universität wien

MASTERARBEIT

Titel der Masterarbeit

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

Angestrebter akademischer Grad
Master of Science

Wien, 2018

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

Zusammenfassung

Abstract

Contents

1 Introduction

Random citation [Mus09] embeddeed in text.

1.1 Motivation

1.2 Terms and definitions

1.3 Related work

2 N2Sky Architecture

2.1 Current Architecture Analysis

2.1.1 Architecture design

2.1.2 Components

2.1.3 Current User Interface

2.1.4 Usability and user experience

2.2 Redesign motivation

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

2.2.1 Redesign Process

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

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

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

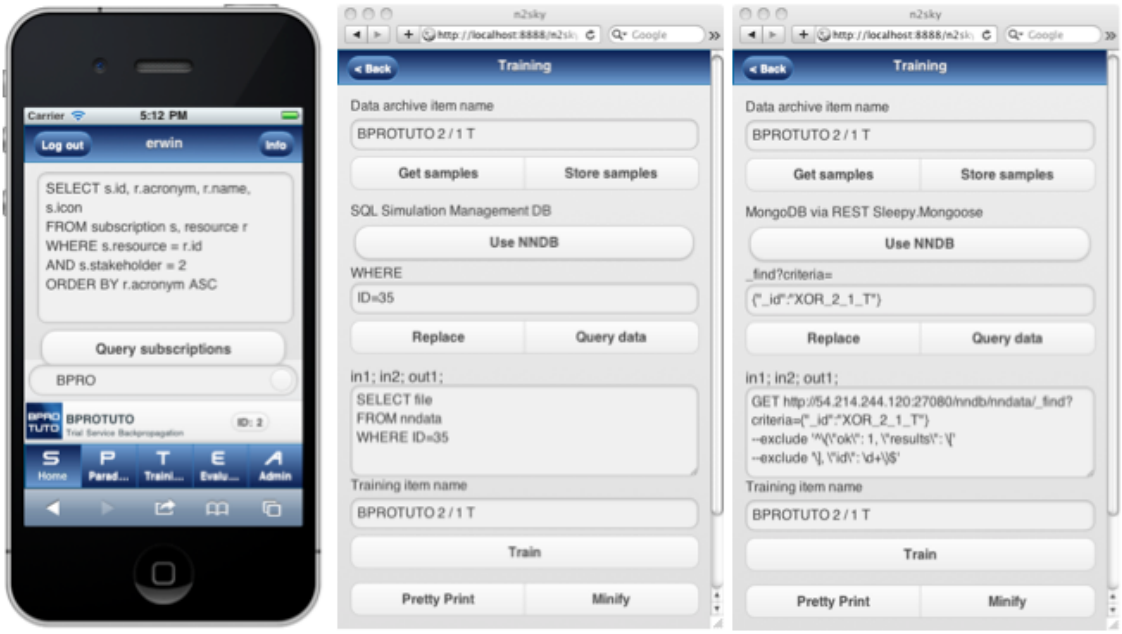

Figure 1: Current N2Sky User Interface

file, user can not download it. The last point is a design in general that does not look up-to-date and user attractive.

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

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

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

- A1: N2Sky gives possibility to test the own neural network. Unfortunately, if the user does not have a neural network it is impossible to test the application.

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

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

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

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

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

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

2.2.2 Refactoring the User Interface

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

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

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

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

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

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

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

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

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

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

Help and documentation. N2Sky will have tutorials that describe the user workflow. The expert users will also get a API documentation with a detailed description and sample requests.

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

TODO // CHECK MERSI DOKU

2.2.3 Introducing a new User Experience Design

TODO

2.3 Contemporary N2Sky Architecture

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 a users. N2Sky was moved from a complex monolithic system to an easy understandable application. Every interested user without having a deep knowledge in the neural network field can freely use N2Sky. The goal was to save and gain the current functionality of the application and decrease the visual complexity of it.

