universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Arbeit / Title of the Master's Thesis

Container Based Execution Stack for Neural Networks

verfasst von / submitted by

Benjamin Nussbaum, BSc

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

Erklärung / Declaration

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Ort, Datum
Date

Unterschrift
Signature

Abstract / Zusammenfassung

Abstract

This thesis presents an execution stack for neural networks using the Kubernetes container orchestration and a Java based microservice architecture, which is exposed to users and other systems via RESTful webservices. The whole workflow including importing, training and evaluating a neural network model, becomes possible by using this service oriented approach. This work is influenced by N2Sky, a framework for the exchange of neural network specific knowledge and aims to support ViNNSL, the Vienna Neural Network Specification Language. The execution stack runs on many common cloud platforms. Furthermore it is scalable and each component is extensible and interchangeable.

Zusammenfassung

Diese Masterarbeit beschreibt einen Ausführungs-Stack für neuronale Netze, der unter Verwendung der Kubernes Container-Orchestrierung und einer Java basierten Microservice-Architektur, für Benutzer und Systeme via RESTful Webservices zugänglich gemacht wird. Der gesamte Arbeitsfluss, der Import, Training und Auswertung eines neuronalen Netzwerk-Modells beinhält, wird durch diese service-basierte Architektur (SOA) unterstützt. Der Ausführungs-Stack läuft auf vielen namhaften Cloud-Umgebungen, ist skalierbar und jede einzelne Komponente ist einfach erweiterbar und austauschbar.

Contents

1 Introduction

This thesis presents an execution stack for neural networks using the *Kubernetes*[1] container orchestration and a Java based microservice architecture, which is exposed to users and other systems via RESTful web services and a web frontend. The whole workflow including importing, training and evaluating a neural network model, becomes possible by using this service oriented approach (SOA). The presented stack runs on popular cloud platforms, like *Google Cloud Platform*[2], *Amazon AWS*[3] and *Microsoft Azure*[4]. Furthermore it is scalable and each component is extensible and interchangeable. This work is influenced by N2Sky [SM13], a framework to exchange neural network specific knowledge and aims to support *ViNNSL*, the Vienna Neural Network Specification Language [Kop15] [BVSW08].

Objectives: The first objective is to specify functional and non-functional requirements for the neural network system. This is followed by the characterisation of the API and the implemention of microservices that later define the neural network composition as a collection of loosely coupled services.

The next step is to setup a *Kubernetes* cluster to create the foundation of container orchestration.

Finally the microservices are deployed to containers and combined in a cluster.

Non-Objectives: The prototype does not fully implement the *ViNNSL* in version 2.0, as described in [Kop15] and provides limited data in-/output. Limitations are described in section TODO.

1 https://kubernetes.io
2 https://cloud.google.com/kubernetes-engine
3 https://aws.amazon.com/eks
4 https://azure.microsoft.com/services/container-service

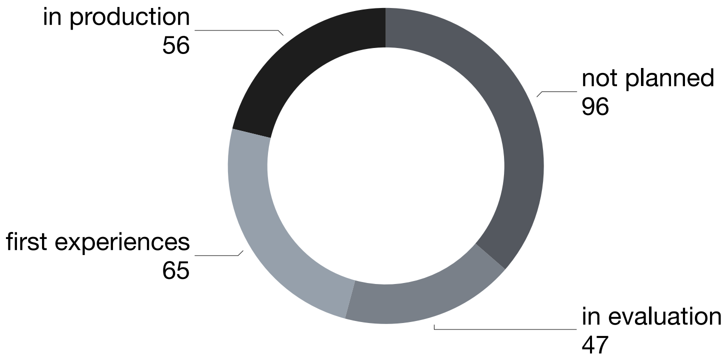

Figure 1.1: Distribution of machine learning of 264 companies in the DACH region [BB16]

1.1 Motivation

Machine learning has become a highly discussed topic in information technology in the past years and the trend is further increasing. It has become an essential part of everyday life when using search engines or speech recognition systems, like personal assistants. Self-learning algorithms in applications learn from the input of their users and decide which news an individual should read next, which song to listen to or which social media post should appear first. Messages are being analyzed and possible answers automatically predicted.

A recent Californian study shows that 6.5 million developers worldwide are currently involved in projects that use artificial intelligence techniques and another 5.8 million developers expect to implement these in near future [Eva17].

Machine learning is not just a business area in the United States, survey results of 264 companies in the DACH region show, that 56 of them already use that kind of technology in production. In the near future 112 companies plan to do so or already have initial experiences (see figure 1.1). It is seen by a fifth of the decision-makers as a core area to improve the competitiveness and profitability of companies in future. [BB16]

At the same time more and more companies shift their business logic from a monolithic design to microservices. Each service is dedicated to a single task that can be developed, deployed, replaced and scaled independently. Test results have shown that not only this architecture can help reduce infrastructure costs [VGO+16][VGC+15], but also reduces complexity of the code base and enables applications to dynamically adjust computing resources on demand [VGC+15].

The presented project combines these techniques and demonstrates a prototype that is open-source and is supported by common cloud providers. Developers can integrate their own solutions into the platform or exchange components ad libitum.

1.2 Structure

TODO

1.3 Problem Statement

TODO

- many different platforms
- complex field, hard to learn
- complex setup for many environments (design, training, auswertung, storage service, ...)

2 State of the Art

2.1 Containers

2.1.1 Docker Containers

Containers enable software developers to deploy applications that are portable and consistent across different environments and providers [Bai15] by running isolated on top of the operating system's kernel [BRBA17]. As an organisation, Docker[1] has seen an increase of popularity very quickly, mainly because of its advantages, which are speed, portability, scalability, rapid delivery, and density [BRBA17].

Building a Docker container is fast, because images do not include a guest operating system. The container format itself is standardized, which means that developers only have to ensure that their application runs inside the container, which is then bundled into a single unit. The unit can be deployed on any Linux system as well as on various cloud environments and therefore easily be scaled. Not using a full operating system makes containers use less resources than virtual machines, which ensures higher workloads with greater density. [Joy15]

2.2 Microservices

The micoservice architecture pattern is a variant of a service-oriented architecture (SOA). An often cited definition originates from Martin Fowler and James Lewis:

1 https://docker.com

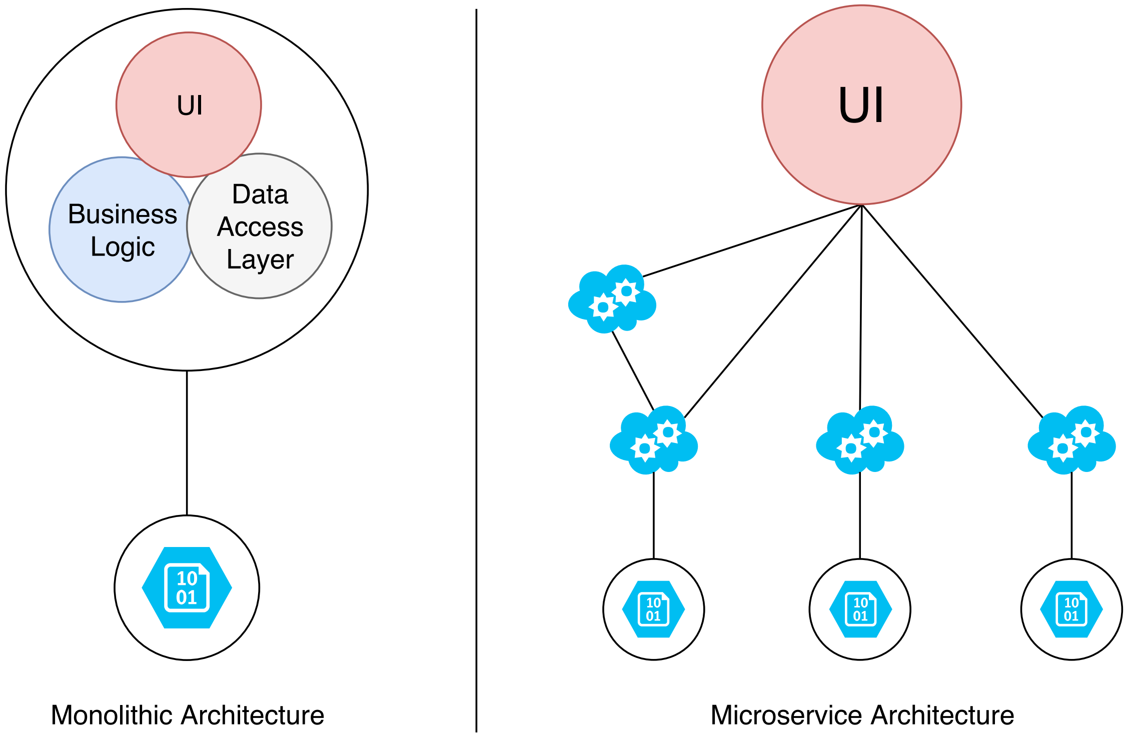

Figure 2.1: Monolithic Architecture vs. Microservice Architecture

> In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies. [LF14]

Figure 2.1 shows the architectural difference between the monolithic and microservice architecture. Monolithic applications bundle user interface, data access layer and business logic together a single unit. In the microservice architecture each task has its own service. The user interface puts together information from multiple services.

2.3 Container Orchestration Technologies

2.3.1 Kubernetes

Kubernetes was developed by Google for administering applications, that are provided in containers, in a cluster of nodes. Services that are responsible for controlling the cluster, are called master components [Ell16].

Master Components

The master consists of the core API server, that provides information about the cluster and workload state and allows to define the desired state [Bai15]. The master server also takes care of scheduling and scaling workloads, cluster-wide networking and performs health checks [Ell16]. Workloads are managed in form of so-called pods, which are various containers that conclude the application stacks [Bai15].

etcd etcd is a key-value store, accessible by a HTTP/JSON API, which can be distributed across multiple nodes and is used by Kubernetes to store configuration data, which needs to be accessible across nodes deployed in the cluster. It is essential for service discovery and to describe the state of the cluster, among other things. [Ell16]

etcd can also watch values for changes [Bai15].

kube-apiserver The API server acts as the main management point for the cluster and provides a RESTful interface for users and other services to configure workloads in the cluster. It is a bridge between other master components and is responsible of maintaining health and spreading commands in the cluster. [Ell16]

kube-scheduler The scheduler keeps track of available and allocated resources on each specific node in the cluster. It has an overview of the infrastructure environment and needs to distribute workload to an acceptable node without exceeding the available resources. Therefore each workload has to declare its operating requirements. [Ell16]

kube-controller-manager The controller manager mainly operates different controllers that constantly check the shared state of the cluster in `etcd` via the apiserver [The] and if the current state differs towards the desired state it takes compensating measures [Ell16].

For example the node controller's task is to react when nodes go offline or down. The replication controller makes sure that the defined number of desired pods is identical to the number of currently deployed pods in the cluster and scales applications up or down accordingly. The endpoints controller populates the endpoints to services [The]

cloud-controller-manager Kubernetes supports different cloud infrastructure providers. As each cloud providers has different features, apis and capabilities, cloud controller managers act as an abstraction to the generic internal Kubernetes constructs. This has the advantage that the core Kubernetes code is not dependent on cloud-provider-specific code. [The]

Node Components

kubelet

kube-proxy

Container Runtime Docker

Addons

DNS

Dashboard

2.3.2 Docker Swarm

https://github.com/GuillaumeRochat/container-orchestration-comparison

2.4 Machine Learning

Machine learning—the process by which computers can get better at performing tasks through exposure to data, rather than through explicit programming—requires massive computational power, the kind usually found in clusters of energy-guzzling, cloud-based computer servers outfitted with specialized processors. But an emerging trend promises to bring the power of machine learning to mobile devices that may lack or have only intermittent online connectivity. This will give rise to machines that sense, perceive, learn from, and respond to their environment and their users, enabling the emergence of new product categories, reshaping how businesses engage with customers, and transforming how work gets done across industries.(https://www2.deloitte.com/insights/us/en/focus/signals-for-strategists/machine-learning-mobile-applications.html) TODO CITATION

2.4.1 Classification

2.4.2 Neural Networks

Tensorflow

DL4J

3 Requirements

3.1 Functional Requirements

TODO

- NN in Cloud Rechnen
- Verwendung der verständlichen Beschreibungssprache ViNNSl
- all Devices, from everywhere
- berechnetes Netzwerk kann in eig App verwendet werden / oder als Webservice exposed

3.1.1 User Interface

Mockup

TODO

Figure 3.1 shows a sketch of the user interface.

