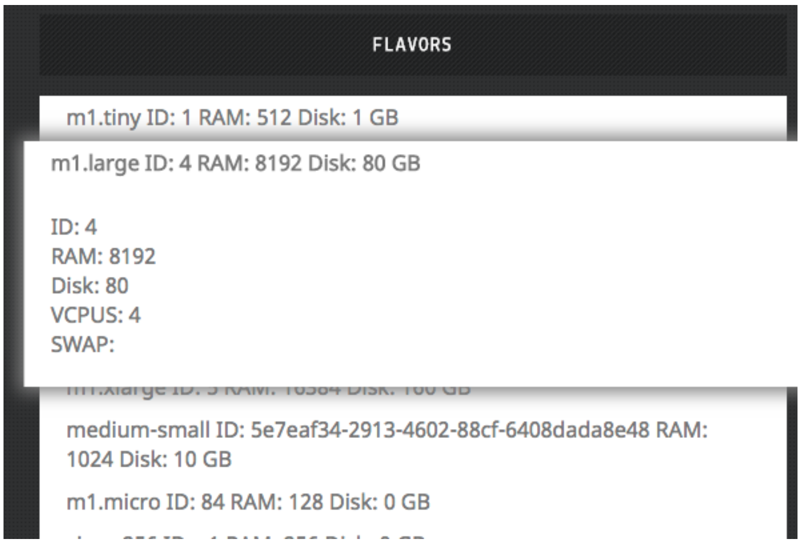
N2Sky Frontend
✓ Status: RUNNING
IP Address: 10.0.0.15, type: fixed
IP Address: 192.168.0.228, type: floating
Flavor ID: 1
ID: 359d2d0f-8ab7-4721-8cdd-21ea1b178b99
Updated: 2017-12-15T07:36:38Z

Figure 43: N2Sky OpenStack Dashboard. Server overview

The server overview grid contains following elements:

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

Flavors. A flavor is defined as an environment configuration. In the project details view, there is the list of flavors. On click on the particular flavor of this list, the flavor details will appear as shown in figure 44.


FLAVORS

m1.tiny ID: 1 RAM: 512 Disk: 1 GB

m1.large ID: 4 RAM: 8192 Disk: 80 GB

ID: 4
RAM: 8192
Disk: 80
VCPUS: 4
SWAP:

m1.xlarge ID: 5 RAM: 16384 Disk: 160 GB

medium-small ID: 5e7eaf34-2913-4602-88cf-6408dada8e48 RAM: 1024 Disk: 10 GB

m1.micro ID: 84 RAM: 128 Disk: 0 GB

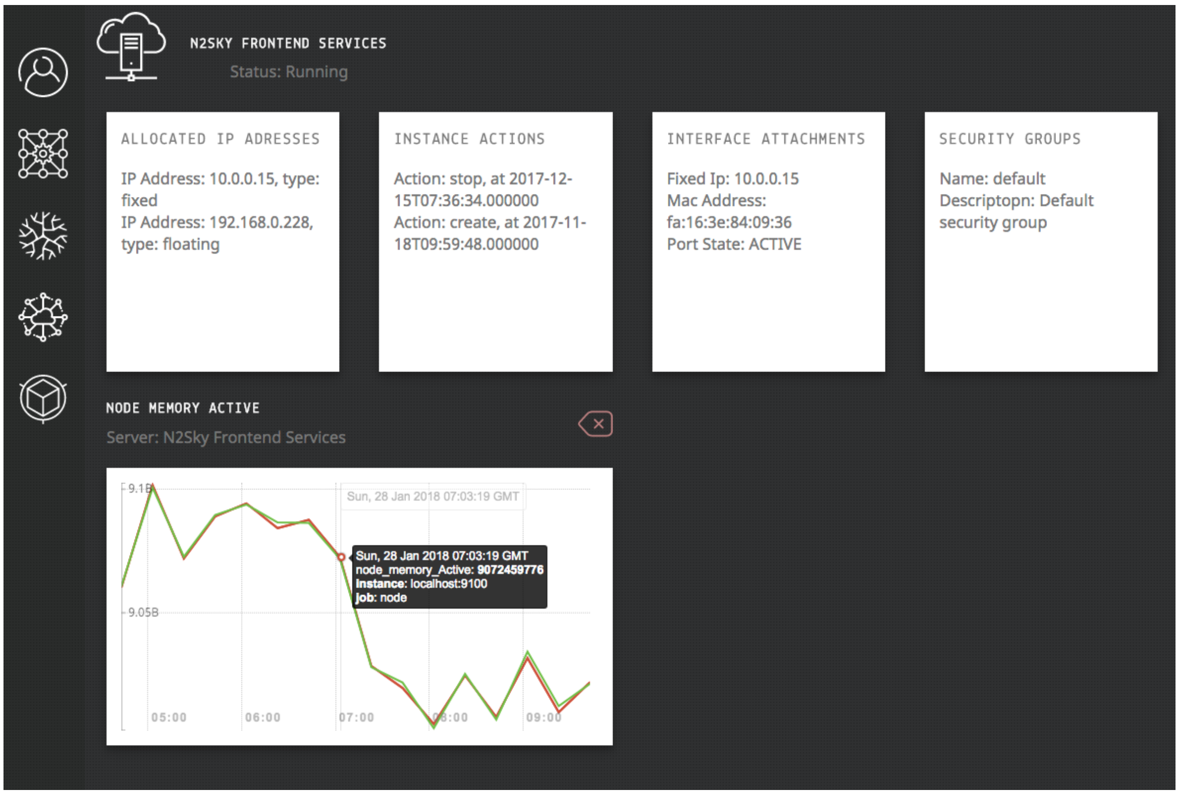
Figure 44: N2Sky OpenStack Dashboard. Flavors

Following elements are displayed:

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

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

When the user chooses some server (instance), he will be redirected to the server details view as shown in figure 45.


N2SKY FRONTEND SERVICES

Status: Running

ALLOCATED IP ADRESSES

IP Address: 10.0.0.15, type: fixed
IP Address: 192.168.0.228, type: floating

INSTANCE ACTIONS

Action: stop, at 2017-12-15T07:36:34.000000
Action: create, at 2017-11-18T09:59:48.000000

INTERFACE ATTACHMENTS

Fixed Ip: 10.0.0.15
Mac Address: fa:16:3e:84:09:36
Port State: ACTIVE

SECURITY GROUPS

Name: default
Descriptopn: Default security group

NODE MEMORY ACTIVE

Server: N2Sky Frontend Services

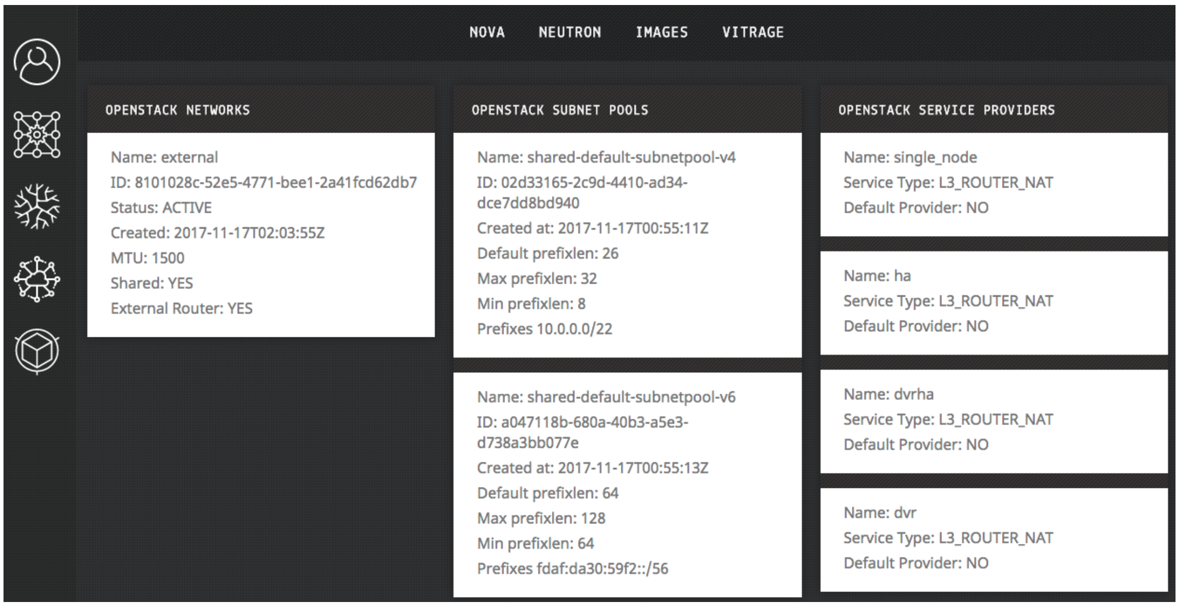
Sun, 28 Jan 2018 07:03:19 GMT
node_memory_Active: 9072459776
Instance: localhost:9100
Job: node

Figure 45: N2Sky OpenStack Dashboard. Server details view.

5.4.2. OpenStack Neutron Service

Neutron is an OpenStack networking service, which provides "network connectivity as a service". Neutron is fully integrated into the OpenStack UI, which allows managing networking directly from there. The networking service is based on quantum architecture, where API clients communicate with virtual switches through Quantum API and Quantum plugin [7]. This service implements Neutron API, which is used by the N2Sky Web UI.

Neutron Service is integrated into N2Sky and represents the overview of networks, subnet pools, and services providers as shown in figure 46.



OPENSTACK NETWORKS

Name: external
ID: 8101028c-52e5-4771-bee1-2a41fcd62db7
Status: ACTIVE
Created: 2017-11-17T02:03:55Z
MTU: 1500
Shared: YES
External Router: YES

OPENSTACK SUBNET POOLS

Name: shared-default-subnetpool-v4
ID: 02d33165-2c9d-4410-ad34-dce7dd8bd940
Created at: 2017-11-17T00:55:11Z
Default prefixlen: 26
Max prefixlen: 32
Min prefixlen: 8
Prefixes 10.0.0.0/22

Name: shared-default-subnetpool-v6
ID: a047118b-680a-40b3-a5e3-d738a3bb077e
Created at: 2017-11-17T00:55:13Z
Default prefixlen: 64
Max prefixlen: 128
Min prefixlen: 64
Prefixes fdaf:da30:59f2::/56

OPENSTACK SERVICE PROVIDERS

Name: single_node
Service Type: L3_ROUTER_NAT
Default Provider: NO

Name: ha
Service Type: L3_ROUTER_NAT
Default Provider: NO

Name: dvrha
Service Type: L3_ROUTER_NAT
Default Provider: NO

Name: dvr
Service Type: L3_ROUTER_NAT
Default Provider: NO

Figure 46: N2Sky OpenStack Dashboard. Neutron Service View

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 the instance is available and running on OpenStack environment, "Shutdown" if the instance is available but not running or an "Error" in case of an error occurring.

Instance information grid . The following grid represents summary information about the server (instance) and consist of these items:

- Allocated IP Address 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 activities and events, which happened in the particular instance. Log information contains the action type and timestamp of action.
- Interface attachments, which contains fixed IP address, mac address add port state ("ACTIVE" or "INACTIVE") of the instance.
- Security groups is a list of security groups which contains the name and description of the security group.

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

The grid represents following components besides the navigation bar:

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

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

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

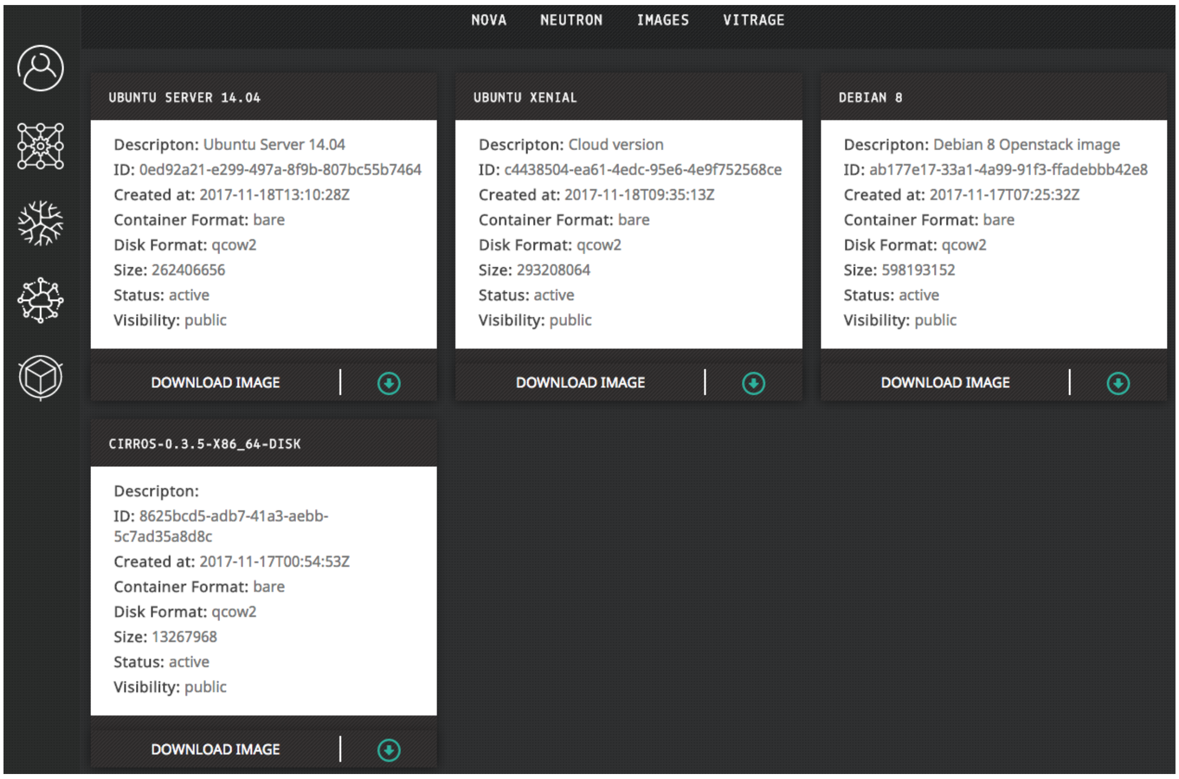
1. Name of the subnetwork
2. ID of the subnetwork
3. Created at is the timestamp of subnetwork creation
4. Default number prefixlen
5. Max number of prefixlen
6. Min number of prefixlen
7. Prefixlen, which represents a pool prefix

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

1. Name of service provider
2. Service type
3. Default Provider. "YES" if it exists, "NO" if not.

5.4.3. OpenStack Images Service

In OpenStack is possible to upload customized images of the Virtual Machine to the provider. N2Sky uses the Images Service only for the representation of images information. The user can download an image, but it is not possible to upload customized image due to the expensive process, because of the image size. It is possible to use template provider with pre-configured virtual machine [35].



UBUNTU SERVER 14.04

Descripton: Ubuntu Server 14.04
ID: 0ed92a21-e299-497a-8f9b-807bc55b7464
Created at: 2017-11-18T13:10:28Z
Container Format: bare
Disk Format: qcow2
Size: 262406656
Status: active
Visibility: public

UBUNTU XENIAL

Descripton: Cloud version
ID: c4438504-ea61-4edc-95e6-4e9f752568ce
Created at: 2017-11-18T09:35:13Z
Container Format: bare
Disk Format: qcow2
Size: 293208064
Status: active
Visibility: public

DEBIAN 8

Descripton: Debian 8 Openstack image
ID: ab177e17-33a1-4a99-91f3-ffadebbb42e8
Created at: 2017-11-17T07:25:32Z
Container Format: bare
Disk Format: qcow2
Size: 598193152
Status: active
Visibility: public

CIRROS-0.3.5-X86_64-DISK

Descripton:
ID: 8625bcd5-adb7-41a3-aebb-5c7ad35a8d8c
Created at: 2017-11-17T00:54:53Z
Container Format: bare
Disk Format: qcow2
Size: 13267968
Status: active
Visibility: public

Figure 47: N2Sky OpenStack Dashboard. Images Service View

The Image Service view, shown in figure 47 is representing a grid of available images. Every grid item contains the 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 uploaded and ready to use, inactive if not
9. Visibility. Public for everyone, or group name for a particular group of OpenStack users

10. Download image button, which creates one more thread and initializes the downloading procedure

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

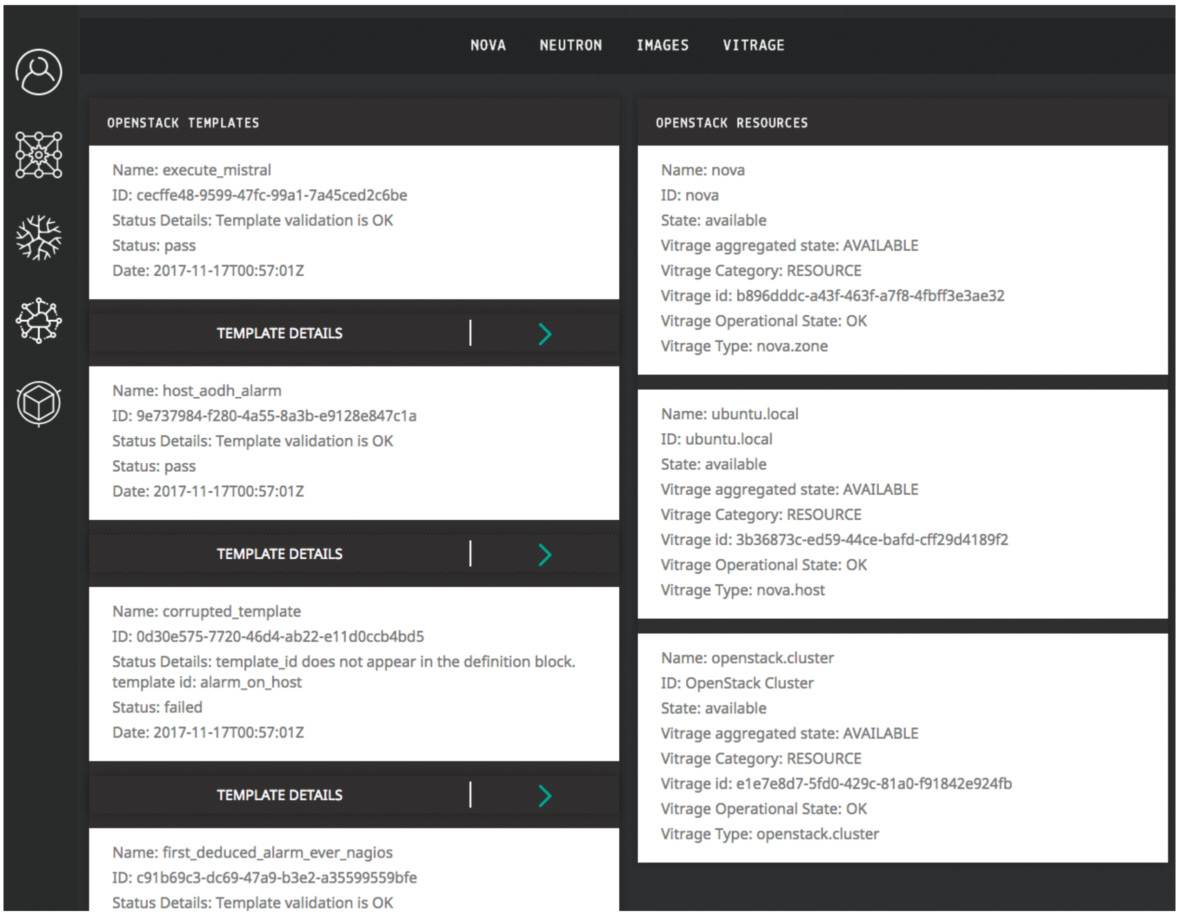
- Preinstalled Docker CLI
- N2Sky Monitoring System instance
- N2Sky Alert Management System rules configuration

N2Sky templates and configuration have to support the 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. It is build in the monitoring system, which helps to analyze and organize the OpenStack alarms and events [15]. In the N2Sky platform, the Vitrage service is used only for representation of templates and resources because N2Sky has its own Monitoring System, which can be propagated in all OpenStack instances.



OPENSTACK TEMPLATES

Name: execute_mistral
ID: cecffe48-9599-47fc-99a1-7a45ced2c6be
Status Details: Template validation is OK
Status: pass
Date: 2017-11-17T00:57:01Z

Name: host_aodh_alarm
ID: 9e737984-f280-4a55-8a3b-e9128e847c1a
Status Details: Template validation is OK
Status: pass
Date: 2017-11-17T00:57:01Z

Name: 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

Name: first_deduced_alarm_ever_nagios
ID: c91b69c3-dc69-47a9-b3e2-a35599559bfe
Status Details: Template validation is OK

OPENSTACK RESOURCES

Name: 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

Name: 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

Name: 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

The two column grid view shown in figure 48, represents the following grid items:

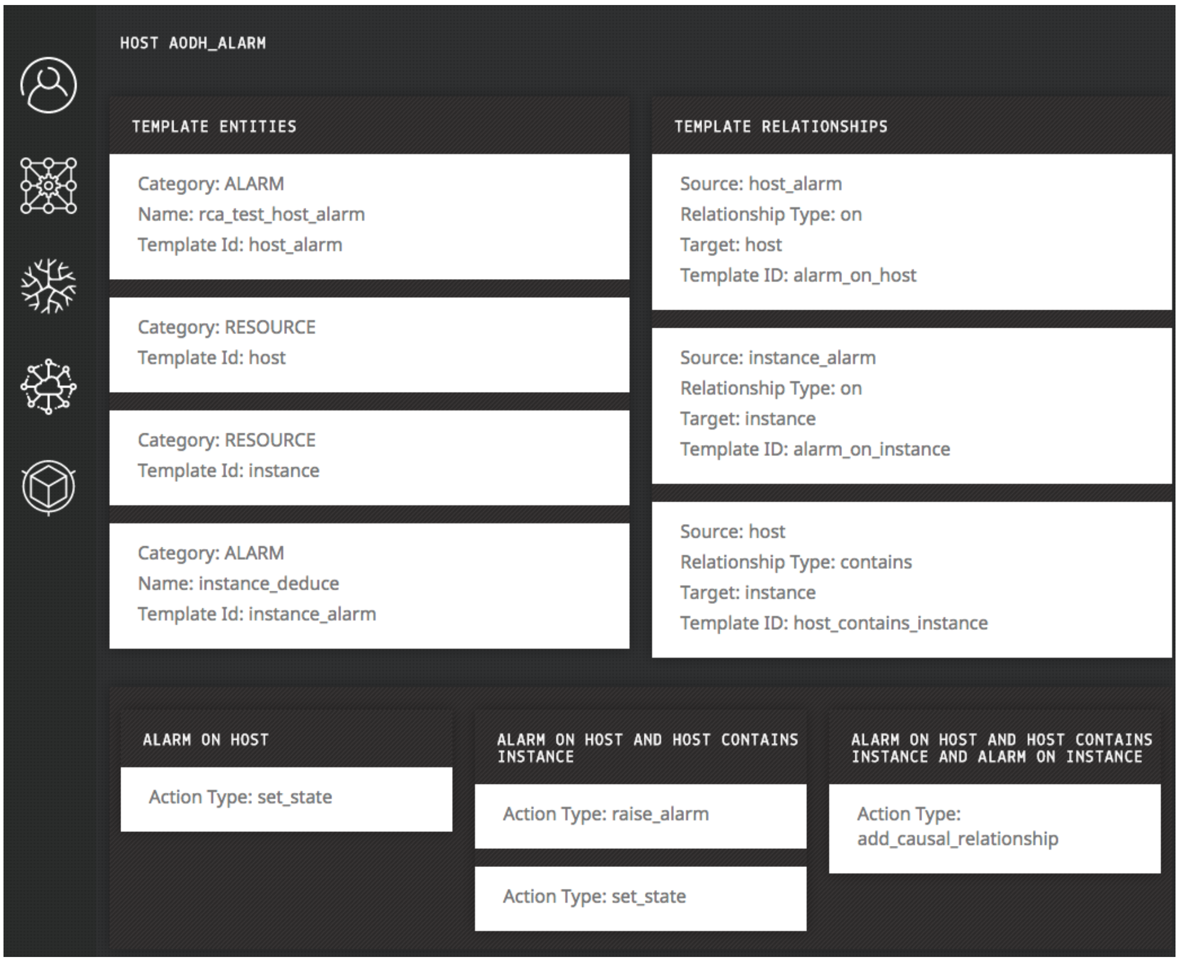
OpenStack Templates. List of OpenStack templates with following data:

1. Name of the template
2. ID of the template
3. Status Details, which shows if the template is validated
4. Date, which shows the timestamp of template creation
5. Template details button, which redirects in detailed information about the 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

4. Vitrage aggregated state can be available or not
5. Vitrage Category can be resource, alarm or another customized category
6. Vitrage ID
7. Vitrage Operational State
8. Vitrage Type


HOST AODH_ALARM

TEMPLATE ENTITIES

Category: ALARM
Name: rca_test_host_alarm
Template Id: host_alarm

Category: RESOURCE
Template Id: host

Category: RESOURCE
Template Id: instance

Category: ALARM
Name: instance_deduce
Template Id: instance_alarm

TEMPLATE RELATIONSHIPS

Source: host_alarm
Relationship Type: on
Target: host
Template ID: alarm_on_host

Source: instance_alarm
Relationship Type: on
Target: instance
Template ID: alarm_on_instance

Source: host
Relationship Type: contains
Target: instance
Template ID: host_contains_instance

ALARM ON HOST

Action Type: set_state

ALARM ON HOST AND HOST CONTAINS INSTANCE

Action Type: raise_alarm

Action Type: set_state

ALARM ON HOST AND HOST CONTAINS INSTANCE AND ALARM ON INSTANCE

Action Type: add_causal_relationship

Figure 49: N2Sky OpenStack Dashboard. Template Details View

When the user clicks on the template details button, he will be redirected to a particular Vitrage template as shown in figure 49. From here, the user can observe the following information:

Template Entities . List of entities like alarm or resource. This grid item contains the 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 if host contains instance.