2.3.1 Architecture design

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

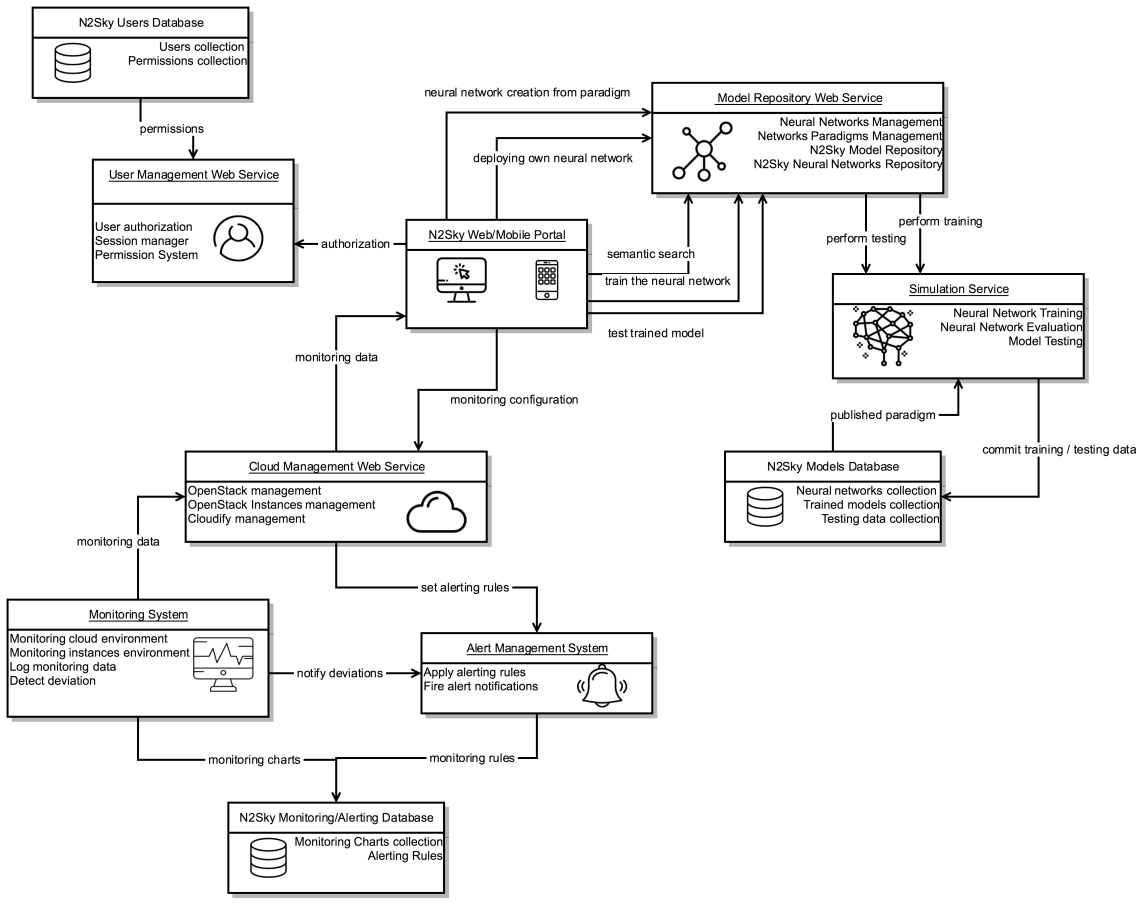

Figure 2: Contemporary N2Sky architecture overview

The architecture overview, which shown in “Fig. 2” represents microservices architecture.

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

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

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

2.3.2 N2Sky Services

N2Sky implements microservices architecture. It has 3 main web services as it shown in “Fig. 2” :

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

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

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

User Management Web Service. This web service responsible for permissions and user management and it has its own database. User can authorise in the system and get a session token. Every token is an unique collection of numbers and latin

letters. Token is assign to the authorised user and will be save until user is active. If user is not active in the next 3 hours, the session token will disappear. If user still have an active browser session, the authorisation token will be revalidate. If user trying to make some illegal request or behave too active, the authorisation token will be revoked and the system administrator will be notified about this incident.

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

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

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

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

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

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

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

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

If user perform training on his own neural network he can also log result about his network and enviroment 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 agains trained neural network model. Since there is absolute freedom by neural network structure customisation, the testing process can be inefficient and could take some resources from the environment. Considering this face it was decided to encapsulate this process too.