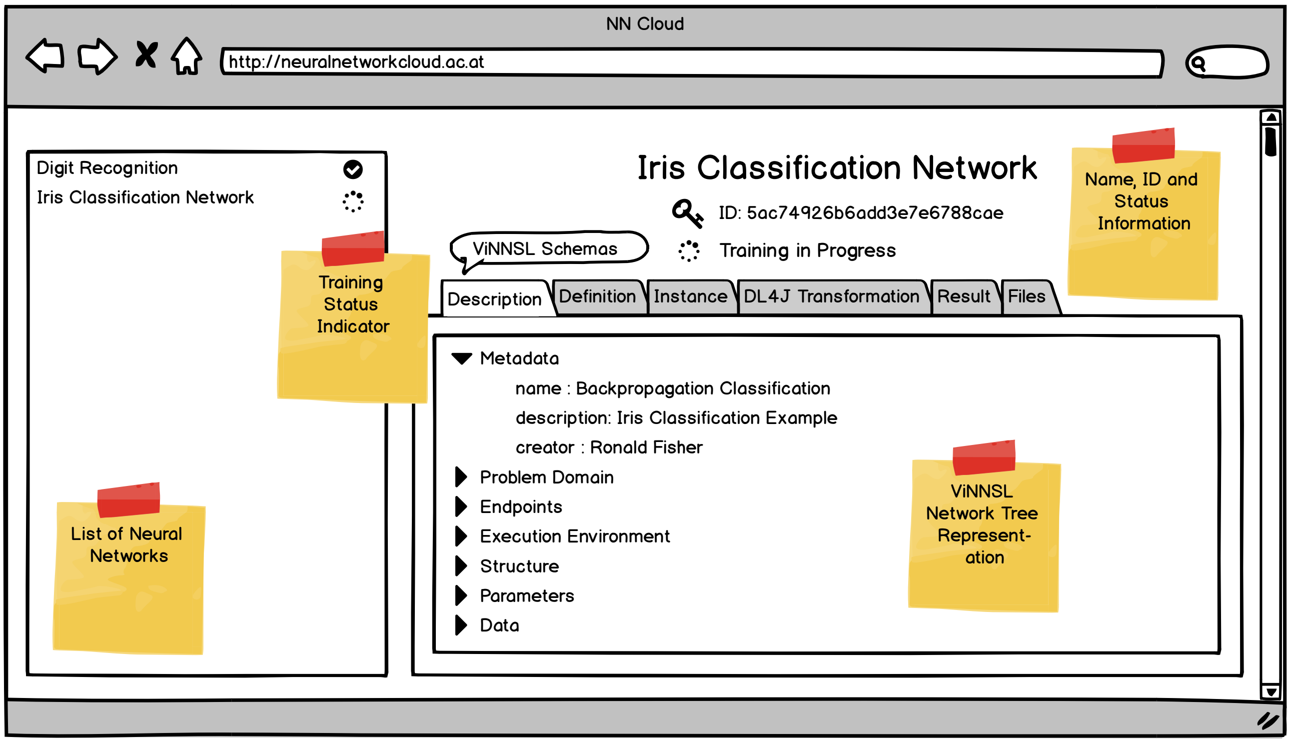

Figure 3.1: Mockup: User Interface of Frontend Service

3.2 Non-Functional Requirements

3.2.1 Quality

3.2.2 Technical

3.2.3 Software

3.2.4 Hardware

3.2.5 Documentation

3.2.6 Developer Environment

4 Specification

4.1 Use Case

Figure 4.1 shows the UML use case diagram.

4.1.1 Use Case Descriptions

TODO (hinzufügen: dev: kann trainiertes netz in eigener app verwenden ,data scientist: trainiertes netzwerk exportieren und developer überreichen)

4.2 Sequence Diagram

4.2.1 Sequence of Training

TODO

Figure 4.2 shows the sequence diagram.

4.3 Data Model Design

TODO

Figure 4.3 shows the data schema.

