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	Unterschrift
Date	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

1.1 Problem Statement

Getting started with machine learning and in particular with neural networks is not a trivial task. It is a complex field with a high entry barrier and most often requires programming skills and expertise in neural network frameworks. In most cases a complex setup is needed to train and evaluate networks, which is both a processor- and memory-intense job. With cloud computing getting more and more affordable and powerful, it makes sense to shift these tasks into the cloud. There are already existing cloud platforms for machine learning, but to my present research all of them do not fulfil at least one of the following criteria:

- platform is open-source
- no programming skills required to define and train a neural network model
- can be deployed on-site and and in the cloud (of your choice)
- components extensible and replaceable by developers
- provides a RESTful interface

This thesis showcases an architecture, that tries to achieve all of that.

1.2 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].

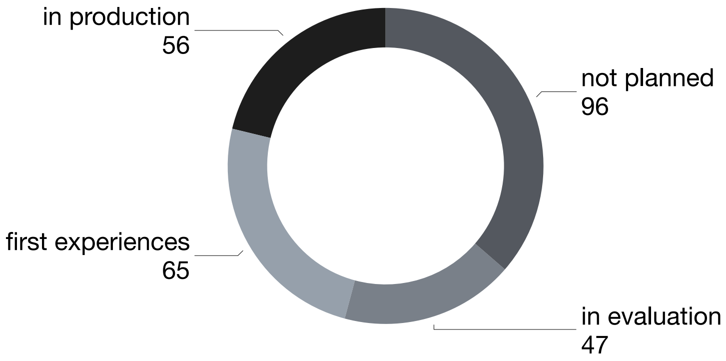

Figure 1.1: Distribution of machine learning of 264 companies in the DACH region [BB16]

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 supported by common cloud providers. Developers can integrate their own solutions into the platform or exchange components ad libitum.

It also integrates with ViNNSL, a descriptive language that does not require programming skills to define, train and evaluate neural networks.

1.3 Structure

TODO

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] compared to other solutions.

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

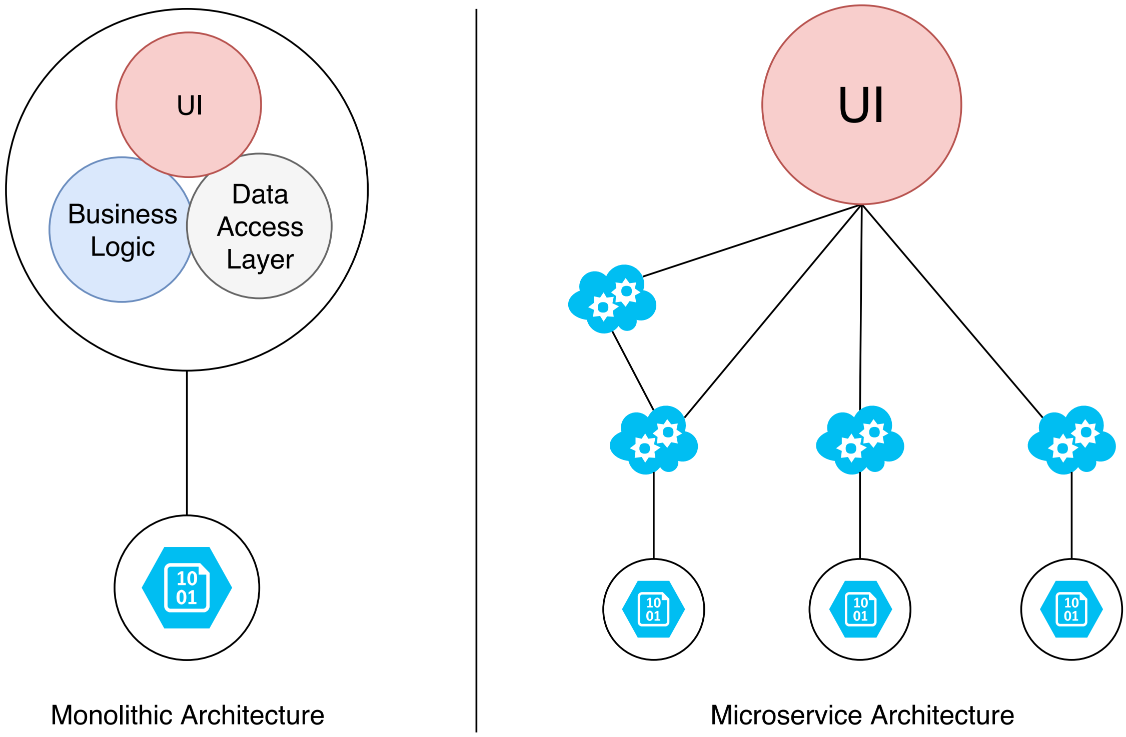

Figure 2.1: Monolithic Architecture vs. Microservice Architecture

> 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 information together from multiple services.