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

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

2.3.3 Modular frontend application design

The central concept of the application is to support the Software as a Service (SaaS) and Platform as a Service (PaaS) distributions. N2Sky consist of two modules: administration module, main application module as it shown in “Fig. 3”.

Redirector. When the user goes to N2Sky web portal, first he will be dealing with a redirector. The redirector is a small service, which is build on Nginix server. The purpose of it is to redirect the user depending on URL path.

/cloud redirects to "Administration module"

/n2sky redirects to "Main application module"

Administration module The administration module allows the system administrator to control the environment. The module supports OpenStack and Cloudify monitoring. Managing is possible through the application dashboard. It also contains custom monitoring and an alerting management system, which can be installed on any server within the N2Sky user interface. The administration module implements PaaS. It is fully configurable and wrapped into the open source project in order to make the module accessible to the third-party applications.

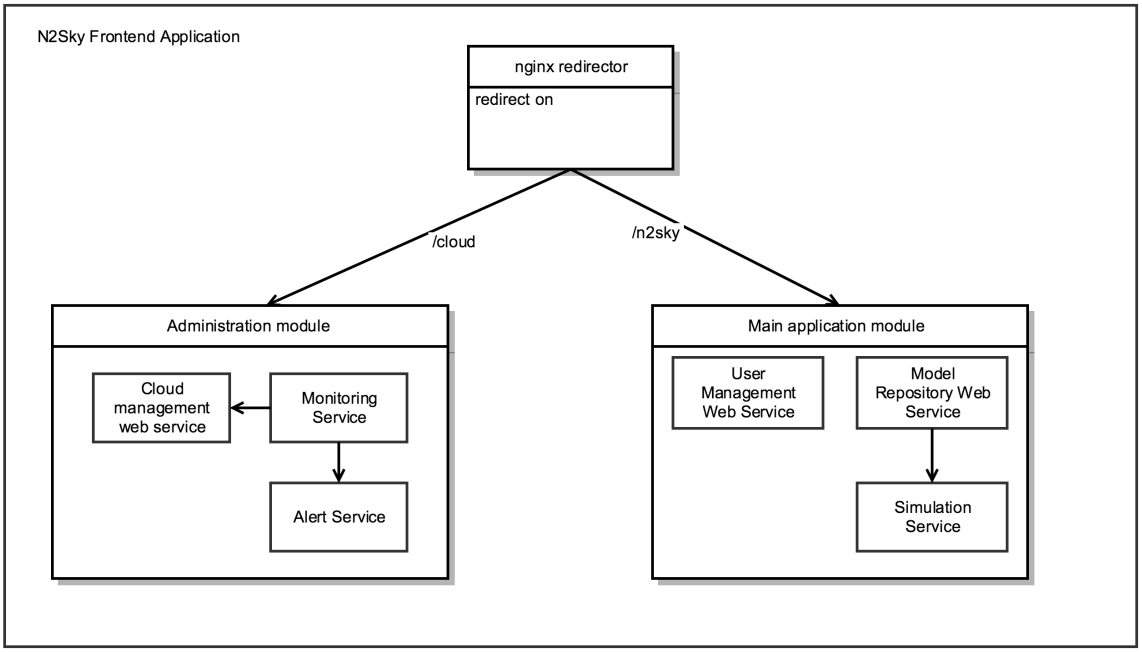

Figure 3: N2Sky frontend application and its services

Main application module The main application module is the central module of N2Sky. Within this module, users can use, train and test existing neural networks. It is possible to reuse the neural network paradigms and create own neural network. N2Sky allows to deploy own network and store data in cloud. Module services are supporting the SaaS distribution. Experts can use an application directly through the N2Sky API or they can integrate N2Sky services to their own application.

2.3.4 Technology Stack

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

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

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

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

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

ReactJS. JavaScript framework, which supports JSX programming language.

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

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

AngularJS. JavaScript framework, which supports TypeScript programming language.

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

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

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

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

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

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

Management System, which is a part of N2Sky, is also using Prometheus API in order to detect deviation and notify on event occurred.