Alarm on host and host contains instance an alarm on the instance. Optional item, visible only if host contains instance and alarm on the instance.

5.5. Dashboard Settings

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

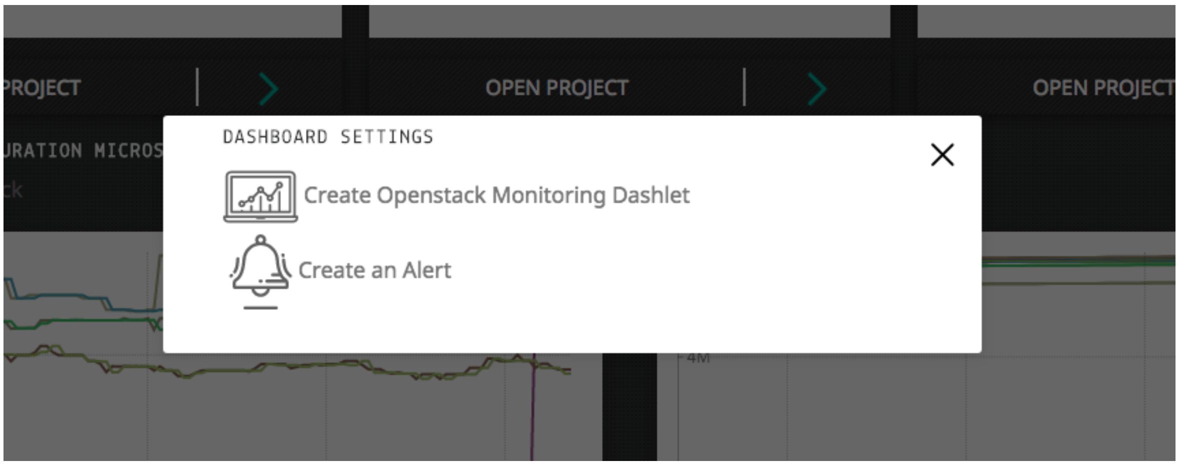


Figure 50: N2Sky Administration Dashboard. Dashboard Settings

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

The Dashboard Settings popup contains two settings:

- Create OpenStack Monitoring Dashlet, which creates one more modal popup upon the existing and initializes the creation of the monitoring chart, described in chapter "Monitoring System" in subsection 5.6.

- Create an Alert, which also creates another modal popup upon the existing and initializes the creation of the alerting rule, described in chapter "Alert System" in subsection 5.7.

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

5.6. Monitoring System

When Administration Dashboard shows the overview of the OpenStack and Cloudify, the monitoring system goes through every dashboard and applies monitoring charts on user request. There are two main parts, mentioned in FRS: displaying of monitoring charts and management of metrics.

5.6.1. Monitoring Charts Creation

After a user initializes the creation of monitoring metrics, described in subsection 5.5 "Dashboard Settings", a new modal window will be opened as shown in figure 51.

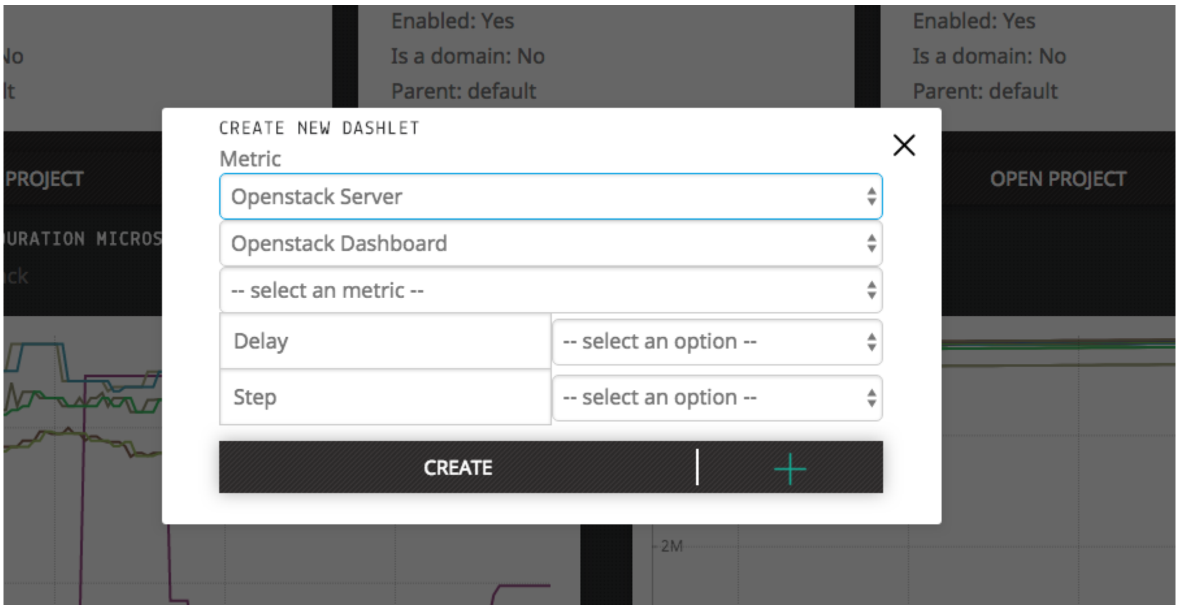


Figure 51: N2Sky Administration Dashboard. Monitoring charts creation modal popup

The popup window has following elements:

1. "CREATE NEW DASHLET" title of the popup

2. Combobox with servers, where the monitoring system instance is installed. The user can choose all available instances, even if they are not running currently. It is also possible to choose the OpenStack cloud server itself because the monitoring system instance is preinstalled there.
3. Combobox with views. The user can choose where the combo box will be added. There are few places where charts can be displayed:
   a) Administration Dashboard
   b) OpenStack Dashboard
   c) Cloudify Dashboard
   d) Servers (OpenStack instances) View.
4. Combobox with metrics. Normally, there are more than hundreds of available metrics. Every operating system has different naming for 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 combo box. This combo box contains following time options:
   a) seconds
   b) minutes
   c) hours
   d) days
   e) weeks
8. Create button, which triggers metrics creation. The view will be reloaded and monitoring chart will appear in this view.

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

5.6.2. Monitoring Charts Representation

Every monitoring chart is a grid item of two column responsive grid as shown in figure 52

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

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

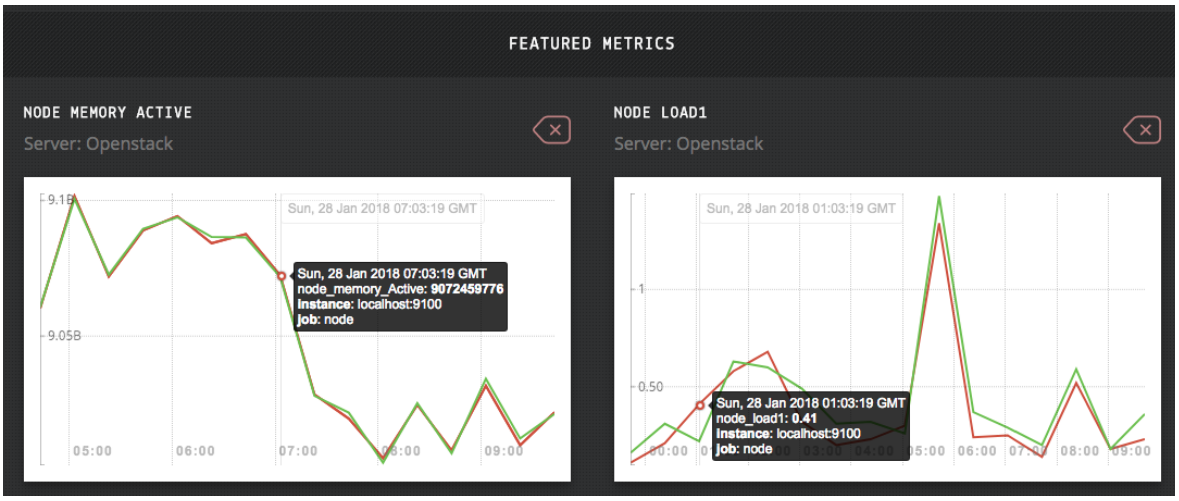


Figure 52: N2Sky Administration Dashboard. Monitoring charts representation

Every monitoring chart has at least one line graph with a random color. If there are a few lines in the graphs chart, then lines should have different colors. Each chart contains on X-axis, the timestamp and on y-Axis, the velocity. The timestamp, in the line graph, has its own toolkit with additional information about the chart and contains the 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

5.7. Alert System

The Alert System has its own page view. It is possible to be redirected to this view from the main navigation menu as well as from the administration dashboard. The URL of the alert page is:

```
<host>/alert
```

5.7.1. Alerting rule Creation

Alert rules can be created directly from the alert page view. The button "Create Alert" initiates the modal popup window as shown in figure 53.

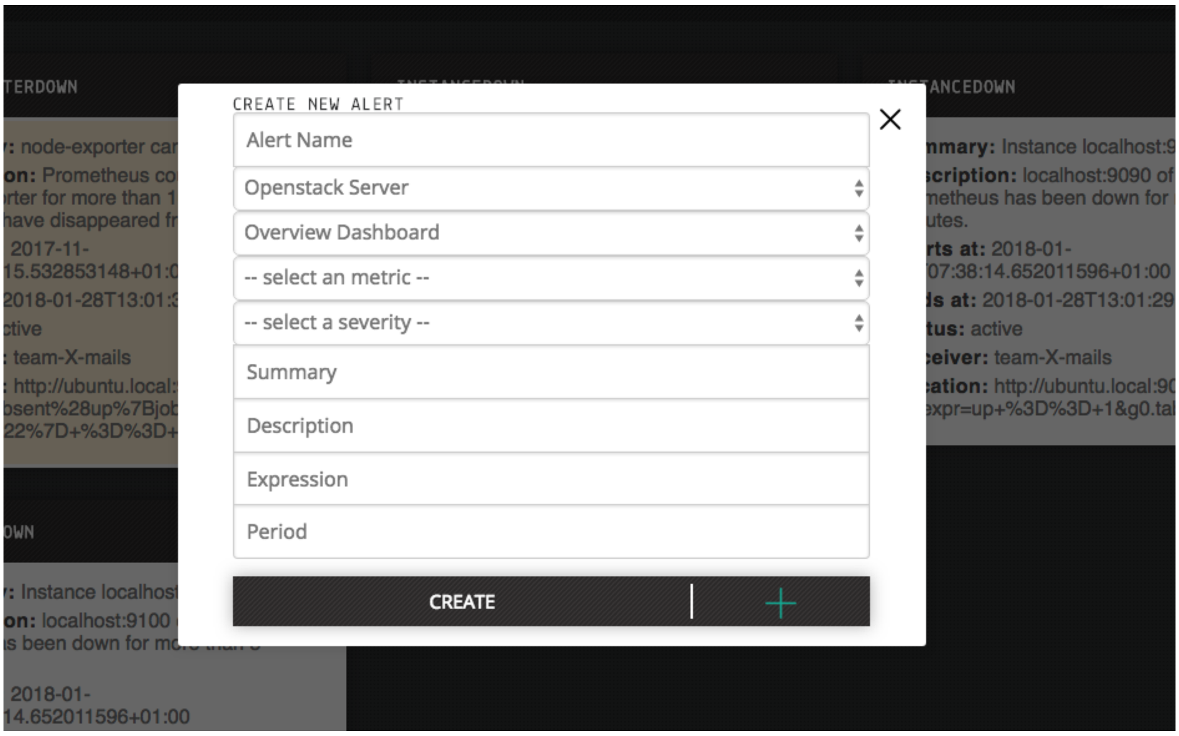


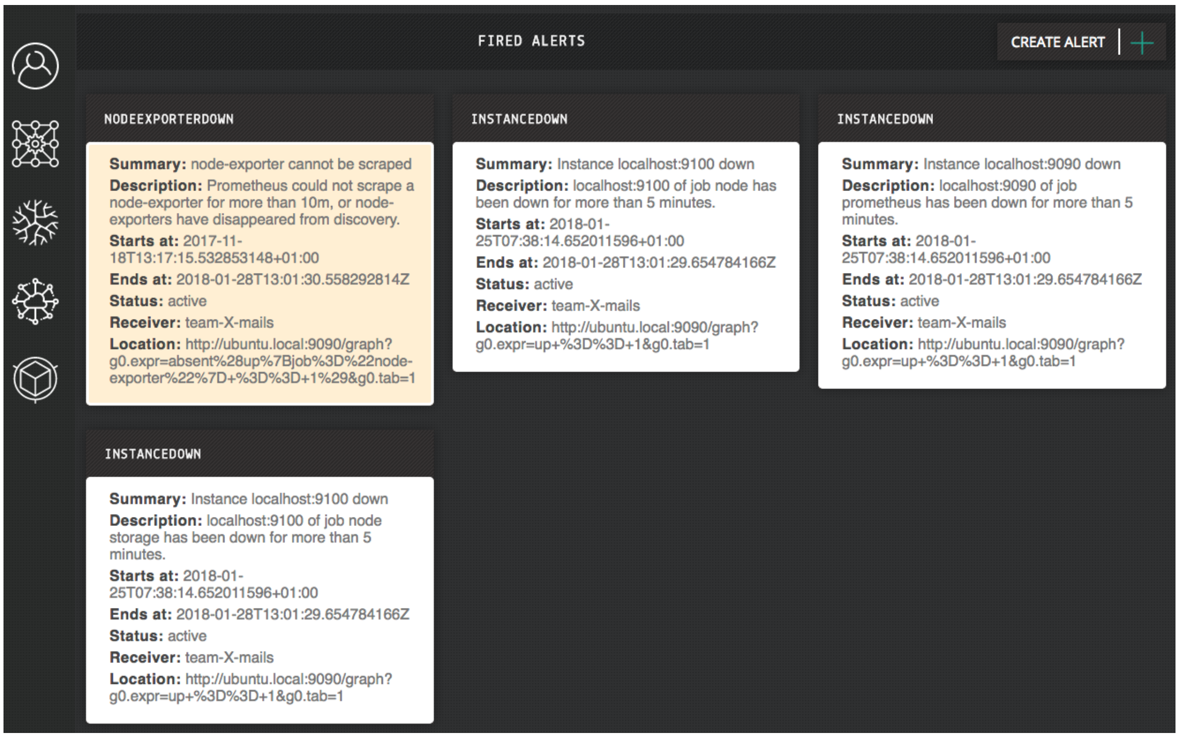
Figure 53: N2Sky Administration Dashboard. Alerting rule creation

The modal window has following elements:

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

5.7.2. Alerts Representation

The alerts are displayed in a three column grid as shown in figure 54



NODEEXPORTERDOWN

Summary: node-exporter cannot be scraped
Description: Prometheus could not scrape a node-exporter for more than 10m, or node-exporters have disappeared from discovery.
Starts at: 2017-11-18T13:17:15.532853148+01:00
Ends at: 2018-01-28T13:01:30.558292814Z
Status: active
Receiver: team-X-mails
Location: http://ubuntu.local:9090/graph?g0.expr=absent%28up%7Bjob%3D%22node-exporter%22%7D+%3D%3D+1%29&g0.tab=1

INSTANCEDOWN

Summary: Instance localhost:9100 down
Description: localhost:9100 of job node has been down for more than 5 minutes.
Starts at: 2018-01-25T07:38:14.652011596+01:00
Ends at: 2018-01-28T13:01:29.654784166Z
Status: active
Receiver: team-X-mails
Location: http://ubuntu.local:9090/graph?g0.expr=up+%3D%3D+1&g0.tab=1

INSTANCEDOWN

Summary: Instance localhost:9090 down
Description: localhost:9090 of job prometheus has been down for more than 5 minutes.
Starts at: 2018-01-25T07:38:14.652011596+01:00
Ends at: 2018-01-28T13:01:29.654784166Z
Status: active
Receiver: team-X-mails
Location: http://ubuntu.local:9090/graph?g0.expr=up+%3D%3D+1&g0.tab=1

INSTANCEDOWN

Summary: Instance localhost:9100 down
Description: localhost:9100 of job node storage has been down for more than 5 minutes.
Starts at: 2018-01-25T07:38:14.652011596+01:00
Ends at: 2018-01-28T13:01:29.654784166Z
Status: active
Receiver: team-X-mails
Location: http://ubuntu.local:9090/graph?g0.expr=up+%3D%3D+1&g0.tab=1

Figure 54: N2Sky Administration Dashboard. Alerts representation

The view contains:

Header. Header is the navigation bar with the title "Fired alerts" and the button. The button has the caption "Create Alert" and on click will initiate creating alert rule popup described in subsubsection 5.7.1 "Alerting rule creation".

Alerts Grid Contains three columns of alerts. When the alert end date is earlier than the current date, then this 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

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

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

MAM is a core component of the N2Sky platform. From this component, users can feel the power of the N2Sky System and try out its whole functionality. The MAM, as well as other N2Sky components, can be used on any device because they support responsive design.

6.2. Affected Users

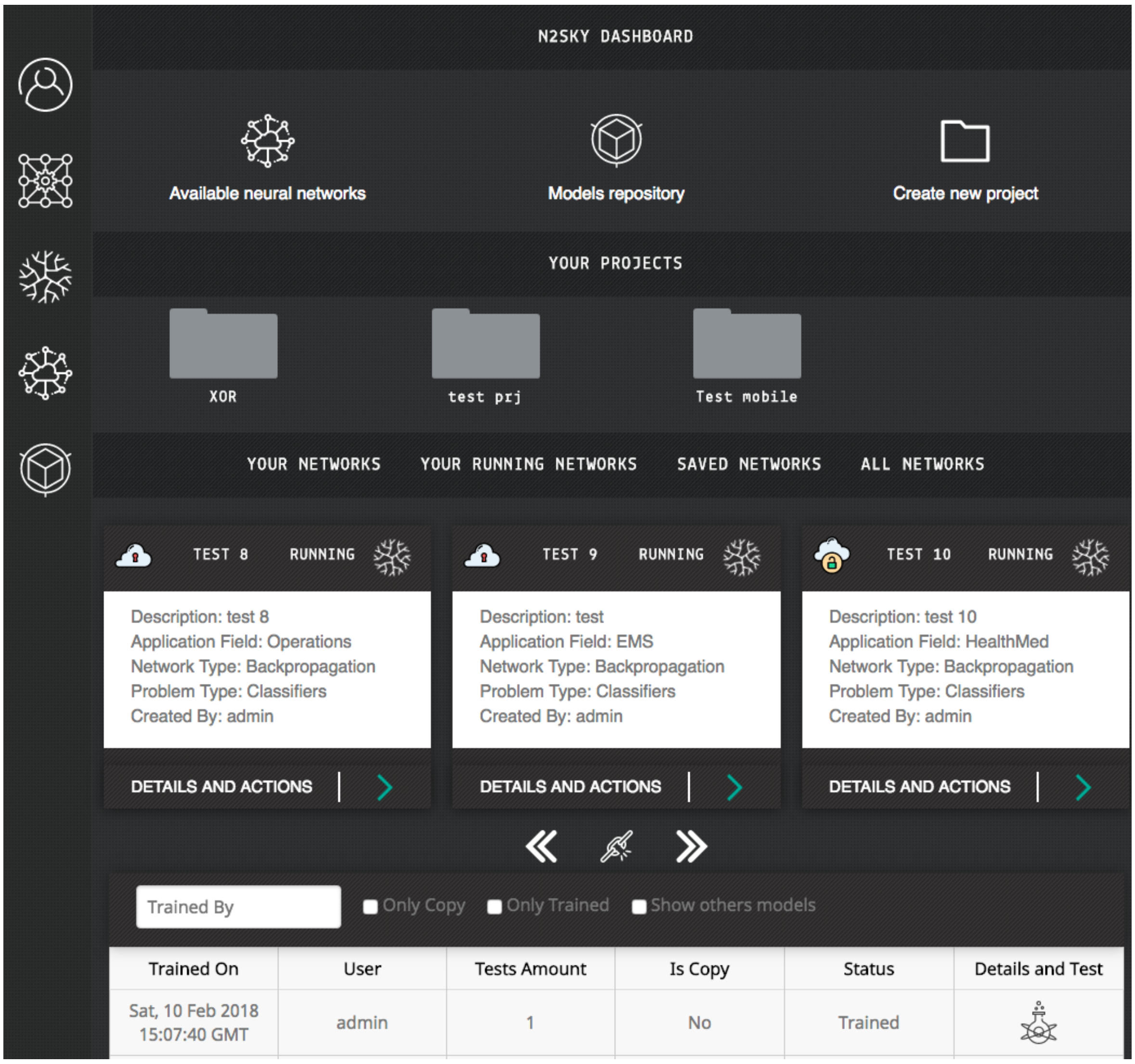
The MAM has one User Interface but users can use it diversely. Only users, which their main function is within MAM, can operate this module on their own purpose. Following types of users have this kind of main function:

- *Arbitrary User.* The main function of the arbitrary user is to learn about neural networks or try out knowledge in this field. The typical use case is when the user logs in from the mobile device, like a smartphone or tablet, and search something in the repositories. This user copies existing neural networks or trained models into his own project and performs operations on them. A detailed description is found in subsubsection 2.2.4.
- *Neural Network Engineer User.* This kind of user normally uses the Desktop version of N2Sky, but he can also use the mobile version because all functionalities are also available there. The user creates his own neural network from existing paradigms, but he also can act as an Arbitrary User. A detailed description is found in subsubsection 2.2.4.
- *Contributor User.* The Contributor uses the N2Sky MAM component mostly for an easier way to check his own neural network paradigm. Since this user can fully use N2Sky open API, UI for him is just for a quick check. The UI is available also in the mobile version, that is why a Contributor can observe his neural networks behavior directly from the mobile device in any place. A detailed description is found in subsubsection 2.2.4.