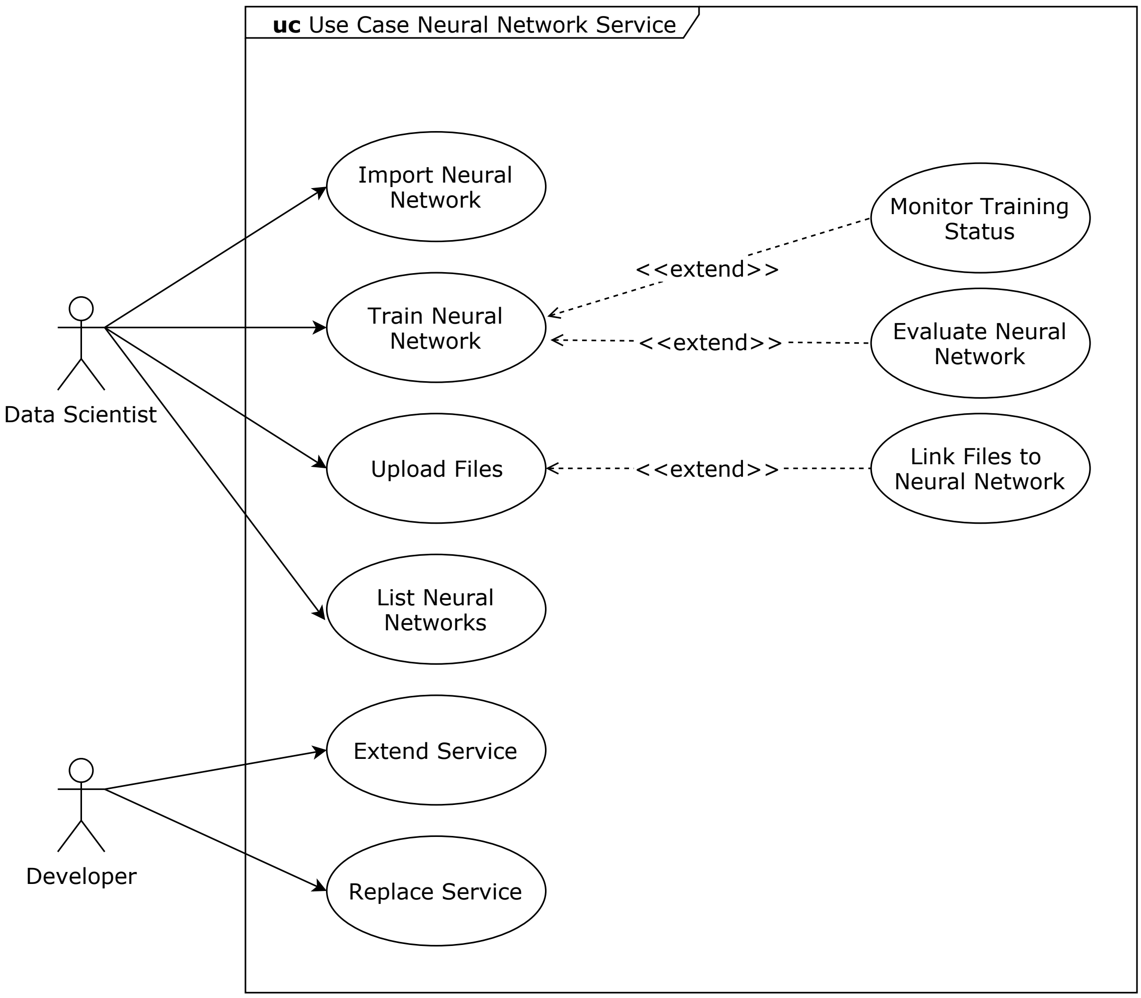

Figure 4.1: UML Use Case Diagram

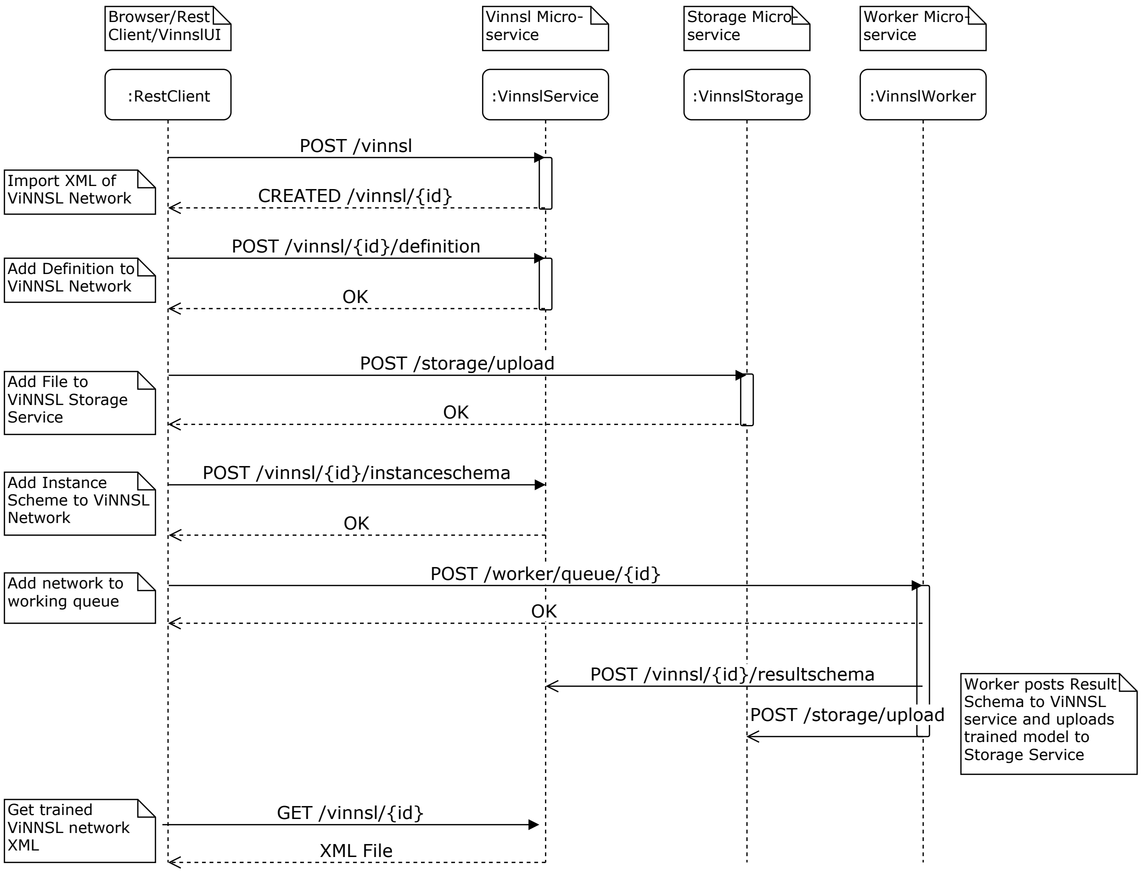

Figure 4.2: Training Sequence Diagram

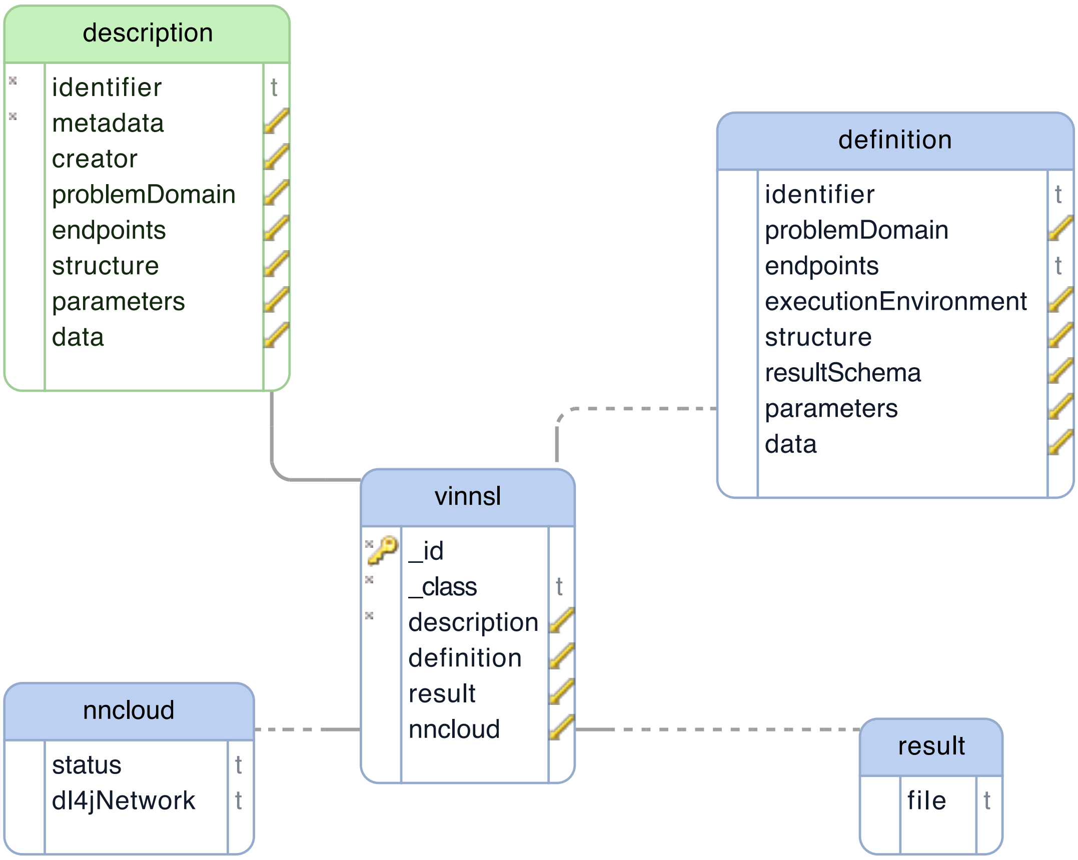

Figure 4.3: NoSQL Data Model

4.4 Overview Microservices

The neural network cloud execution stack consists of four main services that expose a RESTful API to users and two supporting services in charge of persisting data. Figure **??** shows an overview of these services.

4.4.1 Vinnsl Service (vinnsl-service)

The `vinnsl-service` is responsible for handling the import, management and manipulation of neural network objects and it's status. It maps the CRUD[1] operations to HTTP methods. A new neural network is created by sending a `POST` request to the `/vinnsl` endpoint containing a ViNNSL Definition XML as body. Sending a `GET` request to the `/vinnsl` route returns a JSON containing all ViNNSL neural network objects.

The `vinnsl-service` depends on the `vinnsl-db` service, which runs a MongoDB database to store the objects.

4.4.2 Worker Service (vinnsl-nn-worker)

The `vinnsl-nn-worker` implements a queue management for neural network training and transforms ViNNSL neural network models into DL4J models. It provides a wrapper of the DL4J platform, that handles the training or evaluation of the network.

4.4.3 Storage Service (vinnsl-storage-service)

Binary files, like trained network models, images or csv files are essential in the pocess of creating and training neural networks. File management is handled by the `vinnsl-storage-service`.