As a database it was decided to choose NoSQL one. There were two NoSQL databases under consideration ElasticSearch and MongoDB. ElasticSearch supports indexes. It is possible to configure index so, that is impossible to insert something which is not mapped by the index. MongoDB does not user index approach, but MongoDB client supports schema. It was decided to use MongoDB schema and mapped 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 if the while system will go down it is possible to restore every service with a Jenkins Profiles.

3 N2Sky Components

3.1 User Interface Components

3.1.1 Components Overview

3.1.2 User Interface Elements

3.2 N2Sky Services

3.3 Continuous Monitoring System

TODO

3.3.1 Monitoring requirements

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

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

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

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

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

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

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

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

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

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

3.3.2 Applying Monitoring

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

- Proper and self-describable metric name with a key pairs
- Possibility to query metrics and join them in one graph.
- Not resource-intensive

- Support HTTP/HTTPS protocol
- Collect and push to repository time series
- Scalable

After researching it was decided to go for Prometheus Monitoring Toolkit. [1] This tool support all requested requirements. One of most interesting feature is that Prometheus can be used on any UNIX environment. Since in Openstack multiple instance with a different operational system can be created, Prometheus will match exact our needs. Originally Prometheus was created by SoundCloud team in 2012. [2] The core of monitoring application is Prometheus server, which collects time series data from the moment it was executed on environment as it shown in "Fig. 4". All components are written on Go [3] language and support multiple modules for monitoring different environment metrics. To understand the nature of Prometheus it is necessary to explain its architecture.

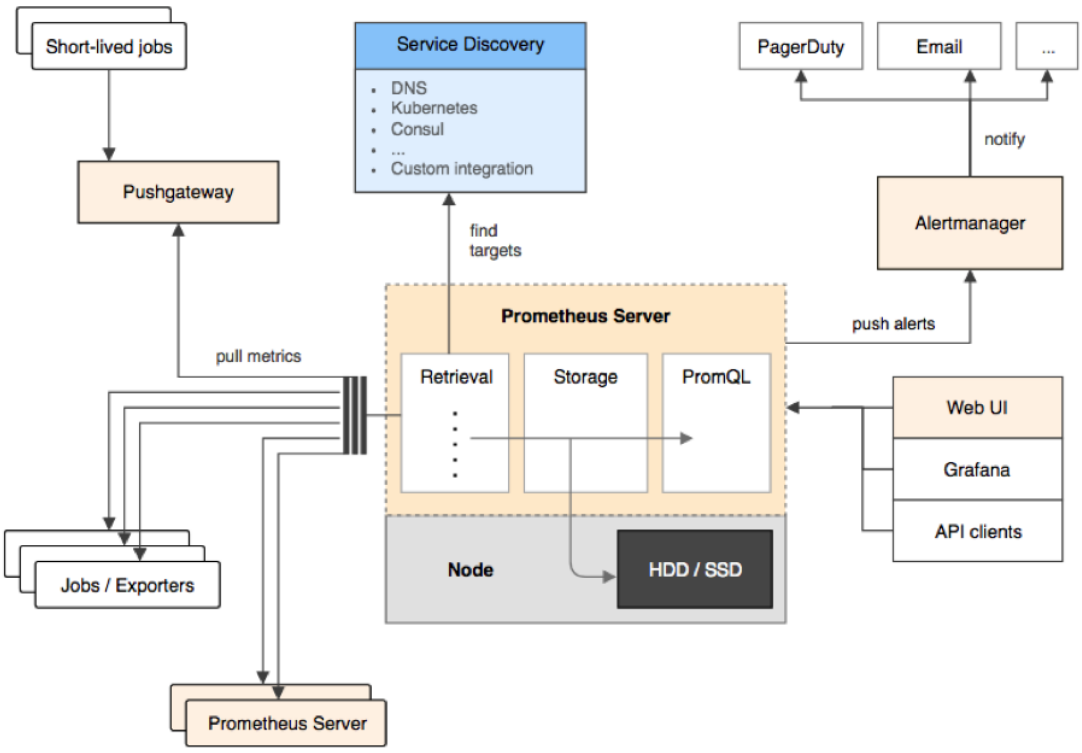

Figure 4: Prometheus monitoring architecture

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

The Prometheus server. This is the base element in the whole architecture. The server include services which collection, storing and retrieving nodes. The principle is scrapping or pulling. It means that the data fetched with some interval, which can me configured and stored accordantly as a time series. Prometheus support different modules, each module represents some node. The nodes expose these ports that Prometheus uses for retrieving the data. For example in N2Sky we are using Node Exporter Module which gives possibility to collect almost all essential data like CPU, RAM, HDD/SSD etc.