- *System Administrator.* His main function is on the AM module, but since he is also the administrator of the N2Sky MAM, he can see all processes of other users. This user can shadow any other users in order to observe what is happening in particular user dashboards. A detailed description is found in subsubsection 2.2.4.

6.3. N2Sky Dashboard

The N2Sky dashboard is the central component of the MAM module. The dashboard gives brief information about available tools, user projects and also user neural networks. Every dashboard is unique for every user, in other words, the content always differs. As displayed in figure 55, the dashboard gives a full overview of the components from one page.



Description: test 8
Application Field: Operations
Network Type: Backpropagation
Problem Type: Classifiers
Created By: admin

Description: test
Application Field: EMS
Network Type: Backpropagation
Problem Type: Classifiers
Created By: admin

Description: test 10
Application Field: HealthMed
Network Type: Backpropagation
Problem Type: Classifiers
Created By: admin

Trained On	User	Tests Amount	Is Copy	Status	Details and Test
Sat, 10 Feb 2018 15:07:40 GMT	admin	1	No	Trained	

Figure 55: N2Sky User Dashboard

The N2Sky dashboard contains following components:

- Available tools
- The user projects
- The user neural networks
- The user trained models

The dashboard is available under the following URL path:

```
<host>/cloud
```

6.3.1. Available Tools

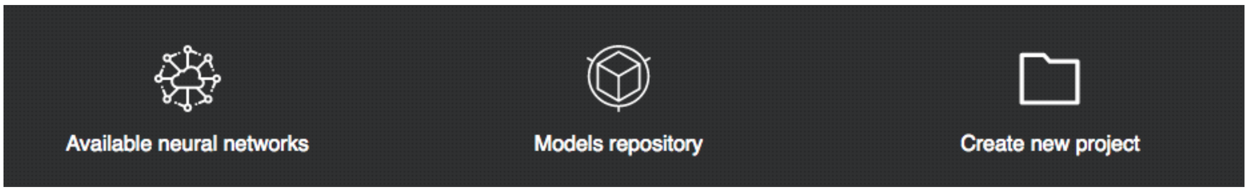


Figure 56: N2Sky Main Module. Available user tools

In figure 56 the available tools component is displayed in a horizontal layout. Every item has an SVG icon and caption under it. The tools contain following functionalities:

- *Available neural network.* Reference to the available repositories view.
- *Model repository.* Reference to the model repository view.
- *Create a new project.* On click, the popup window will be initialized as displayed in figure 57.

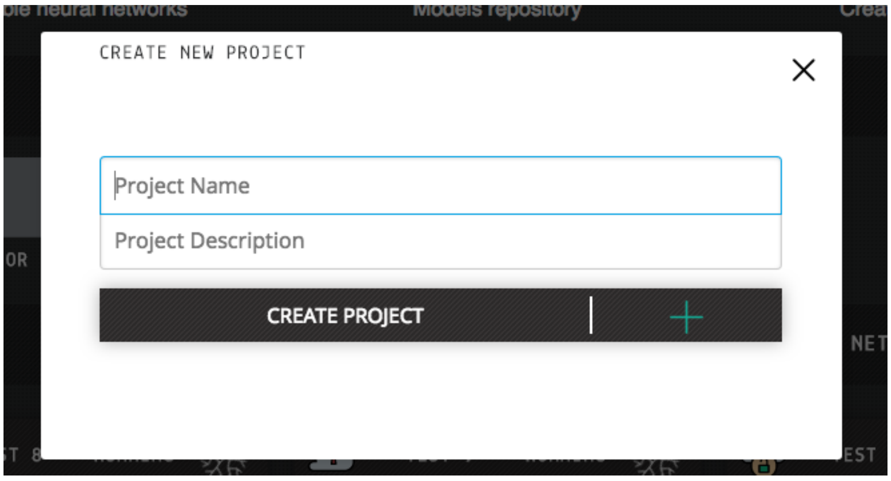


Figure 57: N2Sky Main Module. Create New Project Popup

The following popup modal window contains components:

– *Project Name* field, which represents a short title of the project.
– *Project Description* field with a short description of the project, which will be used for the semantic search.
– *Create Project* button, which creates a project with a filled up form.

After creating the project, it will automatically appear in the dashboard.

6.3.2. User Projects

User projects is a grid with projects. Every project is represented as a folder with a caption under it as displayed in figure 58. On click, the user will be redirected to a particular project.

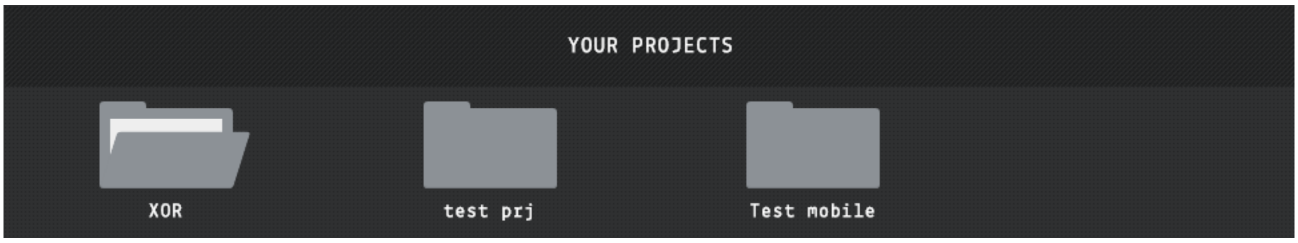


Figure 58: N2Sky Main Module. User projects grid

6.3.3. The User Neural Networks and Models Overview

The component displays the neural networks of the user, which are either created from paradigm or from his own contributed neural networks. The component contains following subcomponents:

- *Neural networks filtering bar.* The filtering bar is a navigation as well as a neural networks filtering subcomponent as shown in figure 59. The subcomponent allows the user to filter through neural networks depending on permissions.

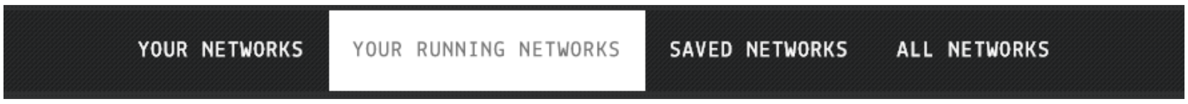


Figure 59: N2Sky Main Module. Neural networks filtering bar.

The following filters are available:

– *Your networks.* The filter shows the running and not running neural networks of the logged in user.

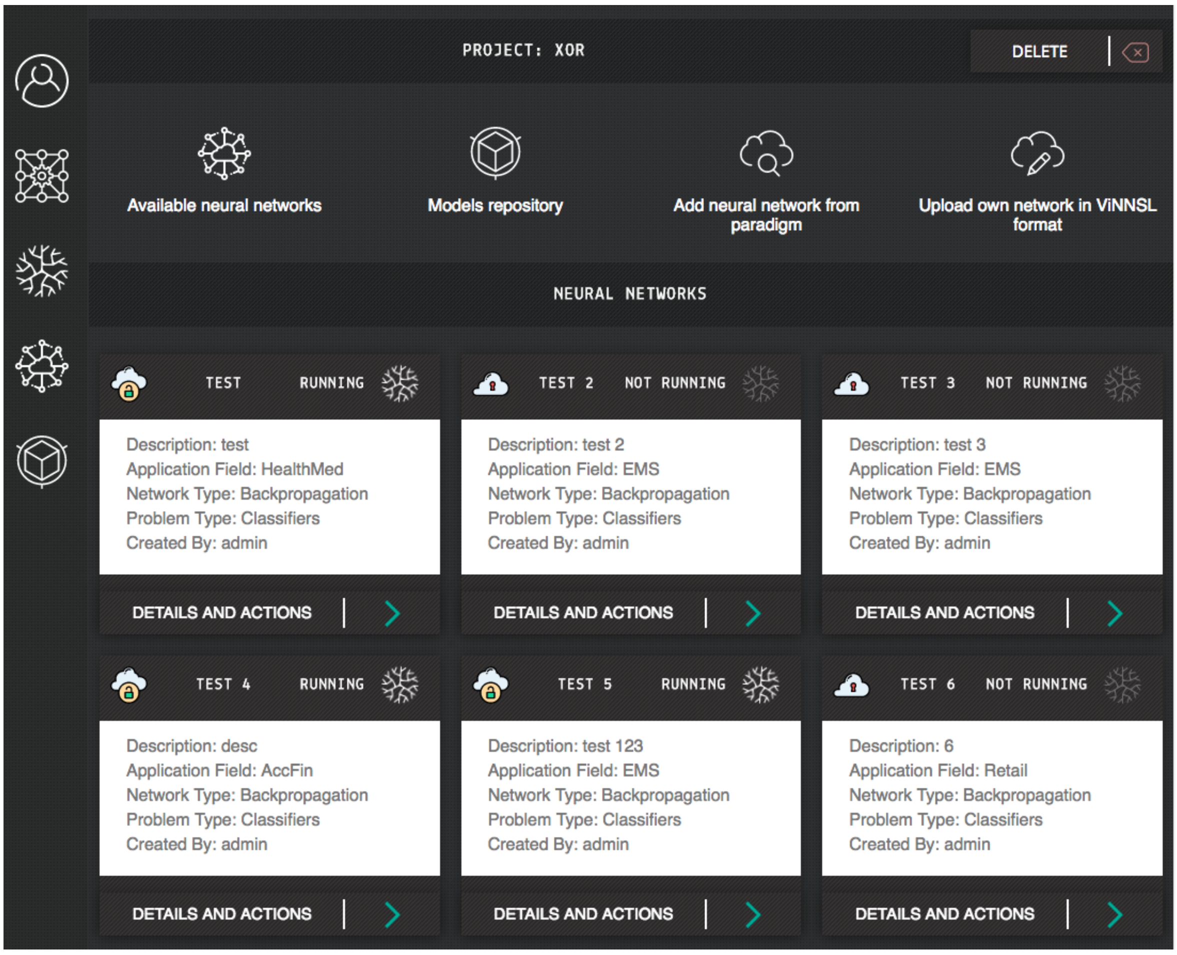
  - *Your running networks.* The representation only of the users running neural networks.
  - *Saved networks.* Only saved neural networks of the user. The list will show the saved neural networks across all projects of the user.
  - *All Networks.* List of all networks on the N2Sky. This filter is available only for the system administrator.

- The neural networks grid. The grid, which contains custom UI components of that particular neural network. Every grid item contains the following information:
  - The title of the neural network
  - The status either running or not running. If the instance is running, the N2Sky icon will spin, if not it will be grey without movement.
  - The short description of the neural network
  - Application field
  - Network type
  - Problem type
  - Created by user
  - Details and actions button, which redirect the user to the page with detailed information of the neural network.
- Navigation bar. The bar under neural networks which contains the navigation buttons and "Chained/Unchained" mode.
  - *Unchained mode* is a mode where all trained models of the displayed neural networks will be shown.
  - *Chained mode* is a mode where only the trained models of the chosen neural network will be shown. In the chained mode, the user has to click on the particular neural network to see the trained models.
- Trained models table. The table with trained models, which contains also a filtering and searching bar. The filtering and searching bar allows to perform a semantic search across trained models and contains following filters:
  - Trained by field available only for contributor user and system administrator.
  - Only Copy checkbox
  - Only Trained checkbox showing only the trained models where the training is finished.
  - Show others modes checkbox will display the trained models from other users. The checkbox is available only to the system administrator.

The table itself contains short information about the trained models. On click, the user will be redirected to the detailed overview of the trained model.

6.4. The User Project Dashboard

The users project dashboard is the dashboard, which is suitable for any type of user. It is possible to create and manage own neural networks directly from the dashboard as shown in figure 60.



PROJECT: XOR

DELETE

Available neural networks

Models repository

Add neural network from paradigm

Upload own network in ViNNSL format

NEURAL NETWORKS

TEST — RUNNING	TEST 2 — NOT RUNNING	TEST 3 — NOT RUNNING
Description: test Application Field: HealthMed Network Type: Backpropagation Problem Type: Classifiers Created By: admin	Description: test 2 Application Field: EMS Network Type: Backpropagation Problem Type: Classifiers Created By: admin	Description: test 3 Application Field: EMS Network Type: Backpropagation Problem Type: Classifiers Created By: admin
DETAILS AND ACTIONS	DETAILS AND ACTIONS	DETAILS AND ACTIONS
TEST 4 — RUNNING	TEST 5 — RUNNING	TEST 6 — NOT RUNNING
Description: desc Application Field: AccFin Network Type: Backpropagation Problem Type: Classifiers Created By: admin	Description: test 123 Application Field: EMS Network Type: Backpropagation Problem Type: Classifiers Created By: admin	Description: 6 Application Field: Retail Network Type: Backpropagation Problem Type: Classifiers Created By: admin
DETAILS AND ACTIONS	DETAILS AND ACTIONS	DETAILS AND ACTIONS

Figure 60: N2Sky User Project Dashboard

As any other dashboards in N2Sky, the project dashboard contains the available tools and the grid of items. The title and delete button are located on the top of the dashboard. Only the project owner or system administrator can delete a project.

6.4.1. Available Tools

The available tools are a common functionality, which are necessary for the project dashboard. Every tool contains an SVG icon and caption underneath as displayed in figure 61.

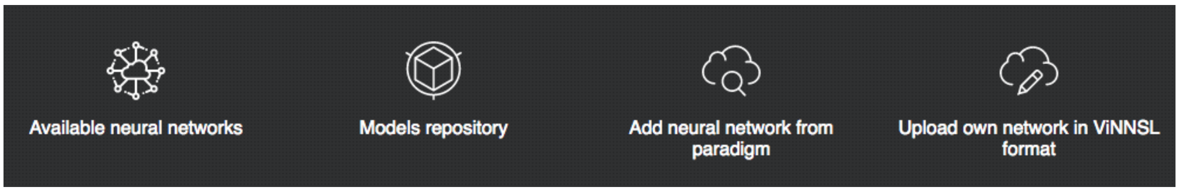
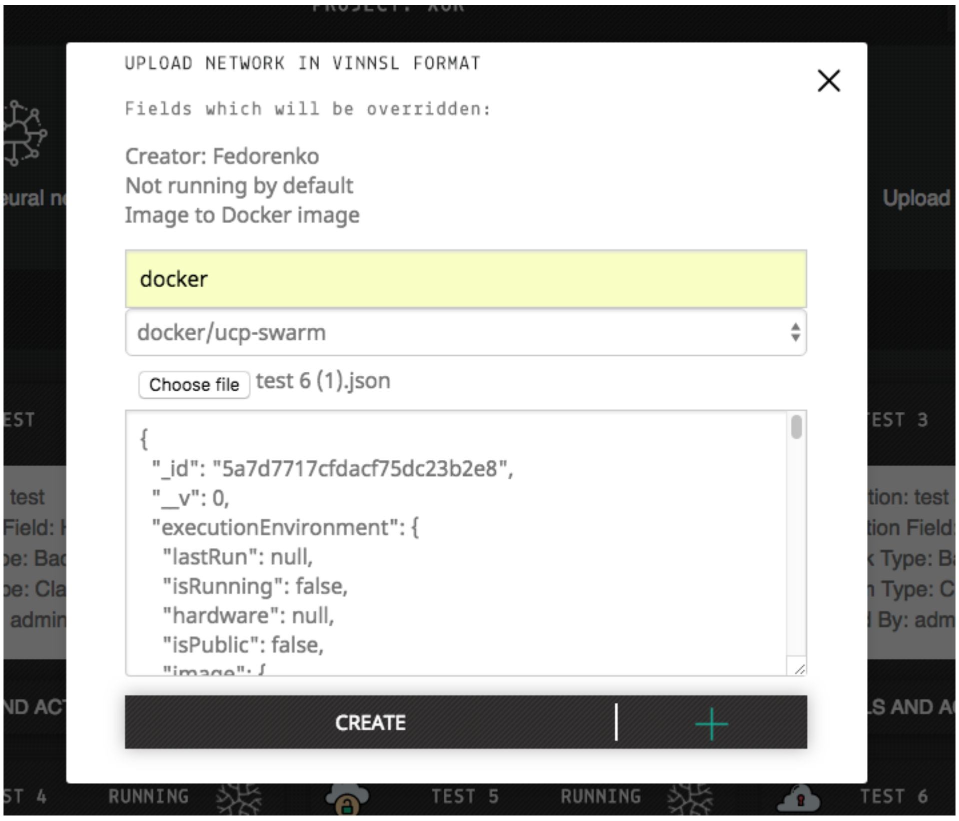

Available neural networks
Models repository
Add neural network from paradigm
Upload own network in ViNNSL format

Figure 61: N2Sky Project Dashboard. Available tools

The following tools are available for all types of users:

- *Available neural network.* Reference to the available repositories view.
- *Model repository.* Reference to the model repository view.
- *Add neural network from the paradigm.* Will redirect to the creation of the neural network from an existing paradigm page.
- *Upload own network in ViNNSL format* is mostly used by the contributor user since he needs to know the ViNNSL language. On click, the popup will be initialized as shown in figure 62.


UPLOAD NETWORK IN VINNSL FORMAT

Fields which will be overridden:

Creator: Fedorenko
Not running by default
Image to Docker image

docker

docker/ucp-swarm

Choose file test 6 (1).json

```
{
  "_id": "5a7d7717cfdacf75dc23b2e8",
  "__v": 0,
  "executionEnvironment": {
    "lastRun": null,
    "isRunning": false,
    "hardware": null,
    "isPublic": false,
```

Figure 62: N2Sky Project Dashboard.Upload own network in ViNNSL format

The popup contains following components:

- The information block about the following operation. The block shows, which fields are going to be overridden, namely: the creator field, image information and it will be not running by default.
- The Docker username field, which references on the username in Docker Hub repository. After the name is entered, the available images list will be listed.
- The Docker image combo box is a list of available images of a chosen Docker Hub user. This image will be used for creating neural network paradigms.
- The Upload field. The user needs to choose from a local machine the neural network paradigm, which is described in ViNNSL XML or JSON format.
- The text field will show the preview of uploaded neural network paradigm in ViNNSL format.
- Create button will process the form and create the neural network paradigm. After creation, the user will be redirected to the neural network details page.

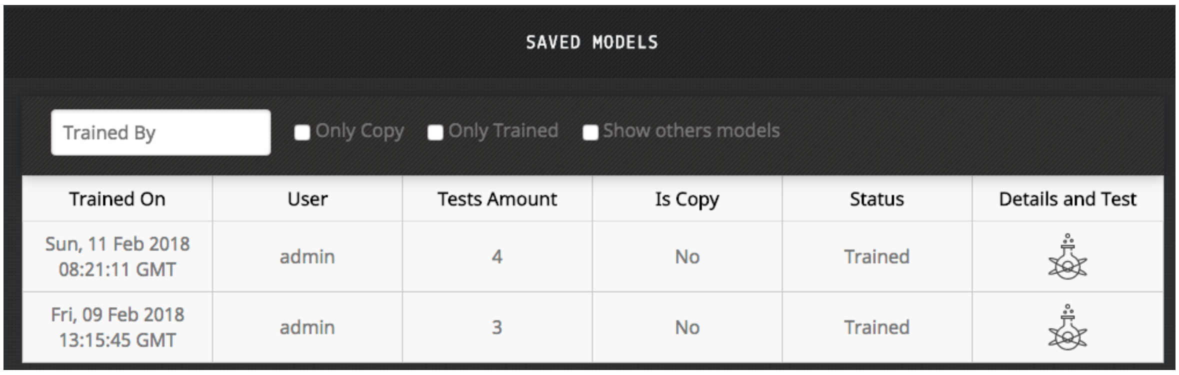
6.4.2. Neural Networks Grid

The neural networks grid represents a list of all neural networks which were created or copied from other users. Every grid item contains some brief information about the neural network:

- The title of the neural network
- The status either running or not running. If the instance is running the N2Sky icon will spin, if not it will be grey without movement.
- The short description of the neural network
- Application field
- Network type
- Problem type
- Created by user
- Details and actions button, which redirect the user to the page with detailed information of the neural network.

6.4.3. Saved Trained Models

The table with saved trained models is located at the bottom of project dashboard as shown in figure 63.


SAVED MODELS

Trained By ☐ Only Copy ☐ Only Trained ☐ Show others models

Trained On	User	Tests Amount	Is Copy	Status	Details and Test
Sun, 11 Feb 2018 08:21:11 GMT	admin	4	No	Trained	
Fri, 09 Feb 2018 13:15:45 GMT	admin	3	No	Trained	

Figure 63: N2Sky Project Dashboard. Saved trained models

This table represents the copied trained models into the particular project. Semantic search can be performed against trained models and this same approach is used in the N2Sky Main Dashboard.

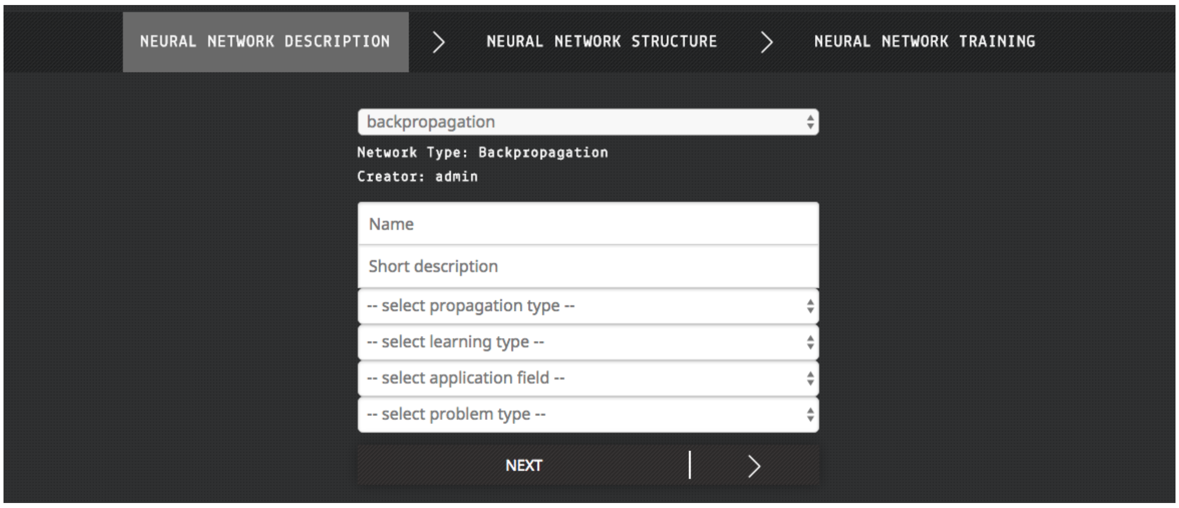
6.5. Three Steps View

Robert Browning in his poem "Andrea del Sarto" said: "Less is more", which became an inspiration of the "Three steps view". It was derived from the simplicity of the arbitrary user. The idea behind is to create the neural network from the paradigm within only three steps. After the creation, the same three steps are reused to provide the overview of this neural network.