2.3 Container Orchestration Technologies

As every single microservice runs as a container, we need a tool to manage, organise and replace these containers. Services should also be able to speak to each other and restarted if they fail. Services under heavy load should be scaled for better performance. To deal with these challenges container orchestration technologies come into place. According to a study from 2017 published by Portworx, Kubernetes is the most frequently used container orchestration tool in organizations, followed by Docker Swarm and Amazon ECS. [Por]

This section describes the architecture of the mentioned container orchestration technologies and compares them.

2.3.1 Kubernetes

Kubernetes is the third container-management system (after Borg and Omega) developed by Google [BGO+16] 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]. Figure 2.2 shows the Kubernetes core architecture, which includes the Master server, the nodes and the interaction between the components.

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].

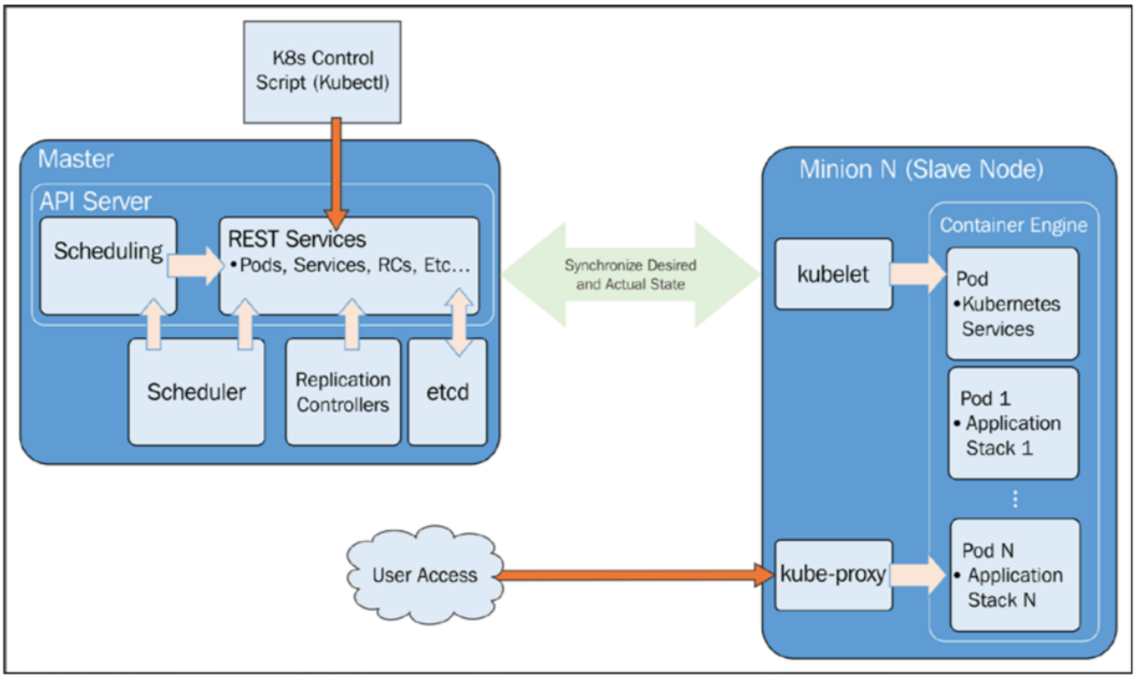

Figure 2.2: Kubernetes core architecture TODO Source!