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

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

Metric notation Following example represent metric notation:

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

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

After executing this query the data will be retrieved from logs and represent as a time series as it shown on “Fig. 5”.

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

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

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

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 5: Metric response

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

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

3.3.3 Integration with N2Sky

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

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

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

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

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

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

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

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

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

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

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

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

3.3.4 Monitoring dashlet Design

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

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

Openstack Instances. Virtual machines with a different OS.

Docker Containers. Virtual machines in the Openstack instances.

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

3.4 Alerting Management System

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

3.4.1 Alerting System Architecture

Since the Alert Manager is a part of Prometheus Tool it has its own binary. The idea behind is to have only one Alert Manager and have monitoring tool on multiple machines. If machine goes down or even Prometheus itself, the Alert Manager can catch and deliver this event. To understand how Alert Manager works it is essential to understand the architecture of the whole system “Fig. 6”.

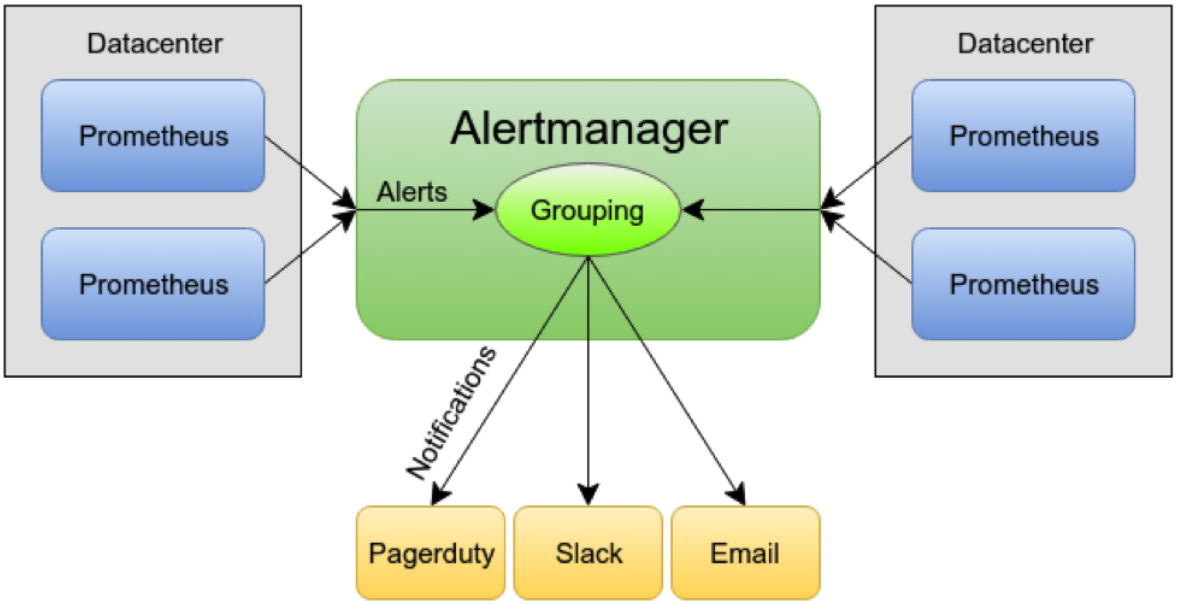

Figure 6: Alerting System Architecture

This architecture is a typical messaging platform. Messaging Service send messages between multiple clients. It is implement Producer-Consumer Pattern. In the Alert Manager Architecture the role of producer is taken by Prometheus Datacenter. Alert Manager consumes the messages. Consumer knows nothing about producer and just subscribe on event. With this approach it is possible to attach multiple producers. [Strategies for Integrating Messaging and Distributed Object Transactions https://link.springer.com/chapter/10.1007/3-540-45559-0_16]

Alerts can be collected in groups by datacenter. It means if an event occurs on multiple machines it can be packed into one notification and fired accordingly. In Prometheus configuration need to be setup only two things: reference on Alert Manager and Alert Rules. When Alert Manager consume an event it just dispatch it via notification “Fig. 7”..