The following steps have to be achieved in order to create the neural network from the paradigm: the neural network description, the neural network structure, and the neural network training.

6.5.1. The Neural Network Description

When the user enters the three step view, the first thing he notices is "The Network Description" tab. In this tab, the user has to choose the propagation method which is available on N2Sky. After choosing a propagation method, the metadata from the ViNNSL template will be loaded. The metadata is represented as a form as shown in figure 64


NEURAL NETWORK DESCRIPTION
NEURAL NETWORK STRUCTURE
NEURAL NETWORK TRAINING
backpropagation
Name
Short description
-- select propagation type --
-- select learning type --
-- select application field --
-- select problem type --
NEXT

Network Type: Backpropagation
Creator: admin

Figure 64: Three Steps View. The Neural Network Description

The mandatory metadata from the ViNNSL template is:

- Name of the neural network
- Short description
- Propagation type
- Learning type
- Application field
- Problem type

This data can be customized on demand. Any additional field in metadata of the ViNNSL template will be reflected in this form.

After filling up the form, the user will be redirected to the "Neural Network Structure" tab.

6.5.2. The Neural Network Structure

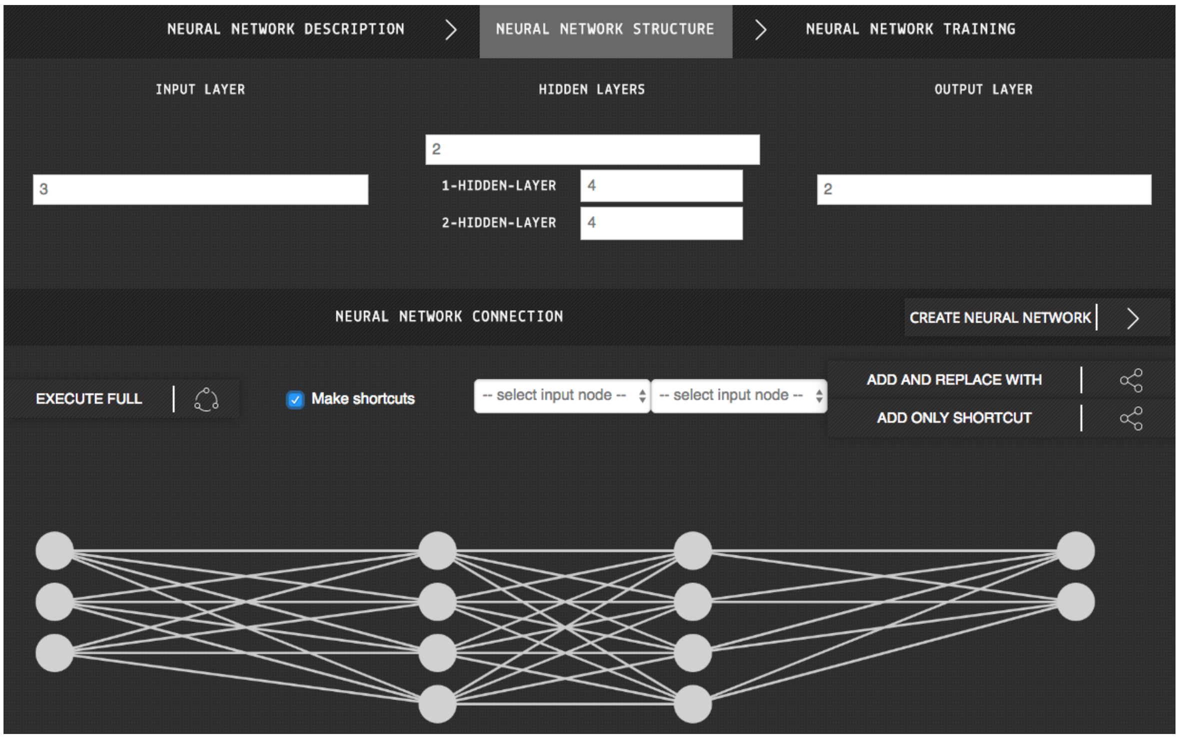
The neural network structure field represents the structure of the neural network, which can be customized. The connection between layers and nodes it is also achieved.

When the user enters this tab, he will see only these three types of layers:

- *Input Layer*, which represents the amount of nodes in the input layer.
- *Hidden Layers* can be multidimensional. It has the matrix structure, meaning that it is possible to choose multiple layers and nodes. The amount of nodes does not have to be equal in each layer.

- *Output Layer* is a layer, which represents the number of nodes for the output. This layer has to be validated, because of the difference between neural networks.

After choosing the correct numbers in each layer, the visual representation will be shown as displayed in figure 65.



INPUT LAYER: 3
HIDDEN LAYERS: 2
1-HIDDEN-LAYER: 4
2-HIDDEN-LAYER: 4
OUTPUT LAYER: 2

Figure 65: Three Steps View. The Neural Network Structure with full connection

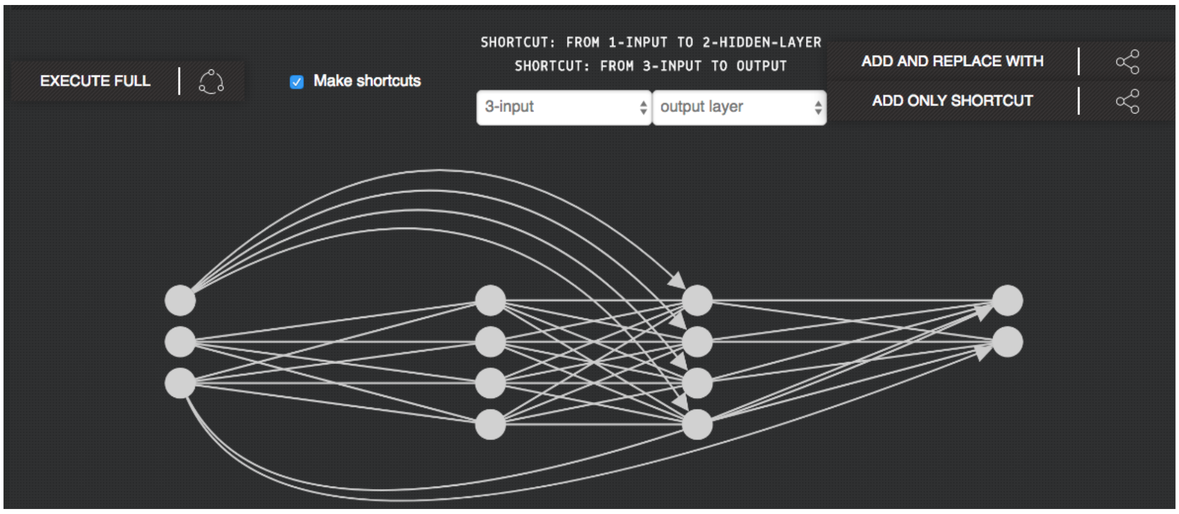
The next step is to connect the nodes. Following connections are possible:

- *Full Connection.* The nodes are fully connected when they have the connection to all activation with other layers. This is a standard approach in every neural network. The matrix multiplication is followed by the bias offset. [2]. This connection can be executed by clicking on "Execute full connection" button as shown in figure 65.

- *Pure Shortcut.* This Shortcut connection is also called skip connection. This connection enables unimpeded information flow and adds the direct connection to other layers. This approach can gain results by training. [40] With the pure connection, direct connection are possible. It will also remove the full connection to the chosen node.

- *Only shortcut.* It is the same approach as in pure shortcut, except the full connection still enabled.

In order to make shortcut connections more understandable, they are will be immediately visualized as shown in figure 66. In this example, the first input node has pure shortcut connection with a second hidden layer and the third input node has only shortcut connection with the output layer. The full connection with this node remains the same.



SHORTCUT: FROM 1-INPUT TO 2-HIDDEN-LAYER
SHORTCUT: FROM 3-INPUT TO OUTPUT

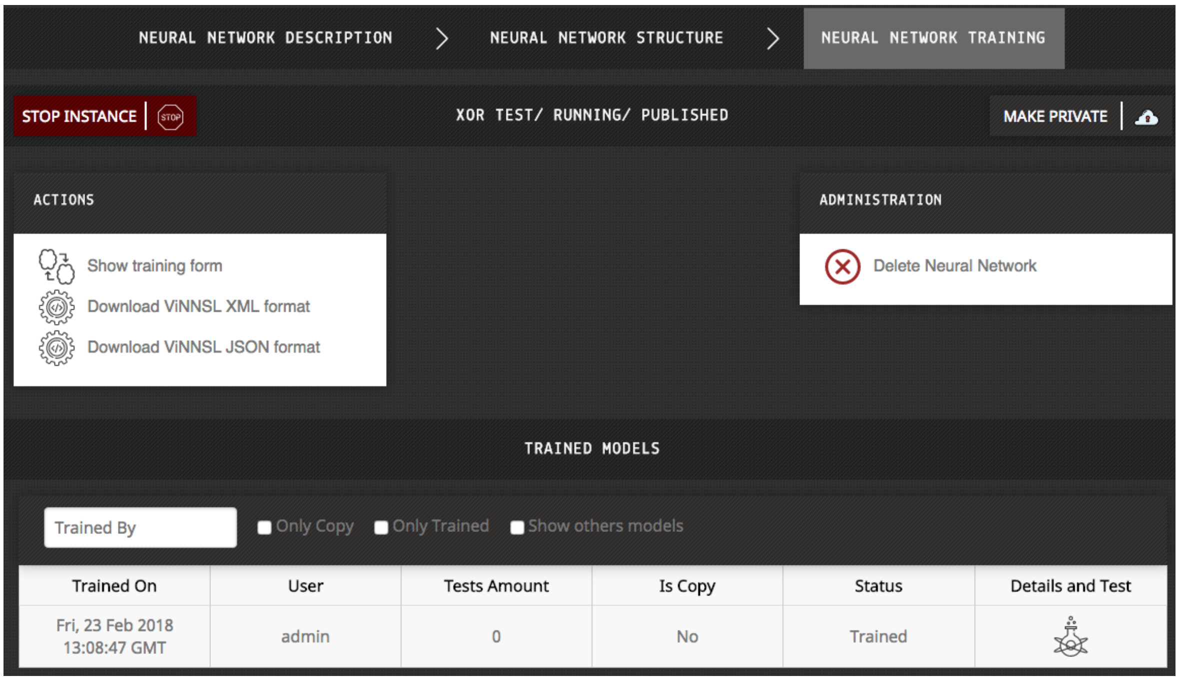
3-input | output layer

Figure 66: Three Steps View. The shortcut connections

After creating the neural network, the user will be redirected to the "Neural Network Training" tab. All the other tabs are clickable, but it is not possible to change either metadata nor the structure of the neural network.

6.5.3. The Neural Network Training

One of the most important steps in the three-step view workflow, because from this tab can be performed testing against the newly created neural network. The tab is multifunctional and scalable on permissions of the particular user as shown in figure 67.

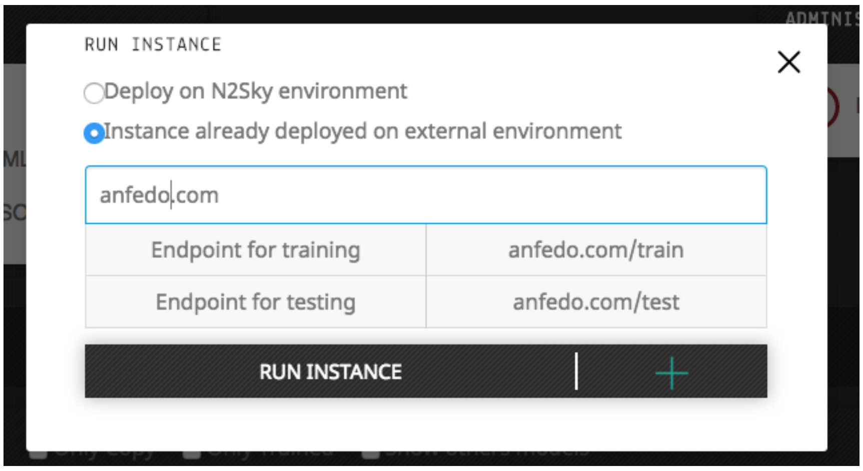


Trained On	User	Tests Amount	Is Copy	Status	Details and Test
Fri, 23 Feb 2018 13:08:47 GMT	admin	0	No	Trained	

Figure 67: Three Steps View. The Neural Network Training.

There are few components which are available to the user:

- *Administration.* This block is available only to neural network owner and the system administrator. The following operations are possible to perform from this block:
  - *Run the instance.* Without running the instance, there cannot be any testing. The user can run the neural network instance on the N2Sky Cloud or deploy his network on his own publicly available cloud as shown in figure 68.



Deploy on N2Sky environment [✓ Instance already deployed on external environment]

anfedo.com

Endpoint for training	anfedo.com/train
Endpoint for testing	anfedo.com/test

Figure 68: Three Steps View. The Neural Network Training. Run instance

Only two endpoints are available:

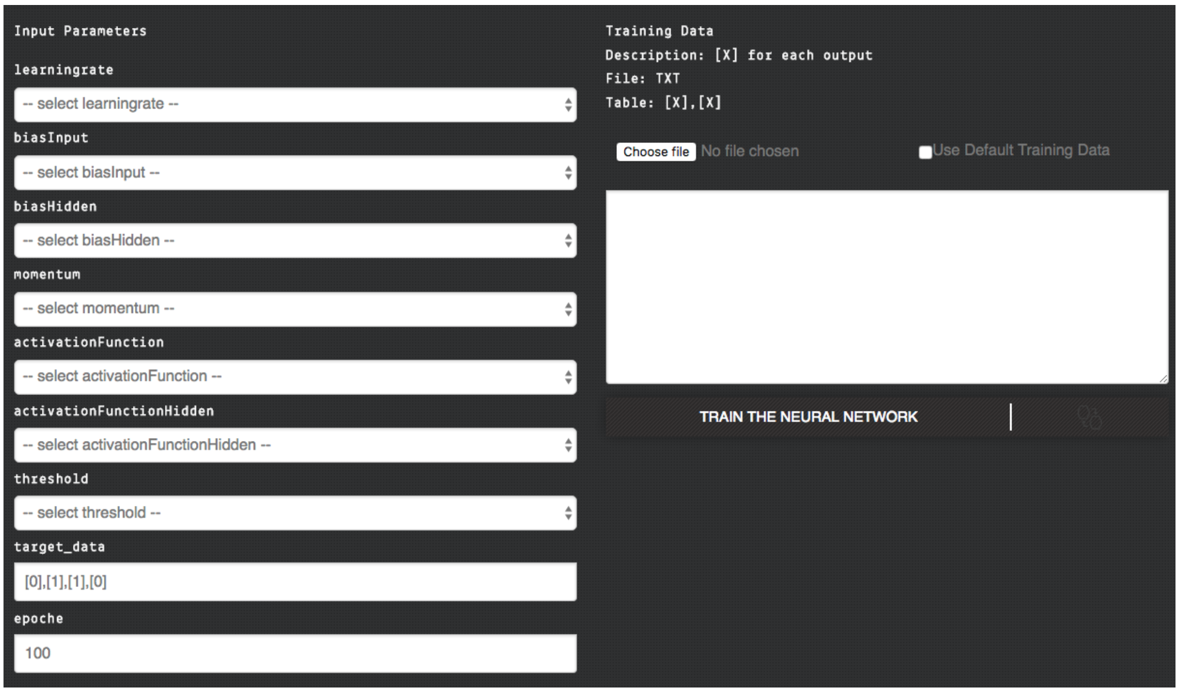
  * */train* - for training the neural network. This endpoint has to accept ViNNSL-formatted neural network description and the training data.
  * */test* - for evaluating the trained model. Has to accept only testing data.

  Detailed information of API can be found in the Development chapter subsection 9.2.

- *Stop the instance.* If the instance is running, it can be stopped. The instance will be removed from the N2Sky cloud if deployed there.
- *Publish the neural network.* The user can publish running neural network. In this case, the neural network and its trained models will be available in the neural network repository. The other users can copy published neural networks in their own projects.
- *Delete the neural network.* If the neural network owner or administrator will decide to remove the neural network, then all the trained models and testing data will also be removed. The neural network will not be published anymore and the running instance will be removed.

• *Actions.* The actions are available for every user if the neural network is published. The following actions can be performed:

- *Show training form.* On click, the training form will be expanded as shown in figure 69


Input Parameters

learningrate	-- select learningrate --
biasInput	-- select biasInput --
biasHidden	-- select biasHidden --
momentum	-- select momentum --
activationFunction	-- select activationFunction --
activationFunctionHidden	-- select activationFunctionHidden --
threshold	-- select threshold --
target_data	[0],[1],[1],[0]
epoche	100

Training Data
Description: [X] for each output
File: TXT
Table: [X],[X]

Choose file No file chosen
Use Default Training Data

Figure 69: Three Steps View. The Neural Network Training. Training form

The form data is fetched by reflection from ViNNSL Template, namely from input parameters. If there are possible values particularly set, the combo box will be shown. If there is only one default value set, the free text field will be displayed. On the right side, there is the training data information box. Uploading own training data or choosing default training data is also possible.

  – *Download ViNNSL XML format*
  – *Download ViNNSL JSON format*

- *Trained Models Table* is similar to the one in the table in "Model Repository." Semantic search against existing trained models can be performed. The user can see the status of the training and on the clock particular models, he will be redirected to the testing view and detailed information about training.

6.5.4. Training Results and the Model Evaluation

The testing is part of the model evaluation and that is why it is under the three-step-view workflow. The training results and model evaluation page are an underline of the workflow. From this page, the user can observe chosen training model, study the training graph and evaluate trained model as shown in figure 70.

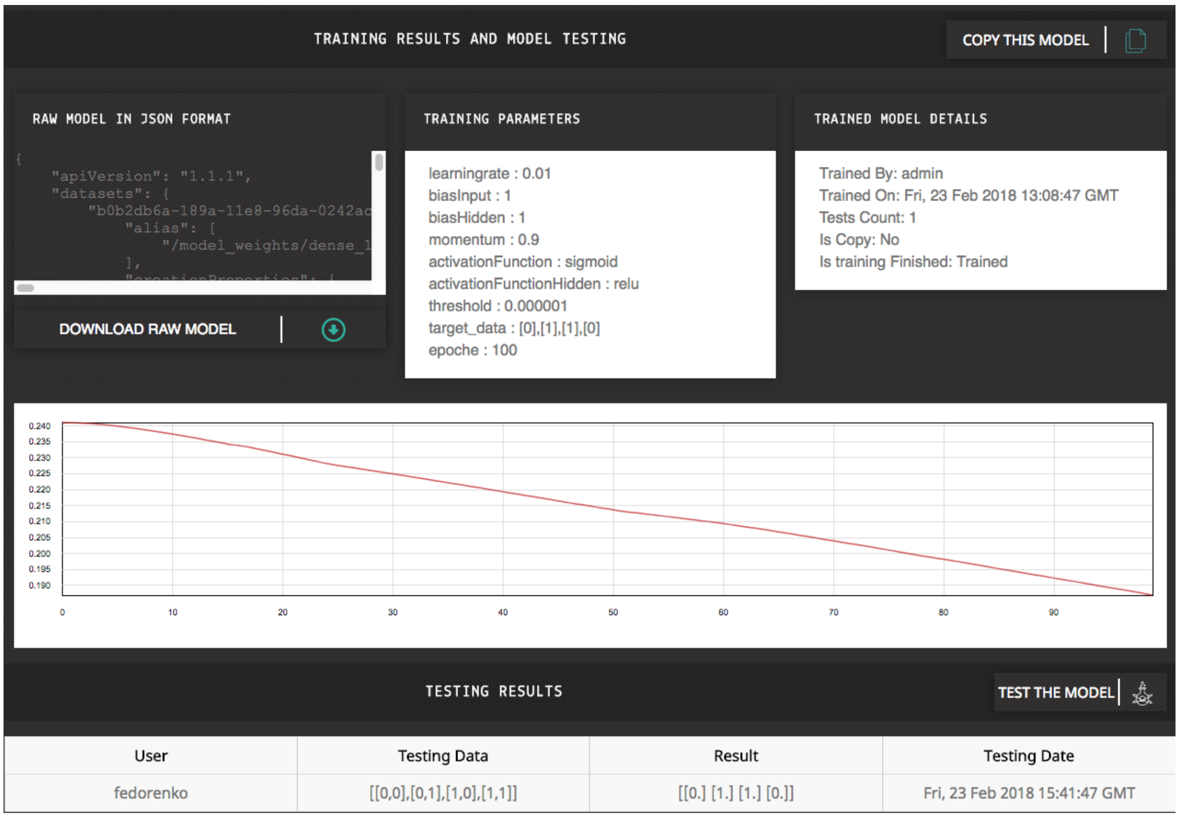
The page view contains following elements:

- *Functional bar* consists of the title itself and the functional copy of the project button. The functionality is almost the same as copying the neural network to the own project from the neural network repository. The user has to choose the project in the popup modal window and then perform copying as shown in figure 71. The trained model can be copied into multiple projects.

- *Raw model in JSON format*, which gives an overview of the trained model. The model can be viewed directly from the UI as well as be downloaded and reviewed from Desktop or mobile devices.

- *Training parameters* are the input parameters, which helped in the training of the neural network. The parameters can differentiate from paradigm to paradigm.