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 [Kubb] 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 [Kubb]

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. [Kubb]

Node Components

Servers that accomplish workloads are called nodes. Each workload is described as one or more containers that have to be deployed. Node components run on every node in the cluster providing the Kubernetes runtime environment [Kubb], that establishes networking and communicates with the master components. They also take care of deploying the necessary containers on a node and keep them running [Ell16]. Kubernetes requires a dedicated subnet for each node server and a supported container runtime [Kubb].

kubelet The kubelet is the primary agent running on each node in the cluster, responsible for running pods [Kubb]. It communicates with the API server to receive commands invoked by the scheduler. Interaction takes place with the etcd store to read and update configuration and state of the pod.

Pods are specified by the *PodSpec*, which defines the workload and parameters on how to run the containers [Ell16]. The kubelet process is responsible that the containers described in the specification are running and healthy [Kubb].

kube-proxy The proxy service is in charge of forwarding requests of defined services to the correct containers. On a basic level, load balancing is also done by the proxy. [Bai15]

Container Runtime The container runtime is an implementation running containers. Currently Docker, rkt, runc and OpenContainer runtimes are supported. [Kubb]

Pods A pod is the smallest deployable unit in a cluster consisting of a group of one or more containers, which share network and storage. [Kubc]

Addons

Cluster DNS Cluster DNS server keeps track of running services in the cluster and updates DNS records accordingly. This allows an easy way of service discovery. Containers include this DNS server in their DNS lookups automatically – that way a service can resolve another service by its name. [Bai15]

Ingress Ingress handles the traffic from outside the cluster and forwards it to the correct service using the dns service acting as a proxy server. Currently there are two official implementations: `ingress-gce` and `ingress-nginx`. *Ingress* also provides basic load balancing. [Kuba]

Dashboard The dashboard is a web-based user interface that allows to manage Kubernetes clusters and applications running in the cluster [Kubb]. It also provides access to log messages in each pod.

Minikube

Minikube is a tool to run a single-node Kubernetes cluster locally on computers supporting various virtual machine drivers. TODO Source

2.3.2 Docker Swarm

On October 17, Docker announced that it will integrate Kubernetes into the Docker platform. Users can chose to use Kubernetes and/or Docker Swarm for orchestration.

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

2.3.3 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

Deeplearning4J

3 Requirements

This section defines functional and non-functional requirements for the developed prototype. The neural network execution stack focuses on two main target groups: data scientists and developers.

Data scientists use the provided services in a deployed environment (cloud or own computer) to develop and train their neural networks. The system should be easy to setup and no programming knowledge should be needed to get started.

Developers can extend the neural network stack with features or use the provided web services to implement their own custom solution.

3.1 Functional Requirements

Due to the fact that neural network training requires a lot of computing power, the main requirement is to design an architecture that can be executed in the cloud or on-site cluster hardware.

To enable developers to extend the application, it is designed as a platform that is open-sourced and documented. An easy setup on a local computer and small micro-services with a clear structure and manageable code base make it easier to get acquainted with the architecture.

The neural network platform should also offer a way to be extended or used by external applications and services, therefore a documented RESTful webservice is provided, that can be consumed by various clients.

3.1.1 User Interface

The user interface shall be a web application that gives a quick overview of all neural networks and their training status. The frontend uses the RESTful API as backend source and does not cover the whole function range of the API.

Mockup

Figure 3.1 shows a sketch of the user interface. On the left side the user can see a list of all created or imported neural networks. Next to the names of the networks, there is an icon representing the training status. In the detailed view on the right side, the title and id of the network is shown followed by an indicator when training is in progress. The visualisation of a neural network is divided into tabs.

The tabs “Description”, “Definition”, “Instance” and “Result” represent the eponymous ViNNSL Description XML file into a graphical tree view. When enough information is provided by ViNNSL XML files, the worker service performs a transformation into the internally used model representation of the *Deeplearning4J* Framework. The *Deeplearning4J* Tab shows the transformed object. In the “Files” tab, imported files of the storage services are listed and can be selected as training- or testset.