3.4.2 Alert Rules

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

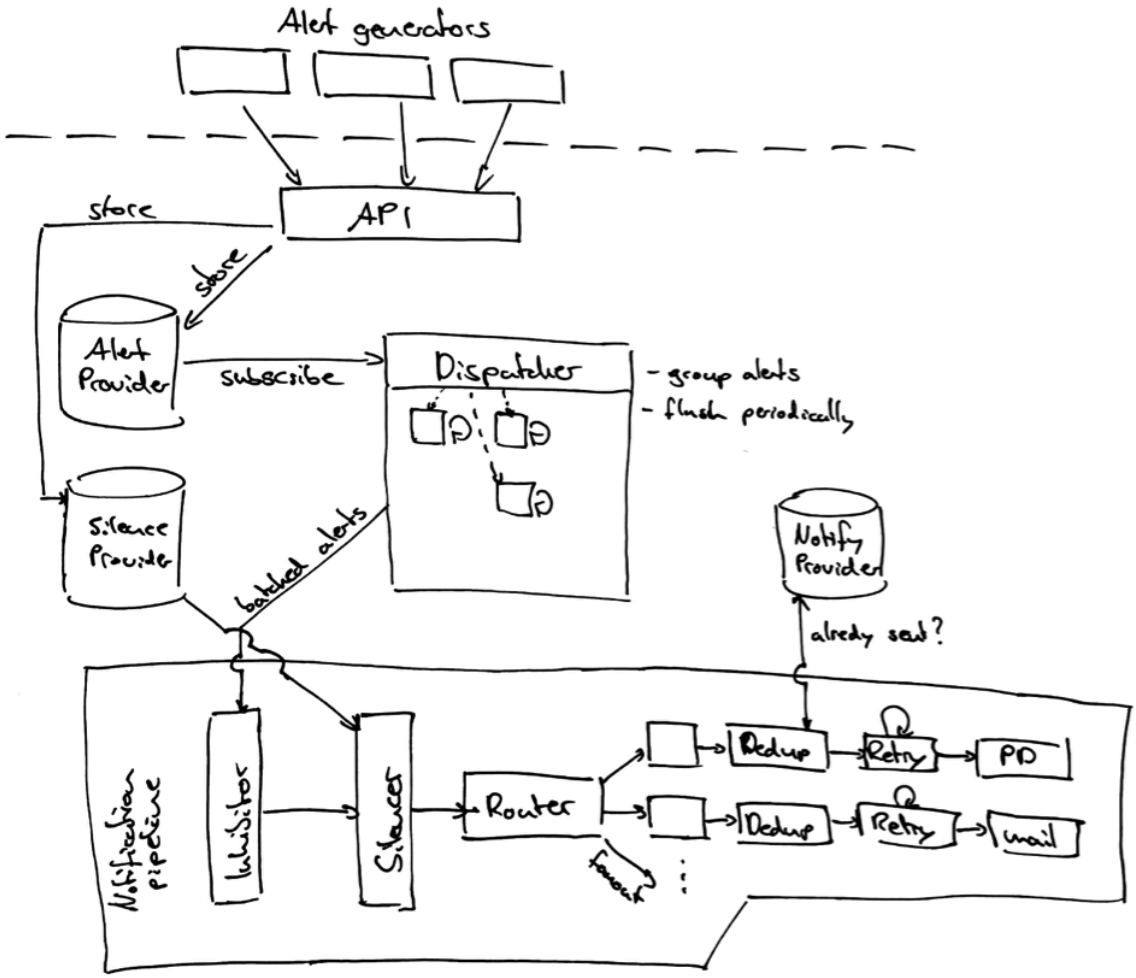

Figure 7: Communication within Alerting Management System

for every monitoring system on every machine. Alerting rules are instructions to the monitoring system it can be user as well for alerting as for recording. Recording rules allows to pre-compute frequently needed expressions or expressions, which are resource or time consuming. These rules are saving result in a new set of time series. It is like indexing this data, so that prevent expansive I/O methods. The rules are being executed sequentially with a predefined interval. With a alerting rules it is possible to define alert namely deviation by particular expression from Prometheus Tool and its exports (modules). It allows building an alert even on combined query. In case if Alert Manager is not available all alerts are saving into buffer. As soon Alert Manager online all events will be fired sequentially.

3.4.3 Integration with N2Sky

Alerting System is represented as a module in N2Sky. Alerting Client is the additional configuration upon Prometheus Monitoring System. When Prometheus will be executed it should have reference on Alerting System and its rules. Since the client should be installed on each OpenStack instance it was integrated into image snapshot. When the new instance will be spawned with a OpenStack snapshot, the client will be automatically executed there. Alerting Client fire alerts depending on configured rules. The rules can be created via user interface. Every alert has its severity level. Depending on it the fired events will be represented differently: ? Saverity

TODO

3.4.4 Alerting System Design

TODO

4 Functional requirements specification

4.1 User Roles

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

Every user has some specific area within he works. For example just registered user does not need to know the current environment monitoring information. These restrictions were motivation to create some user roles in order to restrict of grand some functionality of N2Sky as it shown on “Fig. 8”.

Contributor. The Contributor is an arbitrary user. Such a user has no necessity in having deep knowledge of the neural network field or know any programming language. The main goal of arbitrary user is to study neural networks within