1 Create, Read, Update, Delete

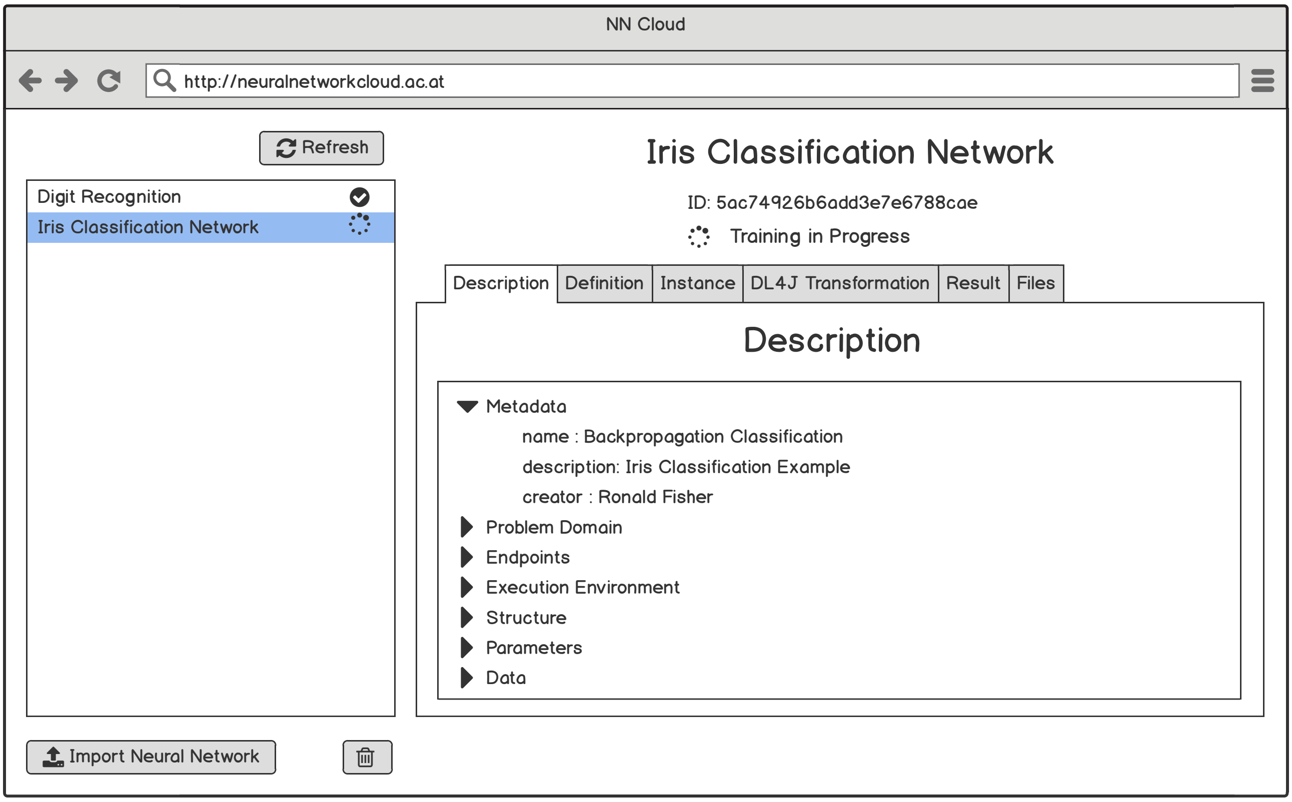

Figure 4.4: User Interface Design for vinnsl-nn-ui

4.4.4 Frontend UI (vinnsl-nn-ui)

The Frontend UI is a web application that gives a brief overview of all neural network models, their training status and linked files.

4.5 User Interface Design

Auf Grundlage des ersten Sketches, wurde ein erstes Designmodell entwickelt.

Figure 4.4 shows the user interface design for the frontend web service.