3.2 Non-Functional Requirements

3.2.1 Quality

The execution stack shall comply with the following quality features:

- Standard RESTful API
- the user interface works on all common browsers and devices (responsive design)
- loading time of the user interface should be less than three seconds

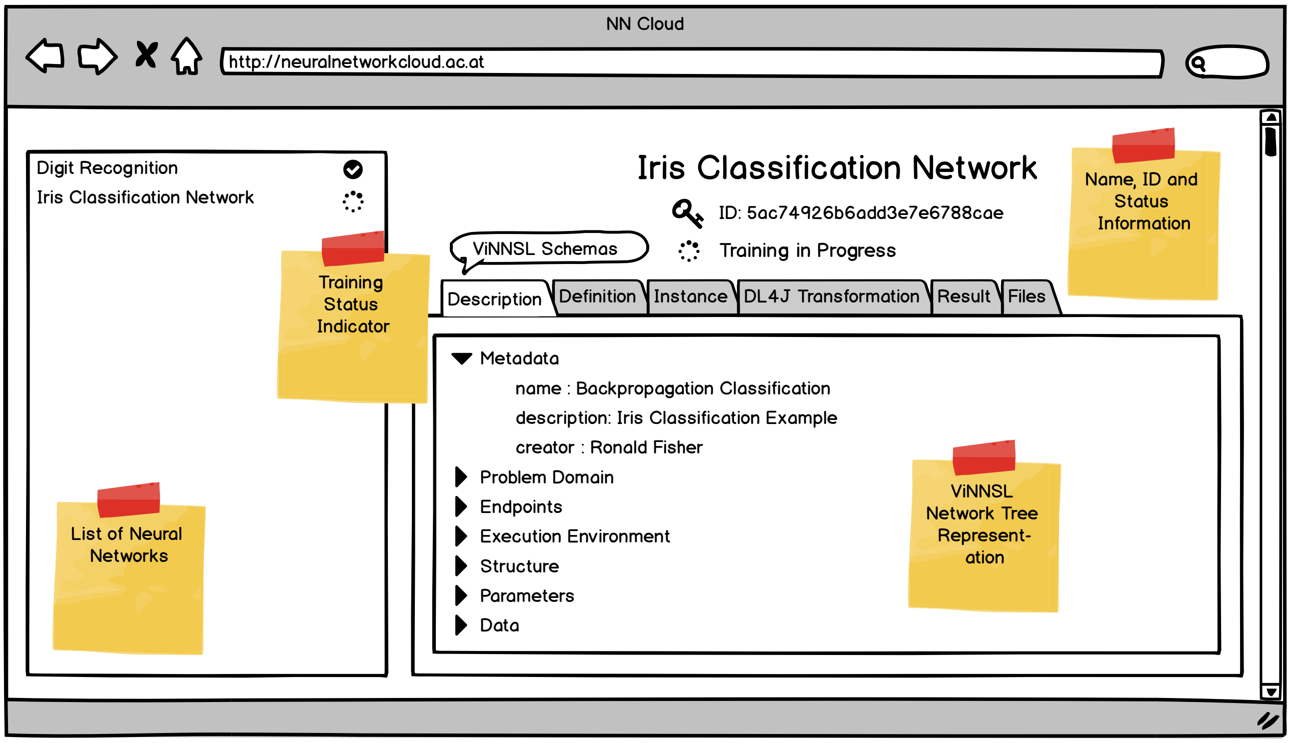

Figure 3.1: Mockup: User Interface of Frontend Service

3.2.2 Technical

3.2.3 Software

- Kubernetes
- Docker
- Java Standard Platform
- Maven Plugin for Java

3.2.4 Hardware

- Kubernetes compatible hardware or Cloud account (Amazon Web Services, Google Cloud Engine)

3.2.5 Documentation

The documentation is provided in Section 6 or online on SwaggerHub[1].

3.2.6 Source Code

The source code is released on GitHub [2].

3.2.7 Developer Environment

Developers can use any Java Based development environment.

1 https://app.swaggerhub.com/apis/a00908270/
2 https://github.com/a00908270/

4 Specification

4.1 Use Case

Figure 4.1 shows the UML use case diagram.

4.1.1 Use Case Descriptions

Use Case	**Import Neural Network**
Description	An existing ViNNSL XML file with a neural network description is imported via the vinnsl web service into the database.
Priority	primary
Actors	Data Scientist
Preconditions	ViNNSL neural network XML description file
Postconditions	—
Normal Course of Events	* The actor sends a POST request to the ViNNSL web service including a XML body* The web service validates and imports the XML file and returns the HTTP status code 201 CREATED
Alternative Courses	* The post request is sent by an application or other service
Exceptions	If the validation fails or an error occurs, the web service returns the HTTP status code 500
Assumptions	Access to the `vinnsl-service`