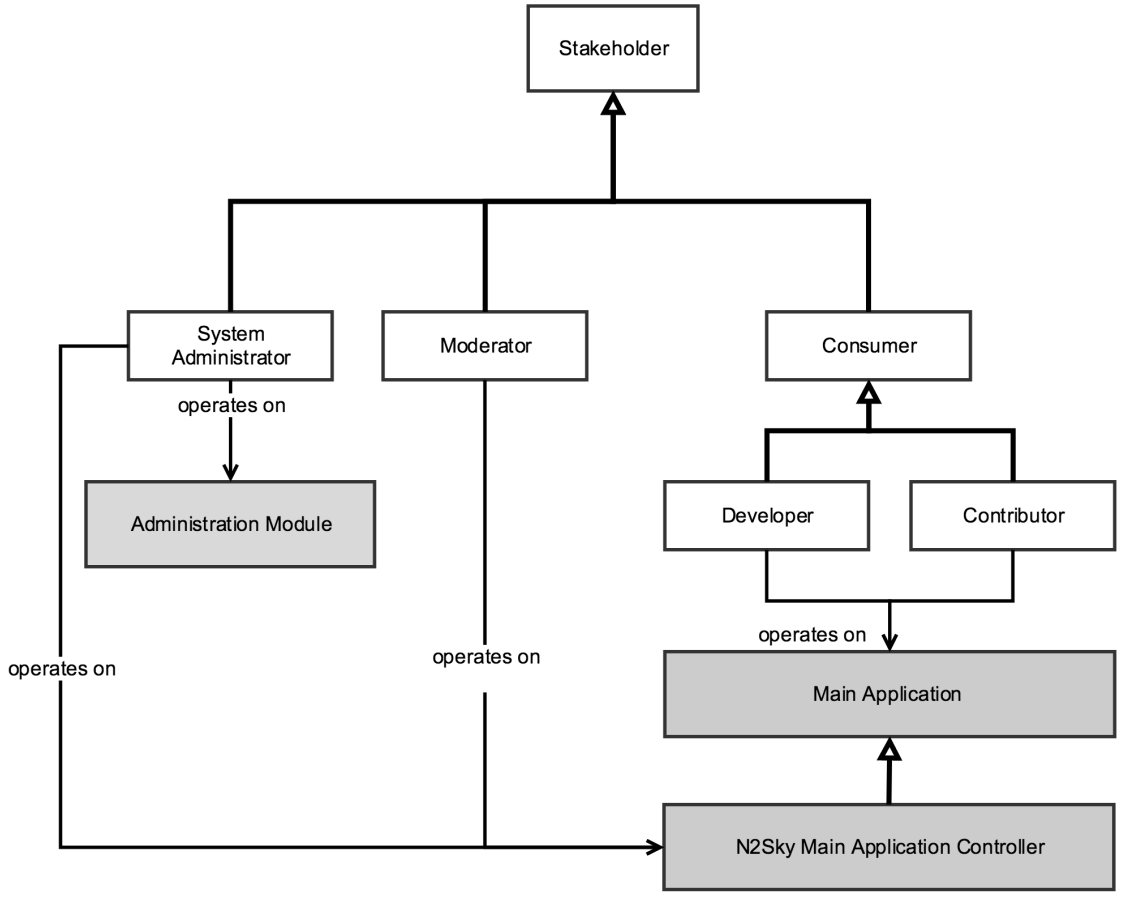

Figure 8: User roles hierarchy and modules where they operating on (marked grey)

N2Sky. The contributor has access only to his own dashboard and public available resources on the main application module. He can perform semantic search for available neural network paradigms and use them. He can also train running neural network instances and test them. This user can share his trained neural network by making it public.

Consumer. The Consumer is an arbitrary user, except he is not creating but using already existing neural networks and trained models. His main purpose is to evaluate 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 behaviour of neural networks. The Consumer can be converted to Contributor user if he has enough knowledge for it.

Developer. The Developer 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 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 developer is the study how his networks will behave with different network structures, input parameters and training data that is provided by other users.

System Administrator. System Administrator is a user who has a full access to application including environment management, monitoring and alerting features. Administrator can manage Openstack and Cloudify instances. He also can shadow any N2Sky user to observe the application from shadowed user perspective. Administrator has access to all dashboards in every module.

Moderator. Is a user with a granted permissions. This user can moderate Main Application Module namely neural network instances and trained models from other users. The Moderator does not have access to Administration Module and can not be converted to System Administrator user role.

4.2 User Main Functions

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

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

User Role	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

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

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

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

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

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

4.3 Administration Module Components

4.3.1 Affected users