- *Trained model detail* is the metadata of the trained model. Namely, timestamp event metadata information. It contains following data:
  – Trained by user information
  – Trained on timestamp
  – Tests count (the tests, which are already performed)
  – Is a copy information



```
{
    "apiVersion": "1.1.1",
    "datasets": {
        "b0b2db6a-189a-11e8-96da-0242ac
            "alias": [
                "/model_weights/dense_1
            ],
            "creationProperties": [
```

learningrate : 0.01
biasInput : 1
biasHidden : 1
momentum : 0.9
activationFunction : sigmoid
activationFunctionHidden : relu
threshold : 0.000001
target_data : [0],[1],[1],[0]
epoche : 100

Trained By: admin
Trained On: Fri, 23 Feb 2018 13:08:47 GMT
Tests Count: 1
Is Copy: No
Is training Finished: Trained

User	Testing Data	Result	Testing Date
fedorenko	[[0,0],[0,1],[1,0],[1,1]]	[[0.] [1.] [1.] [0.]]	Fri, 23 Feb 2018 15:41:47 GMT

Figure 70: Three Steps View. Training Results and the Model Evaluation

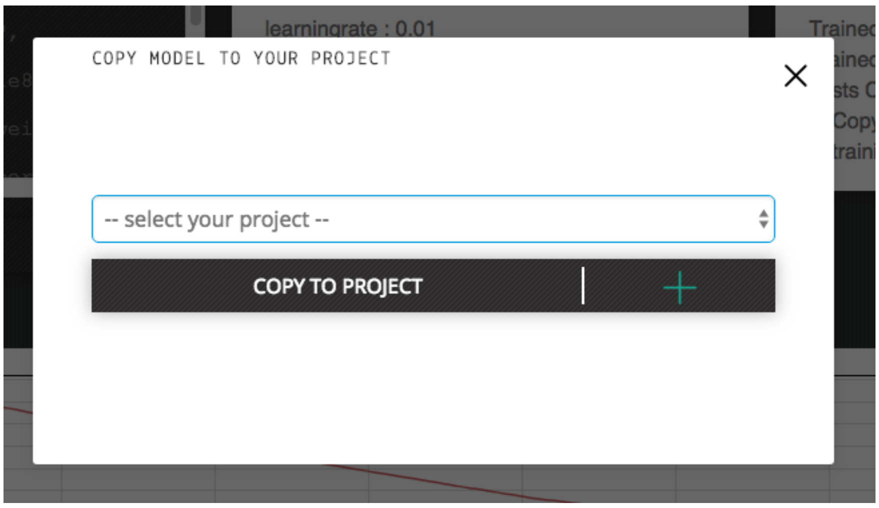


Figure 71: Three Steps View. Training Results and the Model Evaluation. Copy model into own project.

  - Is training finished information

- *Graph*, which represents epochs in x-axis and errors on the y-axis. If the neural network is still processing the training, the graph will move with it.
- *Test the model* button will initiate the testing popup modal window as shown in figure 72. The user has to choose the testing data, which could also be the default values.

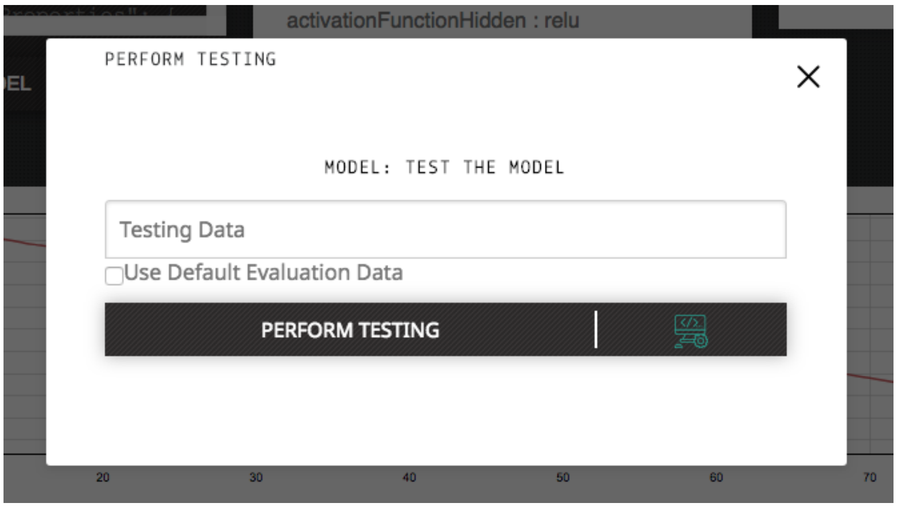
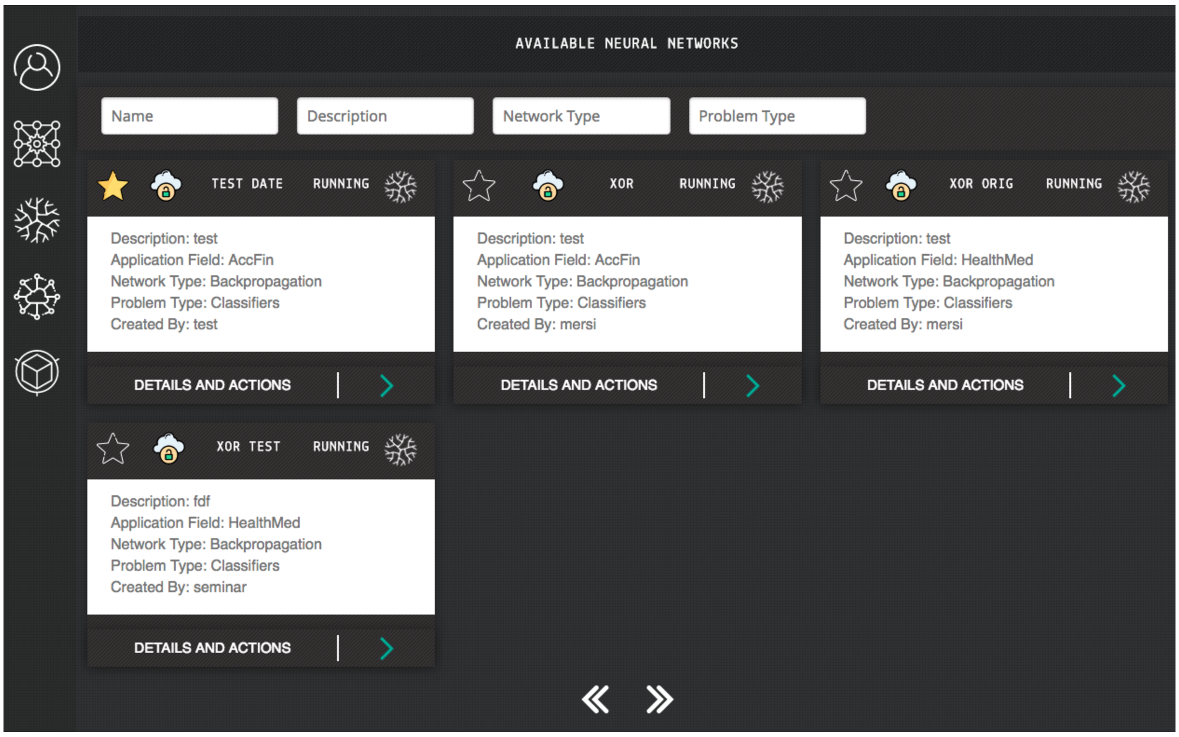


Figure 72: Three Steps View. Training Results and the Model Evaluation. The model testing.

- *Testing results* is a table. It shows the result of a testing. The user can see only his own testing results. If the user is the neural network owner or the system administrator, he can also see the results of other users. The table has the following columns:
  - The user who performs the testing
  - The testing data
  - Results (output)
  - Testing Date

6.6. Neural Networks Repository

The neural repository represents a repository, where any user can find and reuse available neural network. The neural network repository contains the search bar and the grid with the published neural networks as displayed in figure 73. If the neural network is published by the neural network owner, it is automatically added to the repository. It is important to mention, that only published networks of other users will be displayed.



TEST DATE RUNNING
Description: test
Application Field: AccFin
Network Type: Backpropagation
Problem Type: Classifiers
Created By: test

XOR RUNNING
Description: test
Application Field: AccFin
Network Type: Backpropagation
Problem Type: Classifiers
Created By: mersi

XOR ORIG RUNNING
Description: test
Application Field: HealthMed
Network Type: Backpropagation
Problem Type: Classifiers
Created By: mersi

XOR TEST RUNNING
Description: fdf
Application Field: HealthMed
Network Type: Backpropagation
Problem Type: Classifiers
Created By: seminar

Figure 73: N2Sky Neural network repository

1. *Searching bar.* With the searching bar, the semantic search can be performed against available neural networks in the repository. Search is done by using the:
    - Name of the neural network
    - Description of the neural network
    - Network Type
    - Problem Type

   Each field is case insensitive. The inquiry will be processed and converted in order to give most reliable results.

2. *Networks grid* is a grid, which represents the published neural networks. It has three columns in the Desktop version and one in the mobile version. Every grid item has a title, which contains following components:
    - The save button-icon. The button is displayed as a star. If the star is colored, then the neural network has already been saved. If the icon is transparent, it can be saved. On click of the transparent button, the popup modal window will appear as shown in figure 74. From the modal window, the user can choose available projects, where he can copy chosen neural networks.

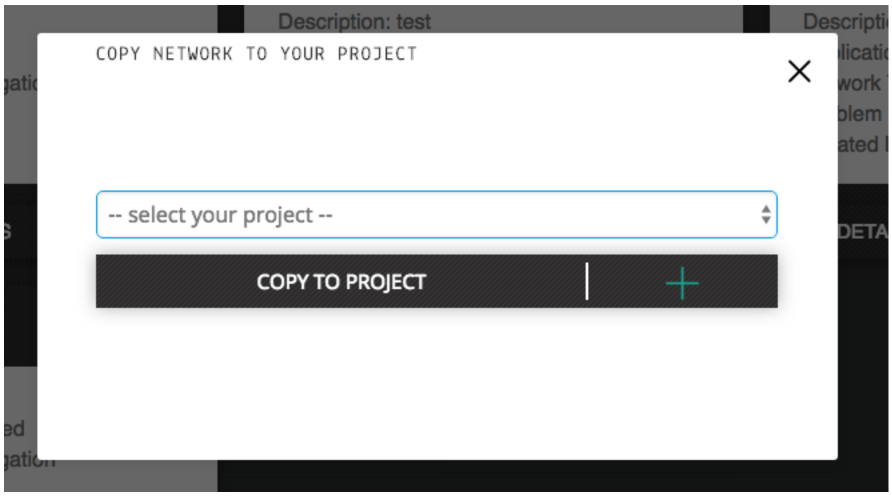


Figure 74: N2Sky Neural network repository. Copy the neural network.

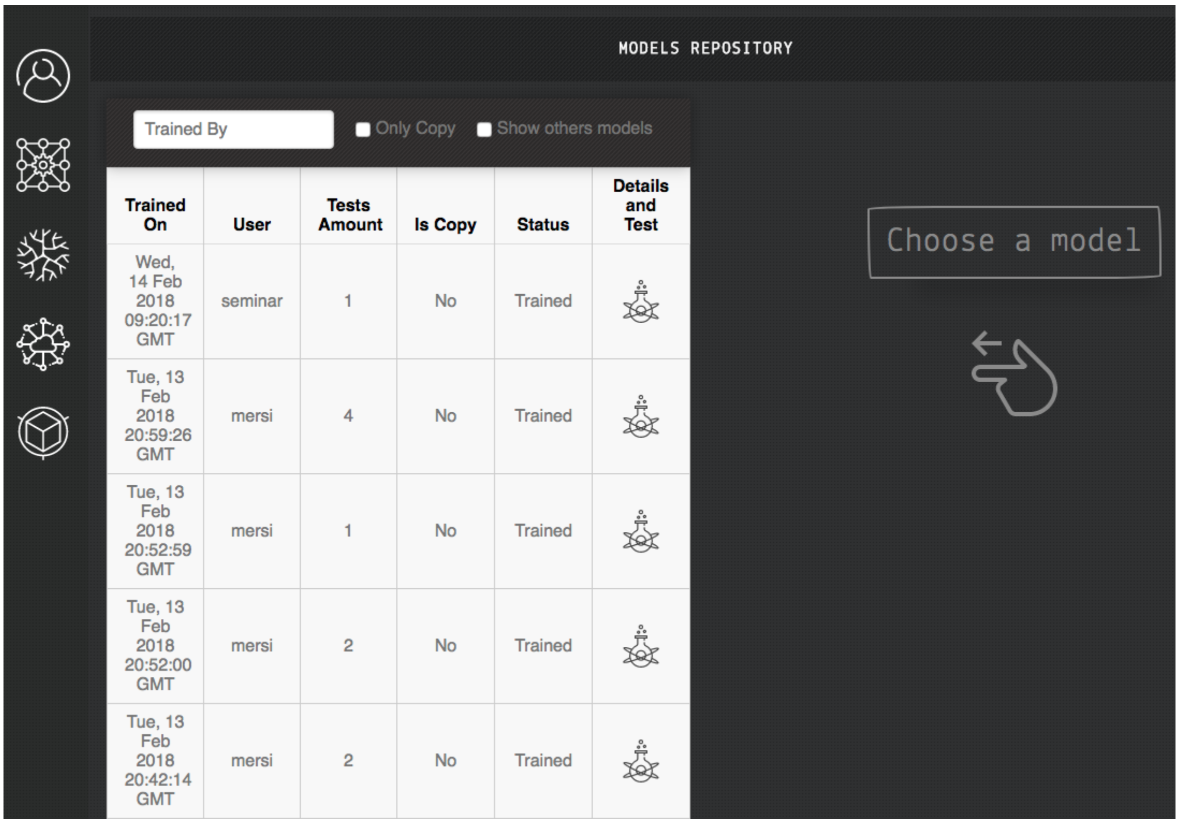
- Availability of the neural network. Can be public (green), private (red) and restricted (yellow). The arbitrary user can observe only public neural networks, while the administrator can observe any neural network in N2Sky.
- The name of the neural network
- The running status. Can be running and not running.
- The running icon. Can be spinning icon for running and standing alone is for not running the network.

Besides the details and actions button, which redirect the user to the neural network details, the brief network information is deployed:

- Short description
- Application field
- Network type
- Problem type
- Created by

6.7. Models Repository

The modal repository is similar to the neural networks repository but contains only trained models. The structure is also different. Because of the big amount of the trained models, the models are displayed in the table as shown in figure 75.



Trained On	User	Tests Amount	Is Copy	Status	Details and Test
Wed, 14 Feb 2018 09:20:17 GMT	seminar	1	No	Trained	⚗
Tue, 13 Feb 2018 20:59:26 GMT	mersi	4	No	Trained	⚗
Tue, 13 Feb 2018 20:52:59 GMT	mersi	1	No	Trained	⚗
Tue, 13 Feb 2018 20:52:00 GMT	mersi	2	No	Trained	⚗
Tue, 13 Feb 2018 20:42:14 GMT	mersi	2	No	Trained	⚗

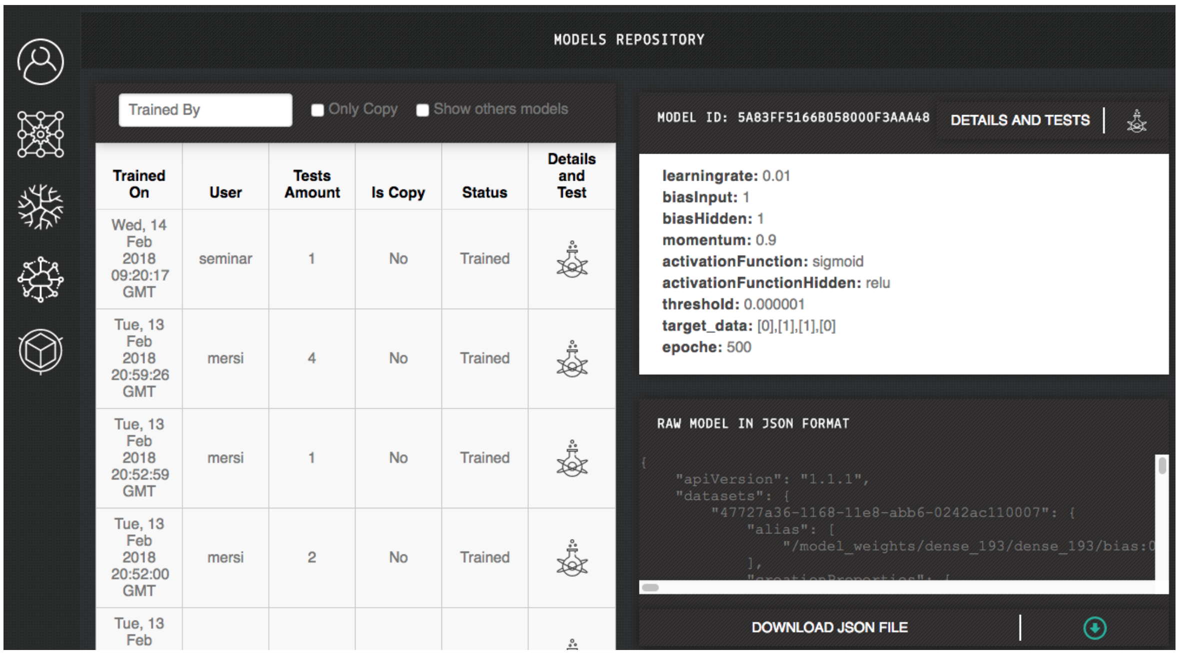
Figure 75: N2Sky Models Repository

The model repository page view contains a models table and a brief description of the chosen model:

- *Trained model table* consists of the search bar and table itself.
  - The search bar performs the semantic search against available trained models. It has following filters:
    - Trained by
    - Only copy checkbox
    - Show other models checkbox
  - The table. On click of the table, the short description of the trained model will be opened. The table contains following elements:
    - Trained on (timestamp)
    - User
    - Tests amount
    - Is copy
    - Status

∗ Details and tests

- *The short description* is located on the right side of the page. When the user enters the page, there will be only notifications, which motivates the user to chose the model. After choosing it, the view will be replaced with a short description as shown in figure 76.



Trained On	User	Tests Amount	Is Copy	Status	Details and Test
Wed, 14 Feb 2018 09:20:17 GMT	seminar	1	No	Trained	
Tue, 13 Feb 2018 20:59:26 GMT	mersi	4	No	Trained	
Tue, 13 Feb 2018 20:52:59 GMT	mersi	1	No	Trained	
Tue, 13 Feb 2018 20:52:00 GMT	mersi	2	No	Trained	
Tue, 13 Feb					

MODEL ID: 5A83FF5166B058000F3AAA48

learningrate: 0.01
biasInput: 1
biasHidden: 1
momentum: 0.9
activationFunction: sigmoid
activationFunctionHidden: relu
threshold: 0.000001
target_data: [0],[1],[1],[0]
epoche: 500

RAW MODEL IN JSON FORMAT

```
{
    "apiVersion": "1.1.1",
    "datasets": {
        "47727a36-1168-11e8-abb6-0242ac110007": {
            "alias": [
                "/model_weights/dense_193/dense_193/bias:0
            ],
```

Figure 76: N2Sky Models Repository. Short Description

The description contains following components:

- Title, which consists of the ID of the model and button, which will redirects to details of the chosen model.
- The list of the input parameters. This list can differ, depending on neural network paradigm.
- The raw model, which can be downloaded.

7. Tutorial

The tutorial describes a few main cases of the arbitrary user, neural network engineer and contributor user. The use cases will be described in UML Activity Diagram. The details will be skipped, only the main function will be described.

7.1. Using Network and Model Repositories

The arbitrary user does not have a lot of experience in the neural network field. In most cases, he just wants to get into the subject by learning on the N2Sky platform.

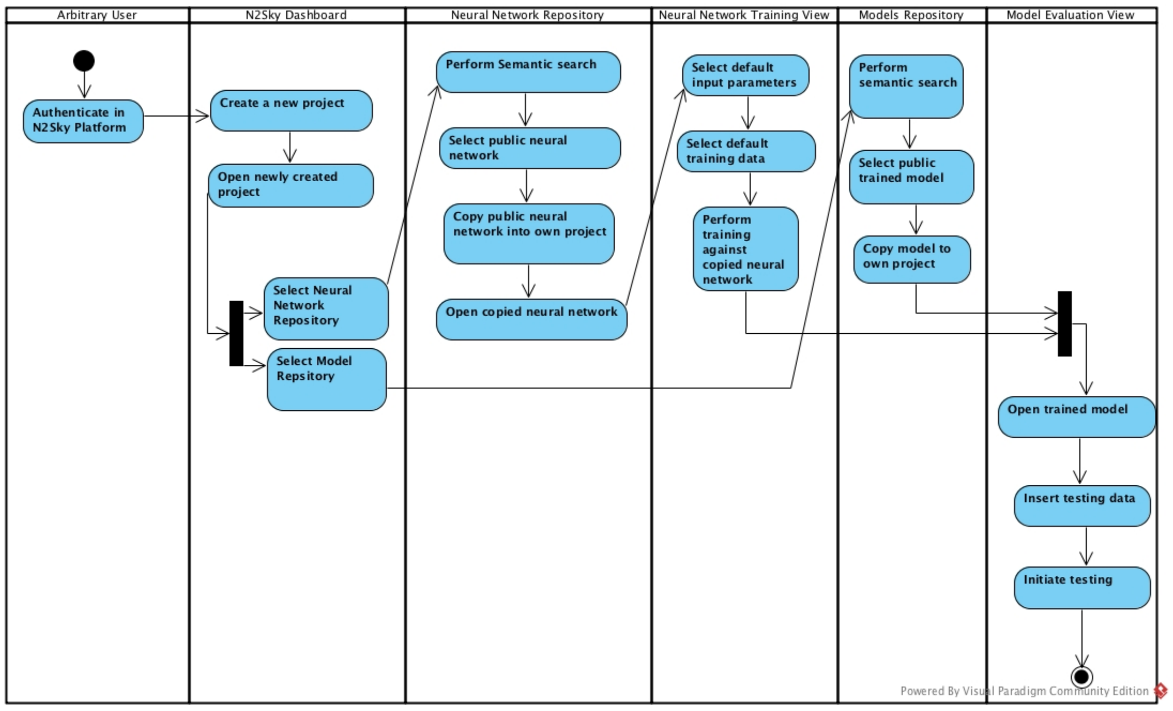

Arbitrary User
N2Sky Dashboard
Neural Network Repository
Neural Network Training View
Models Repository
Model Evaluation View
Authenticate in N2Sky Platform
Create a new project
Open newly created project
Select Neural Network Repository
Select Model Repsitory
Perform Semantic search
Select public neural network
Copy public neural network into own project
Open copied neural network
Select default input parameters
Select default training data
Perform training against copied neural network
Perform semantic search
Select public trained model
Copy model to own project
Open trained model
Insert testing data
Initiate testing
Powered By Visual Paradigm Community Edition

Figure 77: Tutorial. Activity Diagram of using Network and Model Repositories

The activity diagram, displayed in figure 77, shows a typical workflow of using the neural network and the models repositories. There are six main components of the diagram, which are participating in this workflow:

1. The arbitrary user
2. N2Sky Dashboard
3. Neural Network Repository
4. Neural Network Training View
5. Models Repository

6. Model Evaluation View

Every participant is represented as a pool of activities. The user has to proceed through the following steps:

- *The arbitrary user*
  - *Authenticate in N2Sky platform* is the first step for every user, which starts to use the N2Sky platform. The user has to either log in with existing credentials or signup on the start page as shown in figure 78

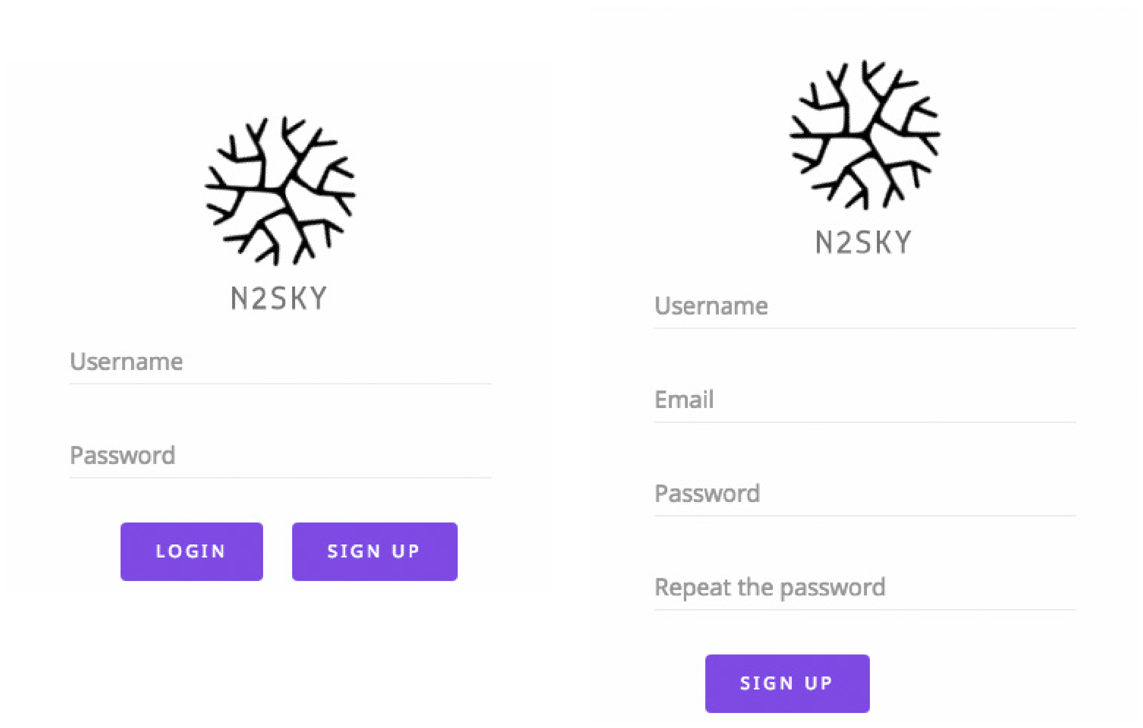


Figure 78: Tutorial. Login form on the left sight and the signup form on the right.