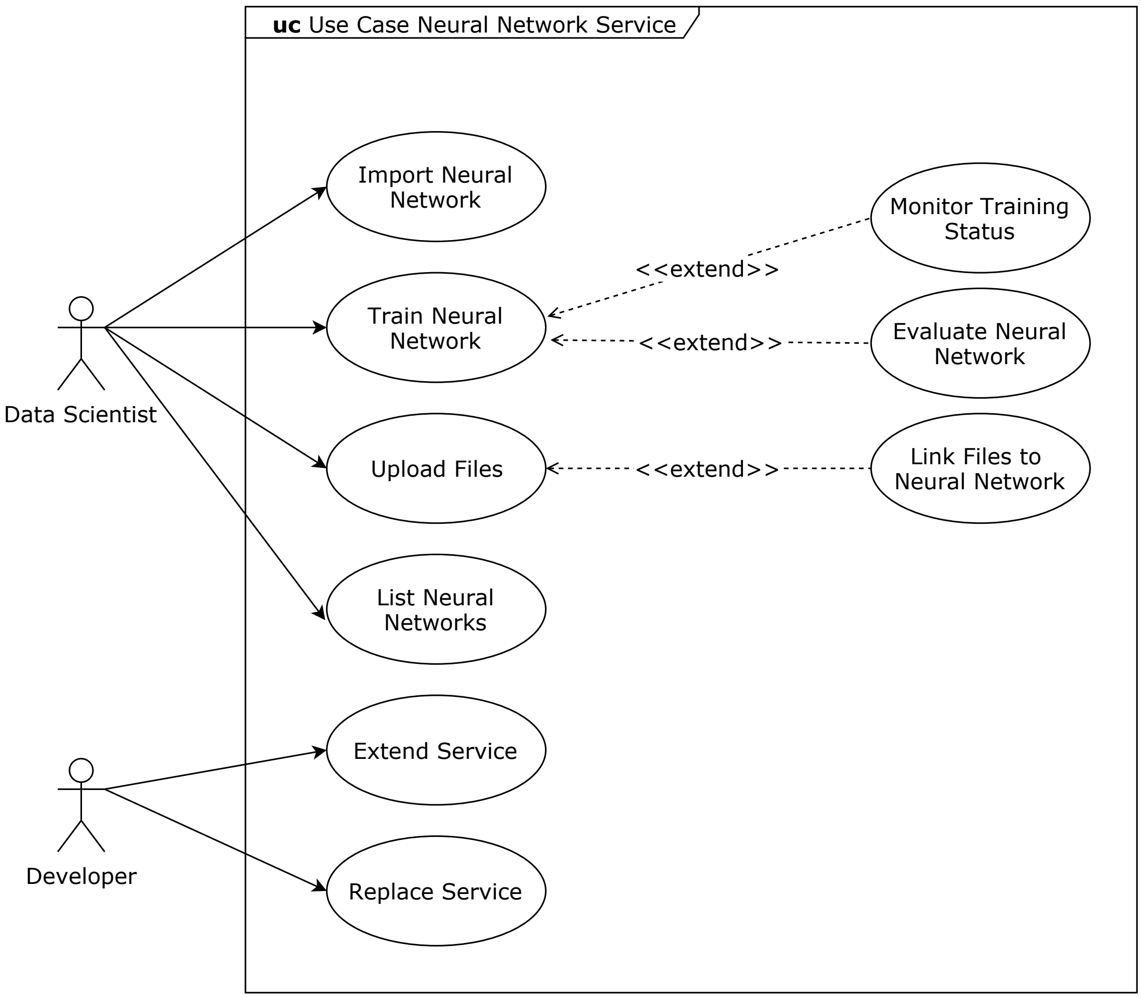

Figure 4.1: UML Use Case Diagram

Use Case	**Train Neural Network**
Description	An imported neural network is trained by passing the configuration over to the worker service.
Priority	primary
Actors	Data Scientist
Preconditions	Imported ViNNSL neural network XML description, definition and instance file
Postconditions	—
Normal Course of Events	* The actor sends a POST request to the working service including the identifier of the neural network that should be trained* The webservice validates the request, adds the network into the training queue and returns the HTTP status code 200.
Alternative Courses	* The post request is sent by an application or other service
Exceptions	If the validation fails or an error occurs, the webservice returns the HTTP statuscode 500
Assumptions	Access to the `vinnsl-nn-worker`
Extensions	* Monitor Training Status * Evaluate Neural Network