4.3.2 Administration Dashboard

4.3.3 Openstack Dashboard

4.3.4 Cloudify Dashboard

4.3.5 Alerting System

4.3.6 Monitoring System

4.4 N2Sky Main Application Module Components

4.4.1 Affected user groups

4.4.2 N2Sky Dashboar

4.4.3 Neural Networks Repository

4.4.4 Models Repository

5 Tutorial

6 User Cases

7 Developer Guide

7.1 System configuration

7.1.1 Setting up the Monitoring System

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

```
1  scrape_interval:     15s
2  evaluation_interval: 15s
```

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

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

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

```
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. TODO

7.1.2 Setting up Alert Management System

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

```
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 is multiple possibilities how work with a alerting rules. The query language allows to use an expression and as a result to check an attribute of time series.

```
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 creation of alerting rules:

- All queries staring with "ALERT" namespace. After it follows name of alert in this case it is "HighLatency".
- "IF" is a condition "api_http_request_latencies_second", which based on Prometheus Tool expression. Set of time series with this expression has one parameter it is "quantile". Reading condition as a whole can be translated in a human language like this: "Send a alert if latency request per second bigger then 0.7".
- "FOR" it is period of time how often this condition should be checked.
- "LABELS" shows a severity level. There are 3 types of severity:
  - Critical
  - Warning
  - Page
- Every severity level can be defined on developer needs.

- "ANNOTATIONS" shows a readable for human comments. There are two sub sections: summary, which shows a short description of the event and description where detailed information about deviation can be written

For deploying alert manager Docker containers technology is used. TODO

7.2 Continuous integration

7.3 API Documentation

7.3.1 N2Sky Monitoring System API Documentation

Literaturverzeichnis

[Mus09] Mustermann, Max: *Ein Beispielbuch.* `http://www.example.com`. Version: 11 2009

Anhang