- *N2Sky Dashboard*
  - *Create a new project.* The user finds himself in his own dashboard and clicks on the "Create project" button. The popup modal window will appear and the user can type the name of the project and a short description. After the project is created, it will authentically appear on the dashboard of the user.
  - *Open newly created project* by clicking on the folder with the newly created project name.

  - *Select Neural Network Repository.* When the user will be in the project view, he will see the list of the tools in the top bar. One of the tools is "Neural network repository". By clicking on it, the user will be redirected to Neural Network Repository page.

- *Neural Network Repository*
  - *Perform Semantic Search.* The user will see on the top bar of the page the searching fields with filters. The user can perform the semantic search in order to find the desirable neural network.
  - *Select public neural network.* The arbitrary user can select publicly available neural network. He can observe the details of the neural network.
  - *Copy public neural network into own project.* If the user is satisfied with the selected neural network, he can copy it into his own project. The copy button is located on the right of the neural network and it is star shaped. On click, the popup window will appear and the user can select his project.
  - *Open copied neural network.* After copying the neural network, the user can upload it and perform training. The arbitrary user does not have permission to see how other people used the neural network.
- *Neural Network Training View.* This view is part of the "3-steps-view" creation of the neural network from paradigm. For arbitrary users, all other views are available in read-only mode.
  - *Select default input parameters.* Since the arbitrary user does not have experience, he is always going with default values in every form.
  - *Select default training data.* The same default possibilities choses the arbitrary user, just by clicking "User default values".
  - *Perform training against copied neural network.* After selecting all default values, the user can click on the "Perform Training" button. The training status will appear in the table below.
- *Models Repository.* This is the second possibility of the arbitrary user if he choses the "Model Repository" tool from his own project in N2Sky Dashboard.
  - *Perform Semantic Search.* The user will see on the top bar of the page the searching fields with filters. The user can perform the semantic search in order to find a desirable trained model.
  - *Select trained model.* The user can select and observe trained model before copying it.
  - *Copy model into the own project* by clicking on "Copy" button. The popup modal window will appear and the user can choose his project.

- *Model Evaluation View.* This step merges the selection of the neural network with performed trained model and the one of the existing trained model because the next activities are same for both cases.
  - *Open trained model.* The user clicks on the trained model and the detailed information with training data will appear.
  - *Insert testing data.* The user clicks on "Test model" button and the popup window will appear. The arbitrary user uses default testing data without changing it.
  - *Initiate testing.* After proceeding with the testing, the evaluation information will appear in the table. The user can study the data and check if the testing results are correct.

7.2. Creation of the Neural Network from the Existing Paradigm

This case is related to the neural network engineer user. Some steps are familiar from the previous case with the arbitrary user, but mostly the user of the platform is different. The arbitrary user wanted to use everywhere default data and copy existing neural networks and model. The neural network engineer user is an already experienced user which can create his own neural network from existing paradigms and perform training or testing with his own data.

The activity diagram, displayed in figure 79, shows a typical workflow of creating the neural network from existing paradigms. There are four main components of the diagram, which are participating in this workflow:

1. The neural network engineer user
2. N2Sky Dashboard
3. The "3-step-view" with sub-components:
   - Neural network description
   - Neural network structure
   - Neural network training
4. Model Evaluation View

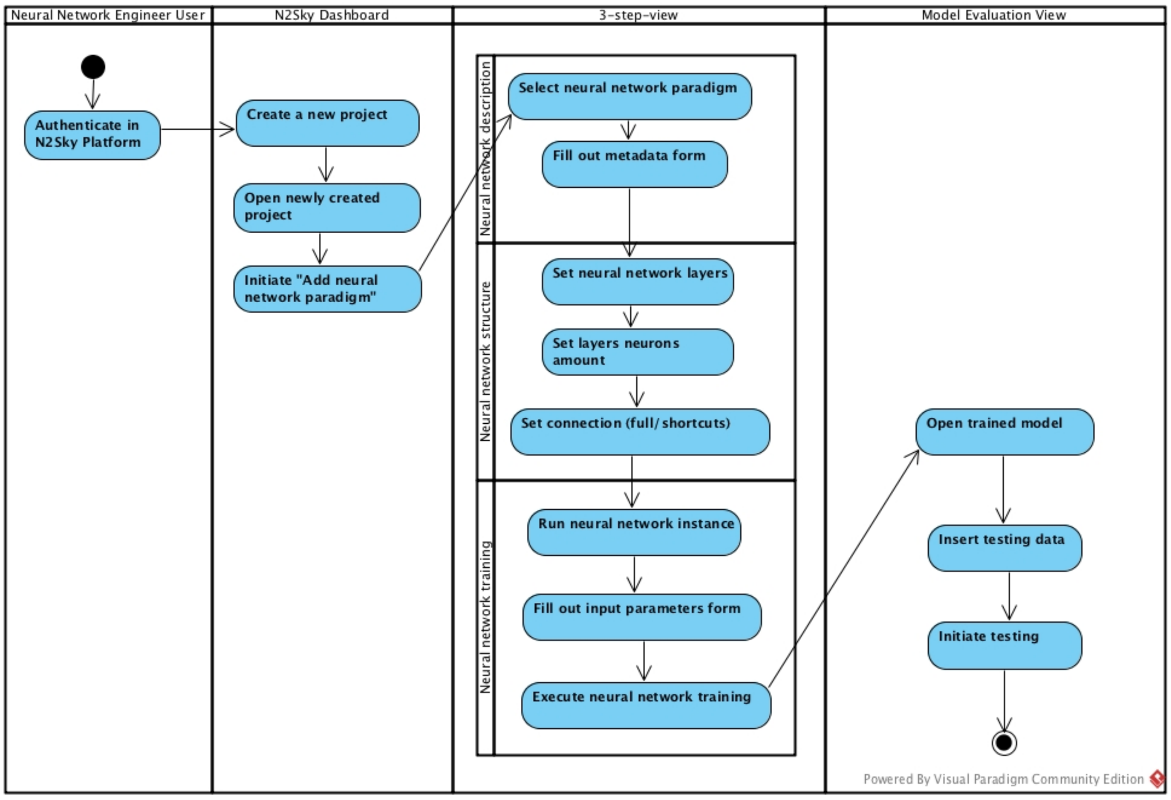

Neural Network Engineer User
N2Sky Dashboard
3-step-view
Model Evaluation View
Authenticate in N2Sky Platform
Create a new project
Open newly created project
Initiate "Add neural network paradigm"
Neural network description
Select neural network paradigm
Fill out metadata form
Neural network structure
Set neural network layers
Set layers neurons amount
Set connection (full/shortcuts)
Neural network training
Run neural network instance
Fill out input parameters form
Execute neural network training
Open trained model
Insert testing data
Initiate testing
Powered By Visual Paradigm Community Edition

Figure 79: Tutorial. Activity Diagram of creation the neural network from the existing paradigm

Every participant is represented as a pool of activities. The user has to proceed through the following steps:

- *The neural network engineer user*
  - *Authenticate in N2Sky platform* is the same step as by the arbitrary user.
- *N2Sky dashboard.* Create the project and open newly created project activities are the same as in the previous tutorial with the arbitrary user. The only difference is when choosing the tool in which the user goes with the "Add neural network paradigm".
- *3-step-vew* is the simplified way of the creation of a neural network from the paradigm. The following tabs and activities are represented:
  - *Neural network description* the first tab that the user sees after being redirected to the page from the project.
    - *Select neural network paradigm.* The user can choose available paradigms from the list. After he selects one, the form with metadata will be loaded.

        * *Fill out metadata form.* The form is loaded by reflection parameters from the ViNNSL template. The user needs to follow the fields with possible values and choose one in each.

    - *Neural network structure* The second tab of the "3-steps-view" after user clicking "Next" button.

        * *Set neural network layers.* The user will see three layers: the input layer, hidden layer, and output layer. Every layer can be single except of the hidden one. The user can choose the amount of the hidden layers.

        * *Set layers neurones amount.* After choosing the layers, the user has to type the amount of nodes in each layer. After typing it in, the graphical representation will appear.

        * *Set connection (full/shortcuts).* Three buttons are available:

            · Execute full connection, which will make the fully connected neural network between all nodes.

            · Pure shortcut, which will remove the full connection and replace it with the shortcut.

            · Only shortcut, which will leave the full connection and additionally add the shortcut.

          The user can choose multiple possibilities and make an absolutely customized neural network structure.

    - *Neural network training* the last and the most important tabs of the "3-step-view". From this tab, the user can perform training and manipulate neural network instances.

        * *Run neural network instance.* The user clicks on "Run neural network" and the popup modal window will appear. It is possible to choose the N2Sky cloud or own cloud. The engineer user does not own a cloud and he chooses the N2Sky cloud. After it, the neural network instance is spawning on the N2Sky cloud.

        * *Fill out parameters form.* Almost the same step as by the arbitrary user, except the engineer user, does not use the default values, but types in everything by himself.

        * *Execute neural network training.* The user performs training with his own training data, which he has uploaded on the N2Sky platform.

- *Model Evaluation View.* The same step as in the previous tutorial, but in this case, the user uses his own testing data form by evaluating the trained model.

7.3. Upload Own Neural Network Paradigm

This tutorial is related to the contributor user, which has a long-term experience with neural network field and wants to test his own projects on the N2Sky platform. Some steps are common, as in the arbitrary and neural network engineer user, but the interaction with the N2Sky platform is different.

The activity diagram, displayed in figure 80, shows a typical workflow of uploading own neural network paradigm. There are four main components of the diagram, which are participating in this workflow:

1. The contributor user
2. N2Sky Dashboard
3. Neural network training (the last step of the "3-step-view")
4. Model Evaluation View

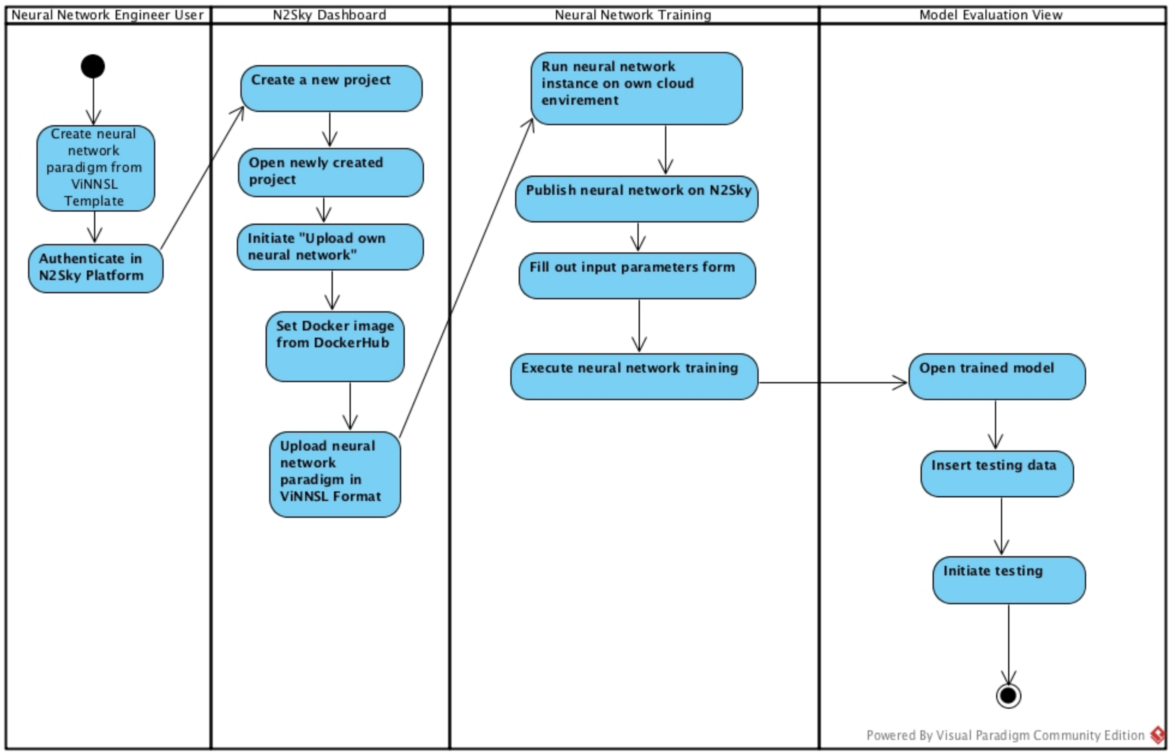

Neural Network Engineer User
N2Sky Dashboard
Neural Network Training
Model Evaluation View
Create neural network paradigm from ViNNSL Template
Authenticate in N2Sky Platform
Create a new project
Open newly created project
Initiate "Upload own neural network"
Set Docker image from DockerHub
Upload neural network paradigm in ViNNSL Format
Run neural network instance on own cloud envirement
Publish neural network on N2Sky
Fill out input parameters form
Execute neural network training
Open trained model
Insert testing data
Initiate testing
Powered By Visual Paradigm Community Edition

Figure 80: Tutorial. Activity Diagram of upload own neural network paradigm

- *The contributor user*
  - *Create neural network paradigm from the ViNNSL template.* This is a pre-defined step, which the contributor user has to make before starting the workflow. Making an own ViNNSL description is described in section 4.

  - *Authenticate in N2Sky platform* is the same step as in the arbitrary user.

- *N2Sky Dashboard.* The creating a new project and opening it are the same activities as described in previous tutorials. The following new steps the user has to make are these:
  - *Initiate "Upload own neural network"*, which will open the popup window where the user can fill out the form.
  - *Set Docker imaged from the DockerHub* since the contributor user uploads an own neural network, he has to push it on DockerHub repository. The N2Sky platform finds this image by DockerHub username and the image name.
  - *Upload neural network paradigm in ViNNSL format.* The user uploads to the N2Sky platform his neural network either in ViNNSL XML or in ViNNSL JSON. After uploading it, the user can overview and even edit his description.
- *Neural Network Training.* The last tab of the "3-step-view", which allows the user to perform training and manipulate the neural network instance.
  - *Run neural network instance on own cloud environment.* Similar as in the neural network engineer user, except the contributor user, will run the instance in his own environment. In other words, the instance is already running on the contributors cloud, the user just needs to set host and endpoints to the N2Sky platform.
  - *Publish neural network on N2Sky* the user can publish neural network on N2Sky by pressing the "publish" button. In this case, the neural network will be available in neural network repository and other users can use it.
  - *Execute neural network training.* The similar procedure as in the engineer user. The contributor user uses his own training data.
- *Model Evaluation View.* The same step as in the previous tutorial, except that the user, as the neural network owner can see the trained models from other users.

8. Use Cases

In this chapter, the typical usages of the N2Sky platform will be described.

8.1. N2Sky as a Learning Platform

The N2Sky platform was developed originally in university for students starting from N2Grid [30]. Today, N2Sky is still mainly working for students and people, who are interested in neural network field without deep knowledge of it.

The most students and young people are using mobile smart devices like mobile phones or tablets for multiple purposes. The N2Sky platform is fully supporting any kind of device, that is why by assumption the students will use the mobile device in this use case.

In figure 81 the typical student use case is displayed.

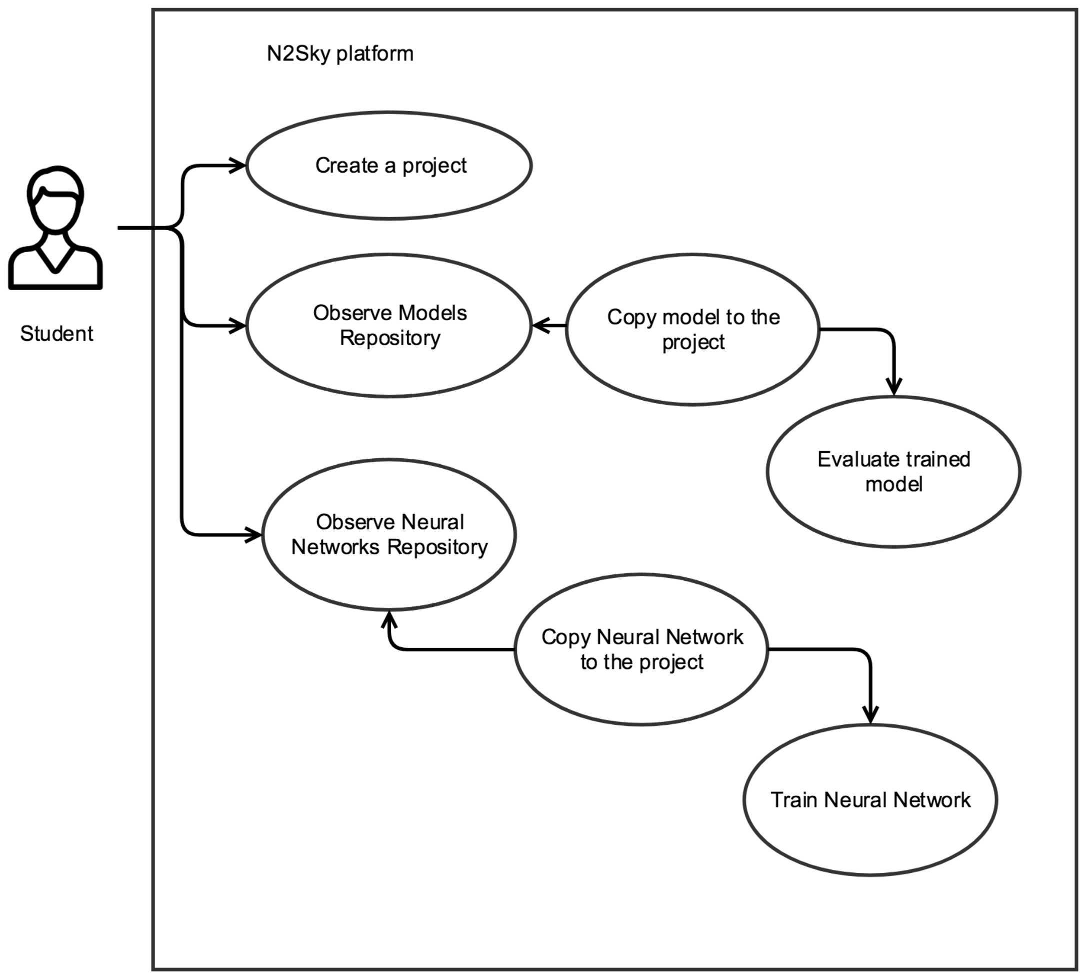

N2Sky platform
Student
Create a project
Observe Models Repository
Copy model to the project
Evaluate trained model
Observe Neural Networks Repository
Copy Neural Network to the project
Train Neural Network

Figure 81: Use Case of typical arbitrary user behaviour

The student is an arbitrary user. The detailed information of main functions of the users are described in subsubsection 2.2.4.

This user can create a project, which it will be in the focus after the first login. The project is a collection of the created or copied neural networks and trained models. After creating the project, the arbitrary user does not have to develop anything or know any programming language in order to use the N2Sky platform. The student can search for an existing neural network in the Neural Networks repository or for existing trained model in the Models Repository.

The user can reuse, namely copy existing neural network or trained model. Before doing this, the arbitrary user can check the details of the neural network or trained model. The user can observe how the neural network was behaving and study already existing trained models.

After copying the neural network or the trained model, the arbitrary user can use the default data across the whole training and evaluation process. The main idea is that the user will learn. The user can test different parameters and observe the results. With this approach, the student can study the behaviour of the neural network.

All these procedures, can be performed directly from the mobile device. Every action is intuitive and fully adapted to the device type.

8.2. Developing the Neural Network with N2Sky

The typical use case when the user has experience in neural network field or the user already developed his own neural network and wants to test it, is displayed in figure 82.

This user operates differently with N2Sky. In most cases, he will use the desktop version of the application. This category of users can be more professional. Some users could use technical jargon while others know just the neural network field more theoretically, then practically. More details about user roles are described in subsubsection 2.2.4.

If the user knows the field theoretically and does not have experience in the actual creation of the neural network, N2Sky gives such a possibility. With the neural network generator, the user can choose existing paradigms and create the neural network with a few simple steps.

If the user has experience in the actual creation of the neural network, he can write a ViNNSL formatted neural network description and upload it to the N2Sky platform. Since the user has developed neural networks before, he could also use his own cloud in order to deploy it.