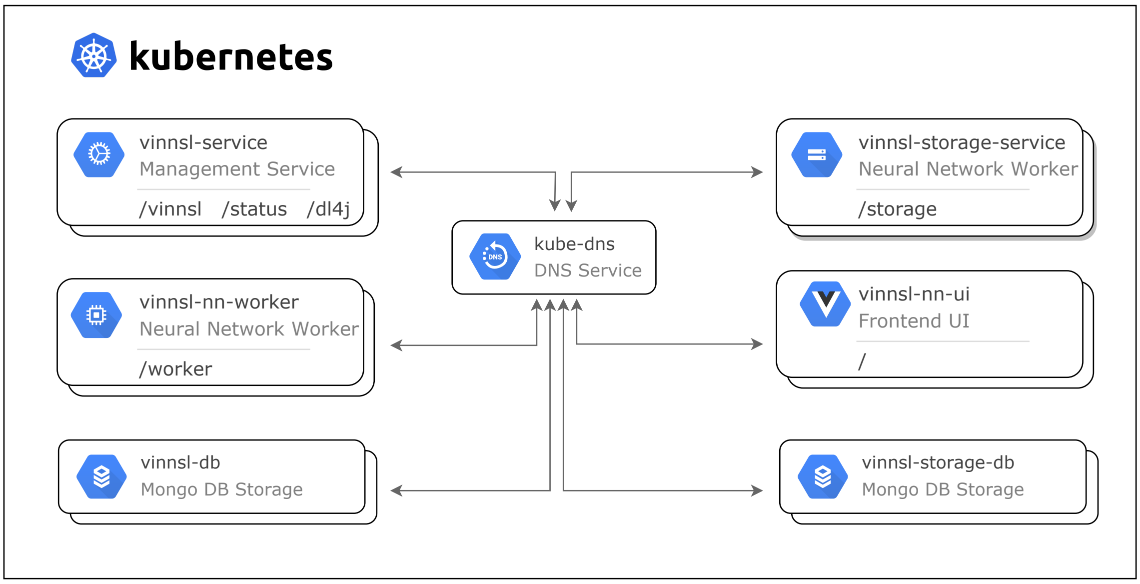

Figure 4.5: Service Discovery with kube-dns

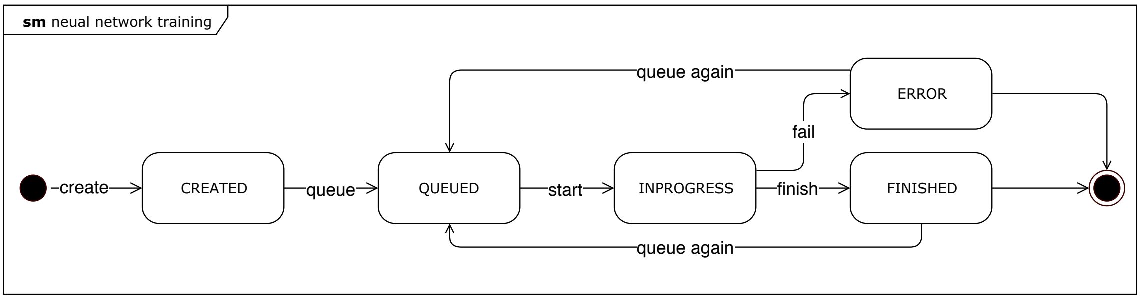

Figure 4.6: State Machine of a Neural Network

4.6 Service Discovery and Load Balancing

4.7 Neural Network Objects

4.7.1 State of Neural Network Objects

4.8 Class Diagram

5 REST API Documentation

Base URL

`http[s]://<clusterip>`

5.1 vinnsl-service

5.1.1 Import a new ViNNSL XML Defintion

`POST /vinnsl`

Parameters

Type	Name	Description	Schema
Body	**vinnsl** *required*	vinnsl	Vinnsl

Responses

HTTP Code	Description	Schema
201	Created	No Content
500	Server Error	Error

Consumes

- `application/xml`

Produces

- `*/*`

Tags

- vinnsl-service-controller

Example HTTP request

Header

```
Content-Type: application/xml
```

Body

```
<vinnsl>
  <description>
    <identifier><!-- will be generated --></identifier>
    <metadata>
      <paradigm>classification</paradigm>
      <name>Backpropagation Classification</name>
      <description>Iris Classification Example</description>
      <version>
        <major>1</major>
        <minor>0</minor>
      </version>
    </metadata>
```

```
<creator>
  <name>Ronald Fisher</name>
  <contact>ronald.fisher@institution.com</contact>
</creator>
<problemDomain>
  <propagationType type="feedforward">
    <learningType>supervised</learningType>
  </propagationType>
  <applicationField>Classification</applicationField>
  <networkType>Backpropagation</networkType>
  <problemType>Classifiers</problemType>
</problemDomain>
<endpoints>
  <train>true</train>
  <retrain>true</retrain>
  <evaluate>true</evaluate>
</endpoints>
<structure>
   <input>
    <ID>Input1</ID>
    <size>
        <min>4</min>
        <max>4</max>
    </size>
   </input>
   <hidden>
    <ID>Hidden1</ID>
    <size>
        <min>3</min>
        <max>3</max>
    </size>
   </hidden>
   <hidden>
    <ID>Hidden2</ID>
    <size>
```

```
        <min>3</min>
        <max>3</max>
      </size>
     </hidden>
     <output>
      <ID>Output1</ID>
      <size>
        <min>3</min>
        <max>3</max>
      </size>
     </output>
   </structure>
   <parameters/>
   <data>
      <description>iris txt file with 3 classifications, 4 input vars</description>
      <tabledescription>no input as table possible</tabledescription>
      <filedescription>CSV file</filedescription>
   </data>
  </description>
</vinnsl>
```


Example HTTP response

Statuscode: 201 CREATED

Header

`Location: https://<baseURL>/vinnsl/5ade36bbd601800001206798`

5.1.2 List all Neural Networks

`GET /vinnsl`

Responses

HTTP Code	Description	Schema
200	OK	< Vinnsl > array
404	Not Found	No Content
500	Server Error	Error

Produces

- `application/json`

Tags

- vinnsl-service-controller

Example HTTP Response


```
[
    {
        "identifier": "5ab91658e8cc45946600ea11",
        "description": {},
        "definition": {},
        "data": {},
        "instance": {},
        "trainingresult": {},
        "result": {},
        "nncloud": {
            "status": "CREATED",
            "dl4jNetwork": "{}
        }
    },
```

```
    ...
]
```

5.1.3 Delete all Neural Networks

`DELETE /vinnsl/deleteall`

Responses

HTTP Code	Description	Schema
200	OK	object
204	No Content	No Content
500	Server Error	Error

Produces

- `application/json`

Tags

- vinnsl-service-controller

5.1.4 Get Neural Network Object

`GET /vinnsl/{id}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Responses

HTTP Code	Description	Schema
200	OK	Vinnsl
404	Not Found	No Content

Produces

- `application/xml`
- `application/json`

Tags

- vinnsl-service-controller

Example HTTP response


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<vinnsl>
    <identifier>5ab91658e8cc45946600ea11</identifier>
    <description>
        <identifier></identifier>
        <metadata>
            <paradigm>classification</paradigm>
            <name>Backpropagation Classification</name>
            <description>Face Recognition Example</description>
```

```
    <version>
        <major>1</major>
        <minor>5</minor>
    </version>
</metadata>
<creator>
    <name>Autor 1</name>
    <contact>author1@institution.com</contact>
</creator>
<problemDomain>
    <propagationType type="feedforward">
        <learningType>supervised</learningType>
    </propagationType>
    <applicationField>EMS</applicationField>
    <applicationField>Operations</applicationField>
    <applicationField>FaceRecoginition</applicationField>
    <networkType>Backpropagation</networkType>
    <problemType>Classifiers</problemType>
</problemDomain>
<endpoints>
    <train>true</train>
    <retrain>true</retrain>
    <evaluate>true</evaluate>
</endpoints>
<structure>
    <input>
        <ID>Input1</ID>
        <dimension>
            <min>1</min>
            <max>1</max>
        </dimension>
        <size>
            <min>960</min>
            <max>960</max>
        </size>
```

```
        </input>
        <hidden>
            <ID>Hidden1</ID>
            <dimension>
                <min>1</min>
                <max>1024</max>
            </dimension>
        </hidden>
        <output>
            <ID>Output1</ID>
            <dimension>
                <min>1</min>
                <max>1</max>
            </dimension>
            <size>
                <min>1</min>
                <max>1</max>
            </size>
        </output>
    </structure>
    <parameters/>
    <data>
        <description>Input are face images with 32x30 px</description>
        <tabledescription>no input as table possible</tabledescription>
        <filedescription>prepare the input as file by reading the image files</file
    </data>
</description>
<definition>
    <identifier></identifier>
    <problemDomain>
        <propagationType type="feedforward">
            <learningType>supervised</learningType>
        </propagationType>
        <applicationField>EMS</applicationField>
        <applicationField>Operations</applicationField>
```

```
    <applicationField>FaceRecoginition</applicationField>
    <networkType>Backpropagation</networkType>
    <problemType>Classifiers</problemType>
</problemDomain>
<endpoints></endpoints>
<executionEnvironment>
    <serial>true</serial>
</executionEnvironment>
<structure>
    <input>
        <ID>Input1</ID>
        <dimension>1</dimension>
        <size>960</size>
    </input>
    <hidden>
        <ID>Hidden1</ID>
        <dimension>1</dimension>
        <size>1024</size>
    </hidden>
    <output>
        <ID>Output1</ID>
        <dimension>1</dimension>
        <size>1</size>
    </output>
    <connections/>
</structure>
<resultSchema>
    <instance>true</instance>
    <training>true</training>
</resultSchema>
<parameters>
    <valueparameter name="learningrate">0.4</valueparameter>
    <valueparameter name="biasInput">1</valueparameter>
    <valueparameter name="biasHidden">1</valueparameter>
    <valueparameter name="momentum">0.1</valueparameter>
```

```
            <comboparameter name="ativationfunction">sigmoid</comboparameter>
            <valueparameter name="threshold">0.00001</valueparameter>
            <comboparameter name="activationfunction">sigmoid</comboparameter>
        </parameters>
        <data>
            <description>Input are face images with 32x30 px</description>
            <dataSchemaID>iris.txt</dataSchemaID>
        </data>
    </definition>
    <data>
        <identifier>5ab4e69c8f136a16bf81f093</identifier>
        <data>
            <file>5ab4e69c8f136a16bf81f093</file>
        </data>
    </data>
</vinnsl>
```


5.1.5 Remove Neural Network Object

```
DELETE /vinnsl/{id}
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Responses

HTTP Code	Description	Schema
200	OK	ResponseEntity
204	No Content	No Content

HTTP Code	Description	Schema
500	Server Error	No Content

Produces

- `*/*`

Tags

- vinnsl-service-controller

5.1.6 Add/Replace File of Neural Network

`PUT /vinnsl/{id}/addfile`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Query	**fileId** *required*	fileId	string

Responses

HTTP Code	Description	Schema
200	OK	Vinnsl
404	Not Found	No Content
500	Server Error	Error

Consumes

- `application/json`

Produces

- `application/xml`
- `application/json`

Tags

- vinnsl-service-controller

5.1.7 Add/Replace ViNNSL Definition of Neural Network

`PUT /vinnsl/{id}/definition`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Body	**def** *required*	def	Definition