Use Case	**Monitor Training Status**
Description	The Data Scientist monitors the training status to evaluate the trained network afterwards.
Priority	secondary
Actors	Data Scientist
Preconditions	Training of neural network started
Postconditions	—

Use Case	Monitor Training Status
Normal Course of Events	* The actor sends a GET request to the status endpoint of the vinnsl service including the identifier of the neural network that is in progress.* The web service validates the request, and returns the training status along the HTTP status code 200.
Alternative Courses	* The post request is sent by an application or other service
Exceptions	If the validation fails or an error occurs, the web service returns the HTTP statuscode 500
Assumptions	Access to the `vinnsl-service`
Extensions	—

Use Case	Evaluate Neural Network
Description	The Data Scientist evaluates the accuracy of the network after its training
Priority	primary
Actors	Data Scientist
Preconditions	Training of neural network successfully finished
Postconditions	—
Normal Course of Events	* The actor sends a GET request to the status endpoint of the vinnsl service including the identifier of the neural network that is finished.* The web service validates the request, and returns the ViNNSL XML file including the result scheme.
Alternative Courses	* The post request is sent by an application or other service
Exceptions	If the validation fails or an error occurs, the webservice returns the HTTP statuscode 500
Assumptions	Access to the `vinnsl-service`
Extensions	—

Use Case	Upload Files
Description	The Data Scientist uploads files, that are usable as datasets (f.ex. CSV files or pictures) to the storage service
Priority	primary
Actors	Data Scientist
Preconditions	—
Postconditions	—
Normal Course of Events	* The actor sends a POST request to the storage service endpoint containing a multipart file.* The web service validates the request, and returns the unique identifier of the file along the HTTP status code 200.
Alternative Courses	* The post request is sent by an application or other service* The file is uploaded with the provided HTML upload form provided by the storage service
Exceptions	If the upload fails or an error occurs, the web service returns the HTTP statuscode 500
Assumptions	Access to the `vinnsl-storage-service`
Extensions	—

Use Case	**List Neural Networks**
Description	Imported neural networks are listed
Priority	primary
Actors	Data Scientist
Preconditions	Imported ViNNSL neural network XML description file
Postconditions	—
Normal Course of Events	* The actor sends a GET request to the ViNNSL web service optionally including a neural network identifier* The web service validates and returns the XML file(s).
Alternative Courses	The request is sent by an application or other service

Use Case	**List Neural Networks**
Exceptions	If the validation fails or an error occurs, the web service returns the HTTP statuscode 500
Assumptions	Access to the `vinnsl-service`

Use Case	**Extend Service**
Description	An existing micro service can be extended by developers
Priority	secondary
Actors	Developer
Preconditions	source code and developer environment present
Postconditions	—
Normal Course of Events	* The developer downloads the source code and extends functionality of a micro service* The modified service is deployed into kubernetes
Alternative Courses	—
Exceptions	—
Assumptions	—

Use Case	**Replace Service**
Description	An existing micro service can be replaced by developers
Priority	secondary
Actors	Developer
Preconditions	source code and developer environment present
Postconditions	—
Normal Course of Events	* The developer writes a new implementation of an existing service respecting the API definition (see API Docuentation)* The service is deployed into kubernetes
Alternative Courses	—

Use Case	**Replace Service**
Exceptions	—
Assumptions	—

4.2 Sequence Diagram

Figure 4.2 shows the sequence diagram of a neural network training process and which microservices are involved in the communication. The *vinnsl service* is the main communication hub that enables access to the neural network object and all of its data and also provides interfaces to update it. The *vinnsl storage service* most importantly stores necessary binary data used by the neural network objects. On one hand that are tables and pictures on the other hand the binary (trained) *Deeplearning4J* model. The *vinnsl worker service* has the role of training the neural networks models.

4.2.1 Sequence of Training

New neural networks are created by sending a `POST` request including a XML ViNNSL network description in the request body. The *vinnsl service* creates a new neural network based on the definition and answers with the HTTP status code 201 (CREATED). The location header points to the URL where the created network can be retrieved. The URL contains the unique identifier. Using this identifier the next step is to add the ViNNSL definition XML file to the network. This is done via a `POST` request appending the id and the `/definition` endpoint. The XML file is placed in the request body. Resources that are required for the training (like the training set) need to be uploaded to the storage service, which returns a unique file id. Before the training can start, the training set needs to be linked to the neural network. This is possible with the `/addfile` endpoint.