The next steps are the same in case if the user creates the neural network from the paradigm or upload his own neural network on the N2Sky platform:

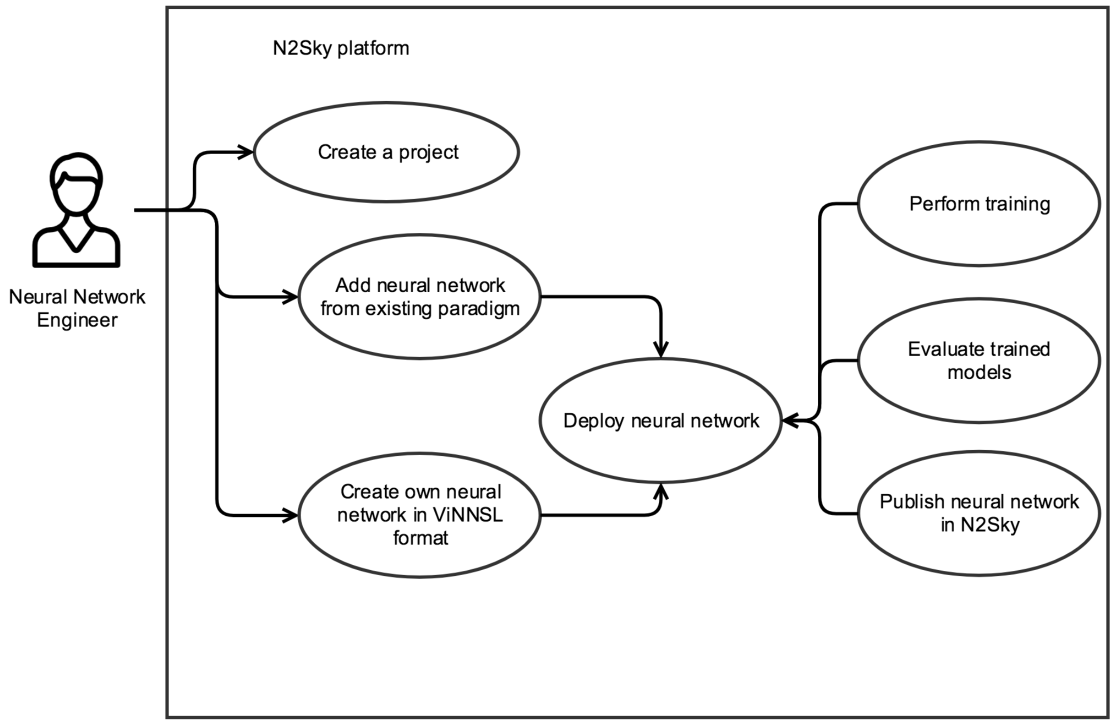

N2Sky platform
Neural Network Engineer
Create a project
Add neural network from existing paradigm
Create own neural network in ViNNSL format
Deploy neural network
Perform training
Evaluate trained models
Publish neural network in N2Sky

Figure 82: Use Case. Developing the neural network with N2Sky.

- *Perform training.* Unlike arbitrary users, the neural network engineers or the contributor users can use own training data.
- *Evaluate the trained models.* The users can perform tests against trained model with their own testing data.
- *Publish neural network on N2Sky* in order to make it available in the neural network repository. When the neural network will be published, arbitrary users can reuse it for their own need.

9. Developer Guide

9.1. System Configuration

9.1.1. Prerequirements

Following technologies has been used:

Table 3: My caption

Technology	**Version**
Docker CE	17.12.1-ce
MongoDB	3.6
NodeJS	7.0.0
npm	4.2.0
Prometheus	2.0.0
node_exporter	0.16.0-rc.0
Alertmanager	0.15.0-rc.0
React	15.6.1
Redux	3.7.2

Every N2Sky service has "package.json" file on the root of the project. This file contains versions of all used packages in particular service.

9.1.2. 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:

```
        docker run -d -p 27017:27017 --name database \
         -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:

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

Create database superuser:

```

db.createUser(
        { user: 'admin', pwd: 'password',
         roles: [
                { role: "userAdminAnyDatabase", db: "admin" }
                          ] });
```

Create database:

```
use n2sky;
```

Create database n2sky user:

```

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

Create system administrator:

```

db.users.insert({
    "name":"admin",
    "password":"password",
    "email":"admin@test.com",
    "type":"admin",
    "active":true
});
```

For production version it is recommended to enable authentication:

```

        docker rm -f database

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

9.1.3. Setting Up the Cloud Web Service

Clone the project from the repository

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

Go to service Dockerfile

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

Build the Docker image

Dockerfile:

```
FROM node:7
WORKDIR /app
COPY package.json .
RUN npm install
RUN npm install -g forever
COPY . /app
CMD forever -c 'node --harmony' server.js
EXPOSE 8080
```

Command:

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

Run the service Docker container

```
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*

9.1.4. Setting Up the Model Repository Service

Clone the project from the repository

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

Go to service Dockerfile

```
cd modelRepository
```

Build the Docker image

Dockerfile:

```
FROM node:7
WORKDIR /app
COPY package.json .
RUN npm install
RUN npm install -g forever
COPY . /app
CMD forever -c 'node --harmony' server.js
EXPOSE 8080
```

Command:

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

Run the service Docker container

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

9.1.5. Setting Up the User Management Service

Clone the project from the repository

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

Go to service Dockerfile

```
cd user-management
```

Build the Docker image

Dockerfile:

```
FROM node:7
WORKDIR /app
COPY package.json .
RUN npm install
RUN npm install -g forever
COPY . /app
CMD forever -c 'node --harmony' server.js
EXPOSE 8080
```

Command:

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

Run the service Docker container

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

9.1.6. Setting Up the N2Sky Frontend

Clone the project from the repository

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

Go to service Dockerfile

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

Build the Docker image Dockerfile:

```
FROM node:7
WORKDIR /app
COPY package.json .
RUN npm install
COPY . /app
CMD npm run dev
EXPOSE 9593
```

Command:

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

Run the service Docker container

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

9.1.7. Setting Up the Backpropogation Neural Network

This neural network made for demonstration purposes.

Clone the project from the repository

```
git clone https://github.com/CN2Sky/backprop.git
```

Go to service Dockerfile

```
cd backprop
```

Build the Docker image

Dockerfile:

```
FROM debian:8

MAINTAINER Kamil Kwiek <kamil.kwiek@continuum.io>

ENV LANG=C.UTF-8 LC_ALL=C.UTF-8

```

```
RUN apt-get update --fix-missing && apt-get install -y wget bzip2 ca-certificates \
    libglib2.0-0 libxext6 libsm6 libxrender1 \
    git mercurial subversion

RUN echo 'export PATH=/opt/conda/bin:$PATH' > /etc/profile.d/conda.sh && \
    wget --quiet https://repo.continuum.io/archive/Anaconda2-5.0.1-Linux-x86_64.sh
     ~/anaconda.sh && \
    /bin/bash ~/anaconda.sh -b -p /opt/conda && \
    rm ~/anaconda.sh

RUN apt-get install -y curl grep sed dpkg && \
    TINI_VERSION=`curl https://github.com/krallin/tini/releases/latest | \
    grep -o "/v.*\"" | sed 's:^..\(.*\).$:\1:'` && \
    curl -L "https://github.com/krallin/tini/releases/download/v \
    ${TINI_VERSION}/tini_${TINI_VERSION}.deb" > tini.deb && \
    dpkg -i tini.deb && \
    rm tini.deb && \
    apt-get clean

ENV PATH /opt/conda/bin:$PATH

# Jupyter has issues with being run directly: https://github.com/ipython/ipython/is
COPY backprop /root/

# Expose Ports for TensorBoard (6006), Ipython (8888)
EXPOSE 6006 8888 5000

WORKDIR "/root"

RUN pip install keras
RUN pip install tensorflow
RUN pip install h5json

CMD ["python", "server.py"]
```

Command:

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

Run the service Docker container

```
docker run -d -p 6006:6006 -p 8888:8888 -p 5000:5000 --name backprop backprop:1
```

9.1.8. Setting Up the Monitoring System

To create custom image (OS: Ubuntu 16.04 Cloud version) with monitoring setup following steps has to be completed:

Clone the project from the repository

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

Go to service Dockerfile

```
cd n2sky-services/monitoring/prometheus
```

Build the Docker image

Dockerfile:

```

FROM prom/prometheus:v2.0.0-beta.2
COPY alert.rules /etc/prometheus/alert.rules
COPY test.rules /etc/prometheus/test.rules
COPY node.rules /etc/prometheus/node.rules
COPY ./prometheus.yml /etc/prometheus/prometheus.yml
```

Command:

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

Run the service Docker container

```
docker run -d -p 9090:9090 prometheus:1
```

In case if developer want to see full stack he needs to deploy node_exporter:

Clone the project from the repository

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

Go to service Dockerfile

```
cd n2sky-services/monitoring/node_exporter
```

Run the service Docker container and start Monitoring

```
docker build -t node_exporter . && \
docker run -d -p 9100:9100 node_exporter && \
cd ../prometheus && \
nohup ./prometheus > /dev/null 2>&1 &
```

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

```
  scrape_interval:     15s
  evaluation_interval: 15s
```

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

```
alerting:
  alertmanagers:
  - static_configs:
    - targets:
       - 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.

```
rule_files:
  - "alert.rules"
  - "node.rules"
  - "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

```
- job_name: 'node'
    scrape_interval: 5s
    target_groups:
-        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.

9.1.9. Setting Up Alert Management System

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

```
global:
  smtp_smarthost: 'localhost:25'
  smtp_from: 'alertmanager@example.org'
  smtp_auth_username: 'alertmanager'
  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.

```
templates:
-        '/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.

```
route:
group_by: ['alertname', 'cluster', 'service']
group_wait: 30s
group_interval: 5m
repeat_interval: 3h
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.

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

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

```
receivers:
- name: 'team-X-mails'
  email_configs:
  - 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 are multiple possibilities how to work with alerting rules. The query language allows to use an expression and as a result to check an attribute of time series.

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

Following notations should be considered be the creation of alerting rules:

- All queries starting with "ALERT" namespace. After it follows the 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 into a human language like this: "Send an 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 subsections: 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.

Clone the project from the repository

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

Go to service Dockerfile

```
cd n2sky-services/monitoring/alertmanager
```

Build the Docker image

Dockerfile:

```

FROM prom/alertmanager:v0.10.0
COPY alertmanager.yml /etc/alertmanager/config.yml
ENTRYPOINT [ "/bin/alertmanager" ]
CMD        [ "-config.file=/etc/alertmanager/config.yml", \
             "-storage.path=/alertmanager" ]
```

Command:

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

Run the service Docker container

```
docker run -d -p 9093:9093 alertmanager:1
```

9.2. API Documentation

The API documentation will consist of three parts:

- Short description
- The API endpoint
- Example call
- Response

9.2.1. N2Sky Monitoring System API Documentation

- ***Get monitoring data with period from chosen server***
  - *GET: /api/monitoring/:server/:query/:minus/:type/:step*
    Parameters:
    * :server - The server api or hostname, where monitoring is installed and running.

  * :query - query against monitoring data.
  * :minus - difference between current minus desired time.
  * :type - can be seconds, minutes, hours
  * :step - the step between difference time and current time

- *Example call:*

```
http://192.168.0.102:9595/api/monitoring/192.168.0.105
/http_request_duration_microseconds/3/hours/15m
```

- *Response:*

```
{
        "metric": {
            "__name__": "http_request_duration_microseconds",
            "handler": "static",
            "instance": "localhost:9090",
            "job": "prometheus",
            "quantile": "0.5"
        },
        "values": [
            [
                1519558760,
                "NaN"
            ]
            ...
```

- ***Get possible metrics***
  - *GET: /api/monitoring/metrics/:host*
    Parameters:
    * :host - The server api or hostname, where monitoring is installed and running.
  - *Example call:*

```
http://192.168.0.102:9595/api/monitoring/metrics/openstack
```

  - *Response:*

```
[
    "ALERTS",
    "go_gc_duration_seconds",
    "go_gc_duration_seconds_count",
    "go_gc_duration_seconds_sum",
    "go_goroutines",
    "go_info",
    "go_memstats_alloc_bytes",
    "go_memstats_alloc_bytes_total",
            ...
```

- ***Create new monitoring chart of the OpenStack or any instance and put it in chosen dashboard***
  - *POST: /api/user/dashboard/openstack*
    Body:
    * user
    * metric
    * delay
    * delaytype
    * step
    * steptype,
    * server
    * show
    * selectedServerId
    * selectedServerName
  - *Example call:*

```
http://192.168.0.102:9595/api/user/dashboard/openstack
```

  - *Response:* EMPTY
- ***Get user metrics config***
  - *GET: /user/dashboard/openstack/:userid/:show*
    Parameters:
    * :userid - the user unique id
    * :show - the server
  - *Example call:*

```
http://192.168.0.102:9595/api/user/dashboard/openstack/admin/openstack
```

  - *Response:*

```
[
    {
        "user": "admin",
        "metric": "node_load1",
        "delay": "15",
        "delaytype": "minutes",
        "step": "10s",
        "steptype": "s",
        "server": "192.168.0.105",
        "selectedServerId": "openstack",
        "selectedServerName": "openstack",
        "__v": 0,
        "show": [
            "all",
            "openstack"
        ]
    },
            ...
```

- ***Delete monitoring chart***
  - *DELETE: /user/dashboard/openstack/:id*
    Parameters:
    * :id - id of the monitoring chart
  - *Example call:*

```
http://192.168.0.102:9595/api/user/dashboard/openstack/1111
```

  - *Response: EMPTY*

9.2.2. Alerting Management System API Documentation

- ***Get Occurred Alerts***
  - *GET: /api/alerts*
  - *Example call:*

```
http://192.168.0.102:9595/api/alerts
```

  - *Response:*

```
[
    {
        "labels": {
            "alertname": "NodeExporterDown",
            "severity": "warning"
        },
        "annotations": {
            "description": "Prometheus could not scrape a node-exporter
            "summary": "node-exporter cannot be scraped"
        },
        "startsAt": "2018-02-10T18:24:30.532998966+01:00",
        "endsAt": "2018-02-25T17:54:00.514487286Z",
        "status": {
            "state": "active",
            "silencedBy": [],
            "inhibitedBy": []
        },
        "receivers": [
            "team-X-mails"
        ],
        "fingerprint": "2cf0947390157d74"
    },
            ...
```

9.2.3. OpenStack API Documentation

- ***Get OpenStack projects***
  - *GET: /api/projects*
  - *Example call:*

```
http://192.168.0.102:9595/api/projects
```

  - *Response:*

```
{
    "links": {
        "self": "http://192.168.0.105/identity/v3/auth/projects",
        "previous": null,
        "next": null
    },
    "projects": [
        {
            "is_domain": false,
            "description": "",
            "links": {
                "self": "http://192.168.0.105/identity/v3/projects/13f1443
            },
            "tags": [],
            "enabled": true,
            "domain_id": "default",
            "parent_id": "default",
            "id": "13f1443509fa496db6e4cad43208170f",
            "name": "demo"
        },
            ...
```

- ***Get OpenStack project by id***
  - *GET: /projects/:id*

    Parameters:
    - :id - project id
  - *Example call:*

```
http://192.168.0.102:9595/api/projects/1111
```

- *Response:*

```
        {
            "is_domain": false,
            "description": "",
            "links": {
                "self": "http://192.168.0.105/identity/v3/projects/13f1443
            },
            "tags": [],
            "enabled": true,
            "domain_id": "default",
            "parent_id": "default",
            "id": "13f1443509fa496db6e4cad43208170f",
            "name": "demo"
            ...
```

- ***Get OpenStack networks***
  - *GET: /api/networks*
  - *Example call:*

```
http://192.168.0.102:9595/api/networks
```

  - *Response:*

```
    "networks": [
        {
            "status": "ACTIVE",
            "router:external": true,
            "availability_zone_hints": [],
            "availability_zones": [
                "nova"
            ],
            "ipv4_address_scope": null,
            "description": "",
            "port_security_enabled": true,
            ...
```

- ***Get OpenStack extentions***
  - *GET: /api/extensions*
  - *Example call:*

```
http://192.168.0.102:9595/api/extensions
```

  - *Response:*

```
    "extensions": [
        {
            "alias": "default-subnetpools",
            "updated": "2016-02-18T18:00:00-00:00",
            "name": "Default Subnetpools",
            "links": [],
            "description": "Provides ability to mark and use a subnetpool
        },
            ...
```

- ***Get OpenStack subnetpools***
  - *GET: /api/subnetpools*
  - *Example call:*

```
http://192.168.0.102:9595/api/subnetpools
```

  - *Response:*

```
{
    "subnetpools": [
        {
            "prefixes": [
                "10.0.0.0/22"
            ],
            "description": "",
            "tags": [],
            "tenant_id": "cbc8b4a7b24143dfbe87c17a684b5375",
            "created_at": "2018-02-06T21:58:47Z",
            "default_quota": null,
            "updated_at": "2018-02-06T21:58:47Z",
            "name": "shared-default-subnetpool-v4",
            "is_default": true,
            "min_prefixlen": "8",
            "address_scope_id": null,
            "revision_number": 0,
            ...
```

- ***Get OpenStack service-providers***
  - *GET: /api/service-providers*
  - *Example call:*

```
http://192.168.0.102:9595/api/service-providers
```

  - *Response:*

```
    "service_providers": [
        {
            "service_type": "L3_ROUTER_NAT",
            "default": false,
            "name": "single_node"
        },
            ...
```

- ***Get OpenStack images***
  - *GET: /api/images*
  - *Example call:*

```
http://192.168.0.102:9595/api/images
```

  - *Response:*

```
{
    "images": [
        {
            "status": "active",
            "name": "cirros-0.3.5-x86_64-disk",
            "tags": [],
            "container_format": "bare",
            "created_at": "2018-02-06T21:58:11Z",
            ...
```

- ***Get OpenStack Download Image By Id***
  - *GET: /api/images/:id/download*
  - *Example call:*

```
http://192.168.0.102:9595/api/images/1111/download
```

  - *Response: bytes file*

- ***Get OpenStack VITRAGE templates***
  - *GET: /rca/template*
  - *Example call:*

```
http://192.168.0.102:9595/api/rca/template
```

  - *Response: List of templates*

- ***Get OpenStack VITRAGE resources***
  - *GET: /rca/resources*
  - *Example call:*

```
http://192.168.0.102:9595/api/rca/resources
```

  - *Response: List of resources*

- ***Get OpenStack flavors by project id***
  - *GET: /api/flavors/:id*
  - *Example call:*

```
http://192.168.0.102:9595/api/flavors/13f1443509fa496db6e4cad43208170f
```

– *Response:*

```
    "flavors": [
        {
            "name": "m1.tiny",
            "links": [
                {
                    "href": "http://192.168.0.105/compute/v2.1/flavors/1",
                    "rel": "self"
                },
                {
                    "href": "http://192.168.0.105/compute/flavors/1",
                    "rel": "bookmark"
                }
            ],
            "ram": 512,
            "OS-FLV-DISABLED:disabled": false,
            "vcpus": 1,
            "swap": "",
            "os-flavor-access:is_public": true,
            "rxtx_factor": 1,
            "OS-FLV-EXT-DATA:ephemeral": 0,
            "disk": 1,
            "id": "1"
        },
            ...
```

- ***Get OpenStack servers by project id***
  - *GET: /api/servers/:id*
  - *Example call:*

```
http://192.168.0.102:9595/api/servers/13f1443509fa496db6e4cad43208170f
```

– *Response:*

```
    "servers": [
        {
            "OS-EXT-STS:task_state": null,
            "addresses": {
                "private": [
                    {
                        "OS-EXT-IPS-MAC:mac_addr": "fa:16:3e:59:4c:31",
                        "version": 4,
                        "addr": "10.0.0.12",
                        "OS-EXT-IPS:type": "fixed"
                    },
                    {
                        "OS-EXT-IPS-MAC:mac_addr": "fa:16:3e:59:4c:31",
                        "version": 6,
                                "addr": "fd5a:ee2f:760e:0:f816:3eff:fe59:4c31",
                                "OS-EXT-IPS:type": "fixed"
                           }
                      ]
                 },
                 ...
```

9.2.4. User Management API Documentation

- ***Get all users***
  - *GET: /users*
  - *Example call:*

```
http://192.168.0.102:9091/api/users
```

  - *Response:*

```
[
    {
        "name": "test",
        "password": "5a7c27142f200a53dff94f37",
        "email": "test@test.com",
        "type": "user",
        "active": true,
        "__v": 0
    },
                ...
```

- ***Get user by id***
  - *GET: /user/:id*
  - *Example call:*

```
http://192.168.0.102:9091/api/user/1
```

  - *Response:*

```
[
    {
        "name": "admin",
        "password": "5a7c27142f200a53dff94f37",
        "type": "admin",
        "active": true,
    },
                ...
```

- ***Login into N2Sky System***

  - *POST: /user/login*
    BODY:
    * user id
    * password
  - *Example call:*

```
http://192.168.0.102:9091/user/login
```

  - *Response:*

```
[
    {
        "name": "admin",
        "token": "ad27823yh872y3d828238d872"
    },
                ...
```

- ***Register in N2Sky System***
  - *POST: /user/signup*
    BODY:
    * username
    * password
    * email
  - *Example call:*

```
http://192.168.0.102:9091/user/signup
```

  - *Response:*

```
[
    {
        "name": "admin",
        "token": "ad27823yh872y3d828238d872"
    },
                ...
```

- ***Delete user from N2Sky System***
  - *DELETE: /user/delete*
    BODY:
    * username
    * token
  - *Example call:*

```
http://192.168.0.102:9091/user/delete
```

  - *Response: EMPTY*

9.2.5. Model Management API Documentation

Due the large response, the example call will not be shown.

- ***Create user project***
  - *POST: /project/create*
    BODY:
    * createdBy
    * name
    * description
- ***Get user projects***
  - *POST: /projects/:from/:limit*
    Parameters:
    * from
    * limit - offset
- ***Get project by id***
  - *GET: /project/id*
    Parameters:
    * id - project id
- ***Add neural network to the project***
  - *POST: /add_nn_id/:id*
    Parameters:
    * id - project id
    * nnid - neural network id
- ***Delete neural network from the project***
  - *POST: /delete_nn_id/:id*
    Parameters:
    * id - project id
    * nnid - neural network id
- ***Add trained model to the project***
  - *POST: /add_model_id/:id*
    Parameters:
    * id - project id

    * modelid - model id

- ***Delete trained model from the project***
  - *POST: /delete_model_id/:id*
    Parameters:
    * id - project id
    * modelid - model id
- ***Create neural network from ViNNSL***
  - *POST: /vinnsl/description/create*
    Body: Generated ViNNSL Description
- ***Upload neural network in ViNNSL format***
  - *POST: /vinnsl/description/upload*
    Body: ViNNSL Description
- ***Get ViNNSL Description with filters***
  - *POST: /vinnsl/descriptions/:from/:limit*
    Body: filter paramteres
- ***Update ViNNSL Description***
  - *POST: /vinnsl/description/update/:id*
    Body: fields to update
- ***Test neural network***
  - *POST: /nn/test*
    Body:
    * vinnslDescriptionId
    * testing_data
- ***Get training logs***
  - *POST: /nn/logs*
    Body:
    * model id

10. Conclusion and Future Work

N2Sky is presented as a novel cloud-based virtual community platform, which allows sharing and exchanging of neural network knowledge and computing resources. Today, N2Sky is tightly coupled with new technologies, which aims to increase scalability, extensibility, portability and performance.

On the infrastructure and application level of this platform, is implemented the scalable micro services architecture. The cloud environment monitoring is implemented for the system administrator. Additionally, the customized alerting system helps to control the environment without constant monitoring by a human.

It has a modern, user-friendly interface and gives positive user experience. The N2Sky platform was tested on desktop PCs, mobile devices and tablets. Because the design is user-oriented, a step-by-step processing workflow of actions was easy for all types of users.

There are already packaged solutions located in the neural networks and models respository, which can be easily reused in own projects. The creation of an own neural network from existing paradigms is possible via the N2Sky user interface without deep knowledge in this field.

The new ViNNSL template format gave the contributors a new and simple way on how to describe and set all needed data about a neural network into one file. The ViNNSL template is dynamic and extendable, can be modified on one side and it will be reflected automatically across all the N2Sky services.

In the future, N2Sky can be extended from the neural network domain to an arbitrary machine learning. Hereby, the ViNNSL template extension and scalable microservices N2Sky architecture can be useful for future work.

A. ViNNSL Backpropagation Template