Responses

HTTP Code	Description	Schema
200	OK	Vinnsl
404	Not Found	No Content

HTTP Code	Description	Schema
500	Server Error	Error

Consumes

- `application/xml`
- `application/json`

Produces

- `*/*`

Tags

- vinnsl-service-controller

Example HTTP request

Request body

```
<definition>
<identifier><!-- will be generated --></identifier>
<metadata>
  <paradigm>classification</paradigm>
  <name>Backpropagation Classification</name>
  <description>Iris Classification Example</description>
  <version>
    <major>1</major>
    <minor>0</minor>
  </version>
</metadata>
```

```
<creator>
  <name>Ronald Fisher</name>
  <contact>ronald.fisher@institution.com</contact>
</creator>
<problemDomain>
  <propagationType type="feedforward">
    <learningType>supervised</learningType>
  </propagationType>
  <applicationField>Classification</applicationField>
  <networkType>Backpropagation</networkType>
  <problemType>Classifiers</problemType>
</problemDomain>
<endpoints>
  <train>true</train>
</endpoints>
<executionEnvironment>
    <serial>true</serial>
</executionEnvironment>
<structure>
   <input>
    <ID>Input1</ID>
    <size>4</size>
   </input>
   <hidden>
    <ID>Hidden1</ID>
    <size>3</size>
   </hidden>
   <hidden>
    <ID>Hidden2</ID>
    <size>3</size>
   </hidden>
   <output>
    <ID>Output1</ID>
    <size>3</size>
   </output>
```

```
    <connections>
     <!--<fullconnected>
         <fromblock>Input1</fromblock>
         <toblock>Hidden1</toblock>
         <fromblock>Hidden1</fromblock>
         <toblock>Output1</toblock>
     </fullconnected>-->
    </connections>
  </structure>
  <resultSchema>
     <instance>true</instance>
     <training>true</training>
  </resultSchema>
  <parameters>
     <valueparameter name="learningrate">0.1</valueparameter>
     <comboparameter name="activationfunction">tanh</comboparameter>
     <valueparameter name="iterations">500</valueparameter>
     <valueparameter name="seed">6</valueparameter>
  </parameters>
  <data>
     <description>iris txt file with 3 classifications, 4 input vars</description>
     <dataSchemaID>name/iris.txt</dataSchemaID>
  </data>
</definition>
```


5.1.8 Add/Replace ViNNSL Instanceschema of Neural Network

```
PUT /vinnsl/{id}/instanceschema
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Type	Name	Description	Schema
Body	**instance** *required*	instance	Instanceschema

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content
500	Server Error	Error

Consumes

- `application/xml`
- `application/json`

Produces

- `*/*`

Tags

- vinnsl-service-controller

Example HTTP request

Request body

```
<instanceschema>
</instanceschema>
```

5.1.9 Add/Replace ViNNSL Resultschema of Neural Network

`PUT /vinnsl/{id}/resultschema`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Body	**resultSchema** *required*	resultSchema	Resultschema

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content
500	Server Error	Error

Consumes

- `application/xml`
- `application/json`

Produces

- `*/*`

Tags

- vinnsl-service-controller

Example HTTP request

Request body

```
<resultschema>
</resultschema>
```

5.1.10 Add/Replace ViNNSL Trainingresult of Neural Network

```
PUT /vinnsl/{id}/trainingresult
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Body	**trainingresult** *required*	trainingresult	Trainingresultschema

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content
500	Server Error	Error