```
{
        "metadata": {
                "name": null,
                "description": null,
                "paradigm": "Backpropagation"
        },
        "creator": {
                "name": null,
                "contact": null
        },
        "executionEnvironment": {
                "isRunning": false,
                "isPublic": false,
                "hardware": null,
                "lastRun": null,
                "image": {
                        "imageType": null,
                        "details": null
                }
        },
        "problemDomain": {
                "propagationType": {
                        "value": null,
                        "possibleValues": ["feedforward"],
                        "learningType": {
                                "value": null,
                                "possibleValues": ["definedconstructed", "trained",
                        }
                },
                "applicationField": {
                        "value": null,
                        "possibleValues": ["AccFin", "HealthMed", "Marketing", "Ret
                },
                "problemType": {
                        "value": null,
                        "possibleValues": ["Classifiers", "Approximators", "Memory"
                },
                "networkType": "Backpropagation"
        },
        "endpoints": [{
                "name": "train",
                "endpoint": null
        }, {
                "name": "test",
                "endpoint": null
        }],
        "structure": {
                "inputLayer": {
                        "result": {
                                "nodesId": []
                        },
                        "config": {
                                "dimensions": {
                                        "min": 1,
                                        "max": 1
                                },
                                "size": {
```

```
                    "min": 960,
                    "max": 960
                }
            }
        },
        "hiddenLayer": {
            "result": {
                "dimensions": [{
                    "id": null,
                    "nodesId": []
                }]
            },
            "config": {
                "dimensions": {
                    "min": 1,
                    "max": 1
                },
                "size": {
                    "min": 960,
                    "max": 960
                }
            }
        },
        "outputLayer": {
            "result": {
                "nodesId": []
            },
            "config": {
                "dimensions": {
                    "min": 1,
                    "max": 1
                },
                "size": {
                    "min": 960,
                    "max": 960
                }
            }
        },
        "connections": {
            "fullyConnected": {
                "isConnected": null
            },
            "shortcuts": {
                "isConnected": null,
                "connections": [{
                    "from": null,
                    "to": null,
                    "isFullConnected": null
                }]
            }
        }
    },
    "parameters": {
        "input": [
            {
                "parameter": "learningrate",
                "defaultValue": "0.01",
                "possibleValues": [
                    "0.1",
                                        "0.2",
                                        "0.3",
                                        "0.4"
                                ]
                        },
                        {
                                "parameter": "biasInput",
                                "defaultValue": "1",
                                "possibleValues": ["1","2","3"]
                        },
                        {
                                "parameter": "biasHidden",
                                "defaultValue": "1",
                                "possibleValues": ["1","2","3"]
                        },
                        {
                                "parameter": "momentum",
                                "defaultValue": "0.9",
                                "possibleValues": ["0.1","0.2","0.3","0.4","0.5","0
                        },
                        {
                                "parameter": "activationFunction",
                                "defaultValue": "sigmoid",
                                "possibleValues": ["sigmoid", "relu", "softmax"]
                        },
                        {
                                "parameter": "activationFunctionHidden",
                                "defaultValue": "relu",
                                "possibleValues": ["sigmoid", "relu"]
                        },
                        {
                                "parameter": "threshold",
                                "defaultValue": "0.000001",
                                "possibleValues": ["0.00001","0.000001"]
                        },
                        {
                                "parameter": "target_data",
                                "defaultValue": "[0],[1],[1],[0]",
                                "possibleValues": []
                        },
                        {
                                "parameter": "epoche",
                                "defaultValue": "100",
                                "possibleValues": []
                        }
                ],
                "output": "[[0,0],[0,1],[1,0],[1,1]]"
        },
        "data": {
                "description": "[X] for each output",
                "tableDescription": "[X],[X]",
                "fileDescription": "TXT"
        }
}
```

B. Generated ViNNSL

```
{
        "_id": "5a900c6b6bb22b64b50327c2",
        "__v": 0,
        "executionEnvironment": {
                "lastRun": null,
                "isRunning": true,
                "hardware": null,
                "isPublic": true,
                "image": {
                        "imageType": null,
                        "_id": "5a900c6b6bb22b64b50327c3"
                }
        },
        "data": {
                "description": "[X] for each output",
                "tableDescription": "[X],[X]",
                "fileDescription": "TXT"
        },
        "parameters": {
                "output": "[[0,0],[0,1],[1,0],[1,1]]",
                "input": [
                        {
                                "parameter": "learningrate",
                                "defaultValue": "0.01",
                                "_id": "5a900c6b6bb22b64b50327cc",
                                "possibleValues": [
                                        "0.1",
                                        "0.2",
                                        "0.3",
                                        "0.4"
                                ]
                        },
                        {
                                "parameter": "biasInput",
                                "defaultValue": "1",
                                "_id": "5a900c6b6bb22b64b50327cb",
                                "possibleValues": [
                                        "1",
                                        "2",
                                        "3"
                                ]
                        },
                        {
                                "parameter": "biasHidden",
                                "defaultValue": "1",
                                "_id": "5a900c6b6bb22b64b50327ca",
                                "possibleValues": [
                                        "1",
                                        "2",
                                        "3"
                                ]
                        },
                        {
                                "parameter": "momentum",
                                "defaultValue": "0.9",
                                "_id": "5a900c6b6bb22b64b50327c9",
                                "possibleValues": [
                                        "0.1",
                                        "0.2",
                                        "0.3",
                                        "0.4",
                    "0.5",
                    "0.6",
                    "0.7",
                    "0.8",
                    "0.9"
                ]
            },
            {
                "parameter": "activationFunction",
                "defaultValue": "sigmoid",
                "_id": "5a900c6b6bb22b64b50327c8",
                "possibleValues": [
                    "sigmoid",
                    "relu",
                    "softmax"
                ]
            },
            {
                "parameter": "activationFunctionHidden",
                "defaultValue": "relu",
                "_id": "5a900c6b6bb22b64b50327c7",
                "possibleValues": [
                    "sigmoid",
                    "relu"
                ]
            },
            {
                "parameter": "threshold",
                "defaultValue": "0.000001",
                "_id": "5a900c6b6bb22b64b50327c6",
                "possibleValues": [
                    "0.00001",
                    "0.000001"
                ]
            },
            {
                "parameter": "target_data",
                "defaultValue": "[0],[1],[1],[0]",
                "_id": "5a900c6b6bb22b64b50327c5",
                "possibleValues": []
            },
            {
                "parameter": "epoche",
                "defaultValue": "100",
                "_id": "5a900c6b6bb22b64b50327c4",
                "possibleValues": []
            }
        ]
    },
    "connections": {
        "shortcuts": {
            "isConnected": true,
            "connections": [
                {
                    "from": "1-input",
                    "to": "1-hidden-layer",
                    "isFullConnected": false,
                    "_id": "5a900c6b6bb22b64b50327cd"
                }
            ]
        },
        "fullyConnected": {
            "isConnected": false
        }
    },
    "structure": {
        "inputLayer": {
            "amount": 3,
            "_id": "5a900c6b6bb22b64b50327d1",
            "nodesId": [
                "1-input",
                "2-input",
                "3-input"
            ]
        },
        "outputLayer": {
            "amount": 1,
            "_id": "5a900c6b6bb22b64b50327ce",
            "nodesId": [
                "1-output"
            ]
        },
        "hiddenLayer": [
            {
                "id": "1-hidden-layer",
                "amount": 4,
                "_id": "5a900c6b6bb22b64b50327d0",
                "nodesId": [
                    "1-node-1-hidden-layer",
                    "2-node-1-hidden-layer",
                    "3-node-1-hidden-layer",
                    "4-node-1-hidden-layer"
                ]
            },
            {
                "id": "2-hidden-layer",
                "amount": 4,
                "_id": "5a900c6b6bb22b64b50327cf",
                "nodesId": [
                    "1-node-2-hidden-layer",
                    "2-node-2-hidden-layer",
                    "3-node-2-hidden-layer",
                    "4-node-2-hidden-layer"
                ]
            }
        ]
    },
    "endpoints": [
        {
            "_id": "5a90124b6bb22b64b50327d8",
            "endpoint": "http://192.168.0.102:5000/train",
            "name": "train"
        },
        {
            "_id": "5a90124b6bb22b64b50327d7",
            "endpoint": "http://192.168.0.102:5000/test",
            "name": "test"
        }
        ],
        "problemDomain": {
                "problemType": "Classifiers",
                "networkType": "Backpropagation",
                "applicationField": [
                        "AccFin"
                ],
                "propagationType": {
                        "propType": "feedforward",
                        "learningType": "supervised"
                }
        },
        "creator": {
                "name": "admin",
                "contact": null
        },
        "metadata": {
                "name": "XOR Test",
                "description": "test",
                "paradigm": "Backpropagation",
                "version": {
                        "major": null,
                        "minor": null
                }
        }
}
```

C. ViNNSL Model

```

[
    {
        "isTrainingDone": true,
        "rawModel": "{// RAW MODEl //}",
        "vinnslDescriptionId": "5a908d0b5a051c64bf0af3aa",
        "trainedBy": "admin",
        "trainedOn": "2018-02-23T21:55:49.280Z",
        "isCopy": false,
        "copiedFromModelId": "",
        "training_data": "[[0,0],[0,1],[1,0],[1,1]]",
        "logs": [
            {
                "binary_accuracy": "0.75",
                "epoch": "0",
                "loss": "0.24829554557800293"
            },
            {
                "binary_accuracy": "0.75",
                "epoch": "1",
                "loss": "0.24829086661338806"
            },
            {
                "binary_accuracy": "0.75",
                "epoch": "2",
                "loss": "0.24828189611434937"
             }

              ..........


            {
                "binary_accuracy": "0.75",
                "epoch": "99",
                "loss": "0.23684203624725342"
            }
        ],
        "__v": 0,
        "tests": [
            {
                "_id": "5a9159865a051c64bf0af47e",
                "createdOn": "2018-02-24T12:24:38.198Z",
                "result": "[[0.]\n [1.]\n [1.]\n [1.]]",
                "testing_data": "[[0,0],[0,1],[1,0],[1,1]]",
                "user": "admin"
            }
        ],
        "endpoints": [
            {
                "_id": "5a908d335a051c64bf0af3bf",
                "endpoint": "http://192.168.0.102:5000/train",
                "name": "train"
            },
            {
                "_id": "5a908d335a051c64bf0af3be",
                "endpoint": "http://192.168.0.102:5000/test",
                "name": "test"
            }
        ],
        "parameters": {
            "input": [
                {
                    "parameter": "learningrate",
                    "value": "0.01",
                    "_id": "5a908de55a051c64bf0af3cc"
                },
                {
                    "parameter": "biasInput",
                    "value": "1",
                    "_id": "5a908de55a051c64bf0af3cb"
                },
                {
                    "parameter": "biasHidden",
                    "value": "1",
                    "_id": "5a908de55a051c64bf0af3ca"
                },
                {
                    "parameter": "momentum",
                    "value": "0.9",
                    "_id": "5a908de55a051c64bf0af3c9"
                },
                {
                    "parameter": "activationFunction",
                    "value": "sigmoid",
                    "_id": "5a908de55a051c64bf0af3c8"
                },
                {
                    "parameter": "activationFunctionHidden",
                    "value": "relu",
                        "_id": "5a908de55a051c64bf0af3c7"
                    },
                    {
                        "parameter": "threshold",
                        "value": "0.000001",
                        "_id": "5a908de55a051c64bf0af3c6"
                    },
                    {
                        "parameter": "target_data",
                        "value": "[0],[1],[1],[0]",
                        "_id": "5a908de55a051c64bf0af3c5"
                    },
                    {
                        "parameter": "epoche",
                        "value": "100",
                        "_id": "5a908de55a051c64bf0af3c4"
                    }
                ]
            }
        }
```

D. Custom Image with the Monitoring System

```

sudo apt-get update && \
sudo apt-get install --assume-yes apt-transport-https ca-certificates \
curl gnupg2 software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg \
 | sudo apt-key add - && \
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; \
   echo "$ID") $(lsb_release -cs) stable" && \
sudo apt-get update && \
sudo apt-get install --assume-yes docker-ce && \
sudo usermod -a -G docker $USER && \
sudo bash && \
git clone https://github.com/latyaodessa/n2sky-services.git && \
cd n2sky-services/monitoring/node_exporter && \
docker build -t node_exporter . && \
docker run -d -p 9100:9100 node_exporter && \
cd ../prometheus && \
nohup ./prometheus > /dev/null 2>&1 &
```

List of Figures

1. Current N2Sky architecure 16
2. Current N2Sky User Interface 18
3. Requirement Specification Qualities 23
4. User roles hierarchy and modules where they are operating on (marked grey) 24
5. Novel N2Sky Architecture 27
6. Microservices approach in frontend application 29
7. N2Sky frontend application and its services 30
8. The sample workflow 32
9. N2Sky Fundamental Layout 38
10. N2Sky Main Layout 40
11. N2Sky Mobile Layout 41
12. N2Sky Dashboard template 45
13. Difference between sans-serif font Arial (A) and the browser serif font Times New Roman (B) 46
14. Average screen resolution of mobile, tablet and desktop devices on 2017 46
15. Customised N2Sky accordion UI element 48
16. Customised N2Sky button UI element 49
17. Customised N2Sky button UI element animation 49
18. Customised N2Sky logo in SVG format 49
19. Customised N2Sky warning message UI element 50
20. Customised N2Sky informational message UI element 50
21. Customised N2Sky navigation UI element 51
22. Customised N2Sky navigation bar with a table 51
23. Responsive N2Sky Grid Item UI comonent 52
24. N2Sky Main Navigation Menu for arbitrary user 52
25. N2Sky Main Navigation Menu for system administrator 53
26. N2Sky Modal Window UI component 54
27. N2Sky User Management Web Service 55
28. N2Sky Model Repository Web Service 57
29. N2Sky Cloud Management Web Service 59
30. Prometheus monitoring architecture 62
31. Metric response 64
32. N2Sky Monitoring Dashlet 67
33. Alerting System Architecture 68
34. Communication within Alerting Management System 69
35. N2Sky Alerting Management System Event representation. 71
36. N2Sky Alerting Management System Severity Level 72
37. N2Sky Administration Dashboard. 84
38. N2Sky Administration Dashboard. Administration tools component. . . 84
39. N2Sky Administration Dashboard. Featured Metrics component. 85
40. N2Sky OpenStack Dashboard. 86
41. N2Sky OpenStack Dashboard. Projects grid 86
42. N2Sky OpenStack Dashboard. Project view 88
43. N2Sky OpenStack Dashboard. Server overview 88
44. N2Sky OpenStack Dashboard. Flavors 89
45. N2Sky OpenStack Dashboard. Server details view. 90

46. N2Sky OpenStack Dashboard. Neutron Service View 91
47. N2Sky OpenStack Dashboard. Images Service View 93
48. N2Sky OpenStack Dashboard. Vitrage Service View 95
49. N2Sky OpenStack Dashboard. Template Details View 96
50. N2Sky Administration Dashboard. Dashboard Settings 97
51. N2Sky Administration Dashboard. Monitoring charts creation modal popup 98
52. N2Sky Administration Dashboard. Monitoring charts representation . . 100
53. N2Sky Administration Dashboard. Alerting rule creation 101
54. N2Sky Administration Dashboard. Alerts representation 102
55. N2Sky User Dashboard 104
56. N2Sky Main Module. Available user tools 105
57. N2Sky Main Module. Create New Project Popup 105
58. N2Sky Main Module. User projects grid 106
59. N2Sky Main Module. Neural networks filtering bar. 106
60. N2Sky User Project Dashboard 108
61. N2Sky Project Dashboard. Available tools 109
62. N2Sky Project Dashboard.Upload own network in ViNNSL format . . . 109
63. N2Sky Project Dashboard. Saved trained models 111
64. Three Steps View. The Neural Network Description 112
65. Three Steps View. The Neural Network Structure with full connection . 113
66. Three Steps View. The shortcut connections 114
67. Three Steps View. The Neural Network Training. 115
68. Three Steps View. The Neural Network Training. Run instance 115
69. Three Steps View. The Neural Network Training. Training form 116
70. Three Steps View. Training Results and the Model Evaluation 118
71. Three Steps View. Training Results and the Model Evaluation. Copy model into own project. 118
72. Three Steps View. Training Results and the Model Evaluation. The model testing. 119
73. N2Sky Neural network repository 120
74. N2Sky Neural network repository. Copy the neural network. 121
75. N2Sky Models Repository 122
76. N2Sky Models Repository. Short Description 123
77. Tutorial. Activity Diagram of using Network and Model Repositories . 125
78. Tutorial. Login form on the left sight and the signup form on the right. 126
79. Tutorial. Activity Diagram of creation the neural network from the existing paradigm 129
80. Tutorial. Activity Diagram of upload own neural network paradigm . . 131
81. Use Case of typical arbitrary user behaviour 133
82. Use Case. Developing the neural network with N2Sky. 135

Listings

1. React Router . . . 41
2. SVG example . . . 49
3. Prometheus query . . . 65
4. Prometheus short query . . . 65
5. Prometheus alerting rule . . . 66
6. Querying alert . . . 69
7. Alert error response . . . 70
8. ViNNSL template metadata . . . 74
9. ViNNSL template environment . . . 75
10. ViNNSL template structure . . . 76
11. ViNNSL template connections . . . 77
12. ViNNSL template input parameters . . . 78
13. Generated ViNNSL model . . . 79
14. ViNNSL generated layers . . . 79
15. ViNNSL generated hidden layers . . . 80
16. ViNNSL trained model testing . . . 81

Bibliography

[1] Adamenko, A., Schikuta, E., and Fedorenko, A. Towards a modularized cloud container-based problem solving environment. In *IEEE SCC2018, The 2018 International Joint Conference on* (2018), IEEE, pp. 1–8.

[2] Akbar, S., Peikari, M., Salama, S., Nofech-Mozes, S., and Martel, A. Transitioning between convolutional and fully connected layers in neural networks. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support* (Cham, 2017), M. J. Cardoso, T. Arbel, G. Carneiro, T. Syeda-Mahmood, J. M. R. Tavares, M. Moradi, A. Bradley, H. Greenspan, J. P. Papa, A. Madabhushi, J. C. Nascimento, J. S. Cardoso, V. Belagiannis, and Z. Lu, Eds., Springer International Publishing, pp. 143–150.

[3] 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.

[4] Beran, P. P., Vinek, E., Schikuta, E., and Weishaupl, T. Vinnsl - the Vienna neural network specification language. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on* (2008), IEEE, pp. 1872–1879.

[5] Borkowski, M., Fdhila, W., Nardelli, M., Rinderle-Ma, S., and Schulte, S. Event-based failure prediction in distributed business processes. *Information Systems* (2018).

[6] Cagle, K. *The Foundations of SVG.* Springer London, London, 2005, pp. 21–62.

[7] Denton, J. *Learning OpenStack Networking (Neutron).* Packt Publishing, 2014.

[8] e-Science, U. UK e-science programme. [online], `http://www.escience-grid.org.uk`, last visited January 2018, 2016.

[9] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] 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.

[11] Hassenzahl, M., and Tractinsky, N. User experience - a research agenda. *Behaviour & Information Technology 25*, 2 (2006), 91–97.

[12] Huqqani, A. A., Li, X., Beran, P. P., and Schikuta, E. N2cloud: Cloud based neural network simulation application. In *Neural Networks (IJCNN), The 2010 International Joint Conference on* (2010), IEEE, pp. 1–5.

[13] Karpathy, A. Deep reinforcement learning: Pong from pixels. [online], `http://karpathy.github.io/2016/05/31/rl/`, last visited May 2016, 2017.

[14] Khalil, W. Reference architecture for virtual organization. *PhD thesis, University of Vienna 2012* (2012).

[15] Lee, C. A., and Sill, A. F. A design space for dynamic service level agreements in openstack. *Journal of Cloud Computing 3*, 1 (Nov 2014), 17.

[16] 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.

[17] 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.

[18] Markelov, A. *OpenStack Compute.* Apress, Berkeley, CA, 2016, pp. 65–86.

[19] Martin, R. C. Clean code: A handbook of agile software craftsmanship. In *Clean Code: A Handbook of Agile Software Craftsmanship* (2008), Prentice Hall.

[20] 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.

[21] Matt T. Proud, J. V. Overview. what is prometheus? — prometheus, 2017. [Online; accessed 2017].

[22] Matt T. Proud, J. V. Sending alerts — prometheus, 2017. [Online; accessed 2017].

[23] Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal 2014*, 239 (2014), 2.

[24] Morgan, D. How mobile web traffic has affected screen resolution — spindogs, 2017. [Online; accessed 21-September-2017].

[25] Paternò, F., Santoro, C., and Scorcia, 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.

[26] 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.

[27] Regula Stopper, R. S. Graphical User Interface layout and design, 2012.

[28] Schikuta, E., Huqqani, A., and Kopica, T. Semantic extensions to the Vienna Neural Network Specification Language. In *Neural Networks (IJCNN), 2015 International Joint Conference on* (2015), IEEE, pp. 1–8.

[29] Schikuta, E., and Mann, E. N2sky - neural networks as services in the clouds. In *Neural Networks (IJCNN), The 2013 International Joint Conference on* (2013), IEEE, pp. 1–8.

[30] Schikuta, E., and Weishaupl, T. N2grid: neural networks in the grid. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on* (2004), vol. 2, IEEE, pp. 1409–1414.

[31] Spolsky, J. Things you should never do, part i. [online], `https://www.joelonsoftware.com/2000/04/06/things-you-should-never-do-part-i/`, last visited April 2000, 2000.

[32] 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.

[33] Technologies, G. Clodify. [online], `http://docs.getcloudify.org/3.4.1/intro/what-is-cloudify/`, last visited January 2018, 2017.

[34] Tom Gross, Jan Gulliksen, P. K. Human-computer interaction. In *Human-Computer Interaction – INTERACT* (2009), IFIP International Federation for Information Processing 2009.

[35] 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.

[36] Walraven, S., Truyen, E., and Joosen, W. Comparing paas offerings in light of saas development. *Computing 96*, 8 (Aug 2014), 669–724.

[37] Weik, M. H. *functional specification.* Springer US, Boston, MA, 2001, pp. 664–664.

[38] Weik, M. H. *typeface.* Springer US, Boston, MA, 2001, pp. 1850–1850.

[39] 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.

[40] Wu, H., Zhang, J., and Zong, C. Shortcut sequence tagging. In *Natural Language Processing and Chinese Computing* (Cham, 2018), X. Huang, J. Jiang, D. Zhao, Y. Feng, and Y. Hong, Eds., Springer International Publishing, pp. 196–207.

[41] 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.

[42] Zurada, J. M., Mazurowski, M. A., Ragade, R., Abdullin, A., Wojtudiak, J., and Gentle, J. Building virtual community in computational intelligence and machine learning [research frontier]. *IEEE Computational Intelligence Magazine 4*, 1 (2009), 43–54.