Consumes

- `application/xml`
- `application/json`

Produces

- `*/*`

Tags

- vinnsl-service-controller

Example HTTP request

Request body

```
<trainingresult>
</trainingresult>
```

5.1.11 Get Status of all Neural Networks

```
GET /status
```

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content

Produces

- `application/json`

Tags

- nn-status-controller

HTTP response example

```
{
    "5ab91658e8cc45946600ea11": "INPROGRESS"
}
```

5.1.12 Get Status of Neural Network

```
GET /status/{id}
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content

Produces

- `application/json`

Tags

- nn-status-controller

5.1.13 Set Status of a Neural Network

`PUT /status/{id}/{status}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Path	**status** *required*	status	enum (`CREATED`, `QUEUED`, `INPROGRESS`, `FINISHED`, `ERROR`)

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content
500	Server Error	Error

Consumes

- `application/json`

Produces

- `application/json`

Tags

- nn-status-controller

5.1.14 Get Deeplearning4J Transformation Object of Neural Network

`GET /dl4j/{id}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Responses

HTTP Code	Description	Schema
200	OK	string
404	Not Found	No Content

Produces

- `application/json`

Tags

- dl4j-service-controller

5.1.15 Put Deeplearning4J Transformation Object of Neural Network

`PUT /dl4j/{id}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Body	**dl4J** *required*	dl4J	string

Responses

HTTP Code	Description	Schema
200	OK	ResponseEntity
404	Not Found	No Content
500	Server Error	Error

Consumes

- `application/json`

Produces

- `application/json`

Tags

- dl-4j-service-controller

5.2 vinnsl-storage-service

5.2.1 Handle File Upload from HTML Form

`POST /storage`

Parameters

Type	Name	Description	Schema
FormData	**file** *required*	file	file

Responses

HTTP Code	Description	Schema
200	OK	string
201	Created	No Content
404	Not Found	No Content

Consumes

- `multipart/form-data`

Produces

- `\*/*`

Tags

- vinnsl-storage-controller

5.2.2 List all Files

`GET /storage`

Responses

HTTP Code	Description	Schema
200	OK	Model
404	Not Found	No Content

Produces

- `application/json`

Tags

- vinnsl-storage-controller

5.2.3 Download File by Original Filename

`GET /storage/files/name/{filename}`

Parameters

Type	Name	Description	Schema
Path	**filename** *required*	filename	string

Responses

HTTP Code	Description	Schema
200	OK	string (byte)
404	Not Found	No Content

Produces

- `\*/*`

Tags

- vinnsl-storage-controller

5.2.4 Download or Show File by FileID

`GET /storage/files/{fileId}`

Parameters

Type	Name	Description	Schema
Path	**fileId** *required*	fileId	string
Query	**download** *optional*	download	boolean

Responses

HTTP Code	Description	Schema
200	OK	string (byte)
404	Not Found	No Content

Produces

- `\*/*`

Tags

- vinnsl-storage-controller

5.2.5 Delete File by FileID

`DELETE /storage/files/{fileId}`

Parameters

Type	Name	Description	Schema
Path	**fileId** *required*	fileId	string

Responses

HTTP Code	Description	Schema
200	OK	ResponseEntity
204	No Content	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content

Produces

- `\*/*`

Tags

- vinnsl-storage-controller

5.2.6 Get File Metadata by FileID

`GET /storage/metadata/{fileId}`

Parameters

Type	Name	Description	Schema
Path	**fileId** *required*	fileId	string

Responses

HTTP Code	Description	Schema
200	OK	< string, object > map
404	Not Found	No Content

Produces

- `\*/*`

Tags

- vinnsl-storage-controller

5.2.7 Upload MultipartFile

`POST /storage/upload`

Parameters

Type	Name	Description	Schema
FormData	**file** *required*	file	file

Responses

HTTP Code	Description	Schema
200	OK	object
201	Created	No Content
404	Not Found	No Content

Consumes

- `multipart/form-data`

Produces

- `application/json`

Tags

- vinnsl-storage-controller

5.2.8 Upload File by URL

`GET /storage/upload`

Parameters

Type	Name	Description	Schema
Query	**url** *required*	url	string

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content

Produces

- `application/json`

Tags

- vinnsl-storage-controller

5.3 vinnsl-worker-service

5.3.1 getWorkingQueue

`GET /worker/queue`

Responses

HTTP Code	Description	Schema
200	OK	< string > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Produces

- `\*/*`

Tags

- worker-controller

5.3.2 addToWorkingQueue

`PUT /worker/queue/{id}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Responses

HTTP Code	Description	Schema
200	OK	< string > array
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- worker-controller

6 Implementation of a Prototype

6.1 User Interface

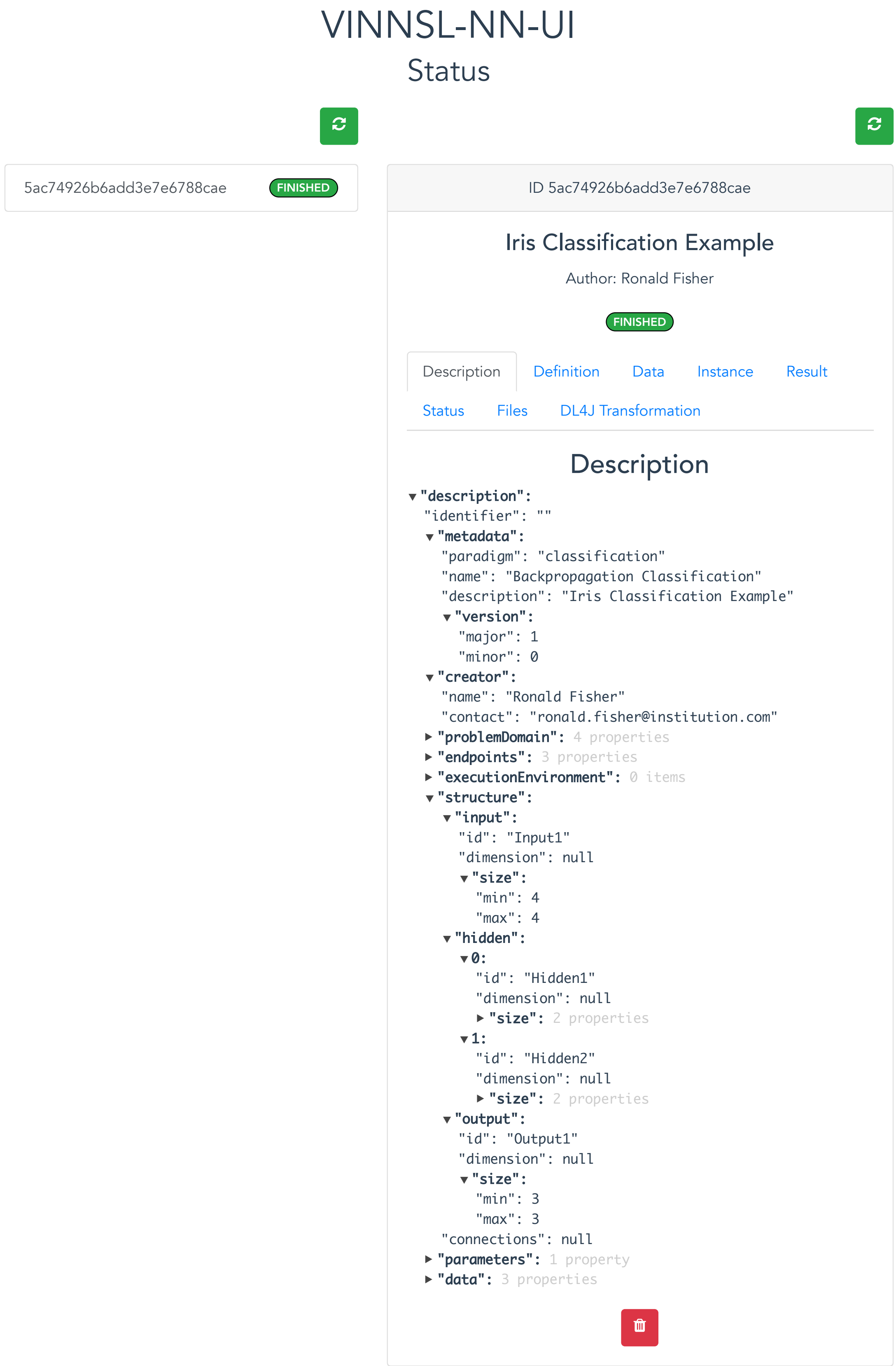

Figure 6.1: User Interface of Prototype

7 Use Cases

1) iris classification
2) MNIST?
3) hosted trained network

8 Future Work

TODO

- more function

- backend für tensorflow
- grafischer NN designer
- trainierte netzwerke als webservice veröffentlichen
- integration in knime platform

9 Conclusions

10 Acknowledgments

11 Dedication

12 Appendices

Bibliography

[Bai15] Baier, Jonathan: *Getting Started with Kubernetes*. Packt Publishing, 2015

[BB16] Björn Böttcher, Dr. Carlo V. Daniel Klemm K. Daniel Klemm: Machine Learning im Unternehmenseinsatz / Crisp Research AG. Version: 2016. `https://www.unbelievable-machine.com/downloads/studie-machine-learning.pdf`. 2016. – Forschungsbericht

[BRBA17] Bashari Rad, Babak; Bhatti, Harrison; Ahmadi, Mohammad: An Introduction to Docker and Analysis of its Performance. In: *IJCSNS International Journal of Computer Science and Network Security* 17 (2017), 03, Nr. 3, S. 228–235

[BVSW08] Beran, P. P.; Vinek, E.; Schikuta, E.; Weishaupl, T.: ViNNSL - the Vienna Neural Network Specification Language. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008. – ISSN 2161–4393, S. 1872–1879

[Ell16] Ellingwood, Justin: *An Introduction to Kubernetes - DigitalOcean*. Version: 2016. `https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes`, Abruf: 2018-05-08

[Eva17] Evans Data Corporation: *AI, ML, and Big Data Survey 2017, Vol. 2*. Version: 2017. `https://evansdata.com/reports/viewRelease.php?reportID=37`

[Joy15] Joy, A. M.: Performance comparison between Linux containers and virtual machines. In: *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, S. 342–346

[Kop15] Kopica, Thomas: *Vienna Neural Network Specification Language 2.0*, Masterthesis, 2015

[LF14] Lewis, James; Fowler, Martin: *Microservices: a definition of this new architectural term*. 2014

[SM13] Schikuta, Erich; Mann, Erwin: N2Sky - Neural networks as services in the clouds. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*, IEEE, aug 2013. – ISBN 978–1–4673–6129–3, 1-8

[The] The Kubernetes Authors: *Kubernetes Components*. `https://kubernetes.io/docs/concepts/overview/components/`, Abruf: 2018-05-10

[VGC+15] Villamizar, M.; Garcés, O.; Castro, H.; Verano, M.; Salamanca, L.; Casallas, R.; Gil, S.: Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: *2015 10th Computing Colombian Conference (10CCC)*, 2015, S. 583–590

[VGO+16] Villamizar, M.; Garcés, O.; Ochoa, L.; Castro, H.; Salamanca, L.; Verano, M.; Casallas, R.; Gil, S.; Valencia, C.; Zambrano, A.; Lang, M.: Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. In: *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016, S. 179–182