Next the network is marked for training by calling the worker service with its identifier. The worker service confirms that the training is queued. As soon as the training is finished, the worker service updates the neural network object with the result schema and uploads the trained binary model to the storage service for retraining.

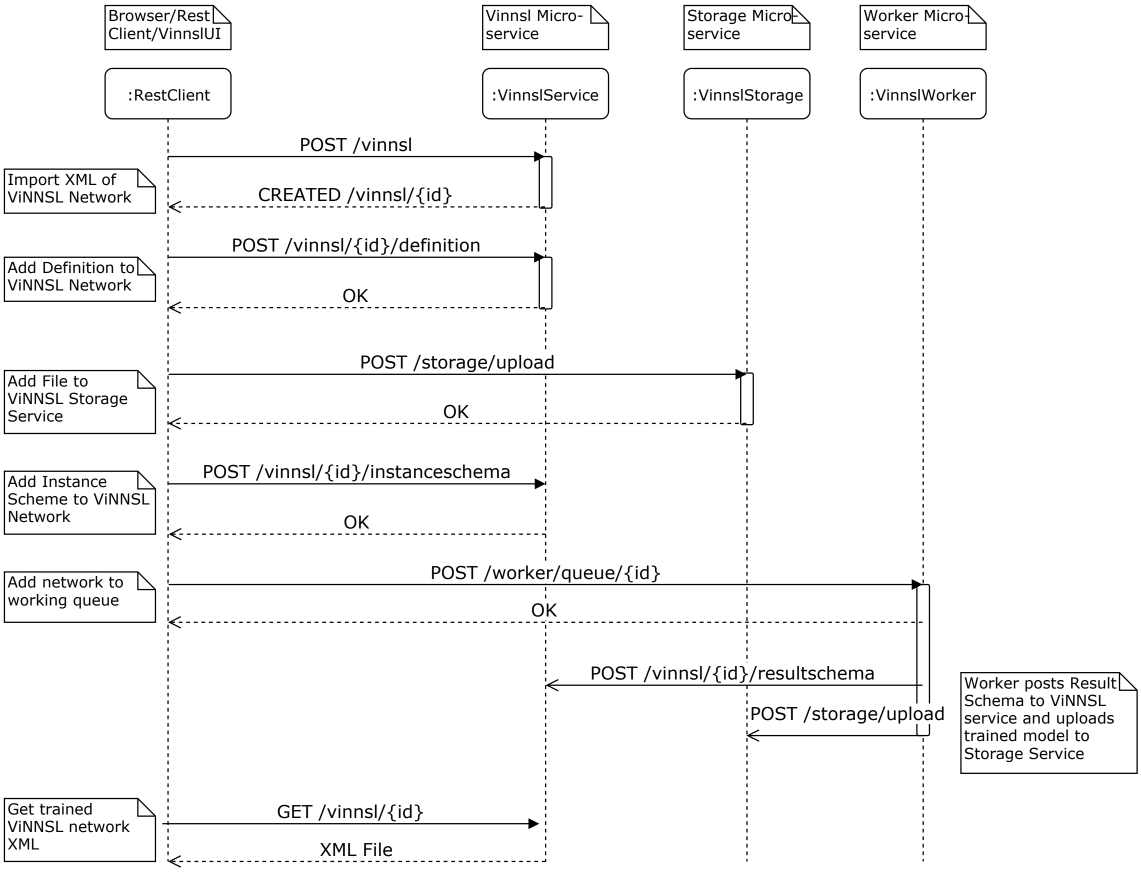

Figure 4.2: Training Sequence Diagram

A simple GET request to the vinnsl service along with the identifier returns the current trained neural network model.

4.3 Data Model Design

4.3.1 vinnsl-service

All neural network data managed by the `vinnsl-service` is stored in a documented-oriented database. The saved documents will internally be mapped to Java Classes. The main object is `vinnsl`.

`vinnsl` is the primary object owning the `_id` field that is unique. The `nncloud` property stores the status of the network and the representation of the transformed *Deeplearn-*

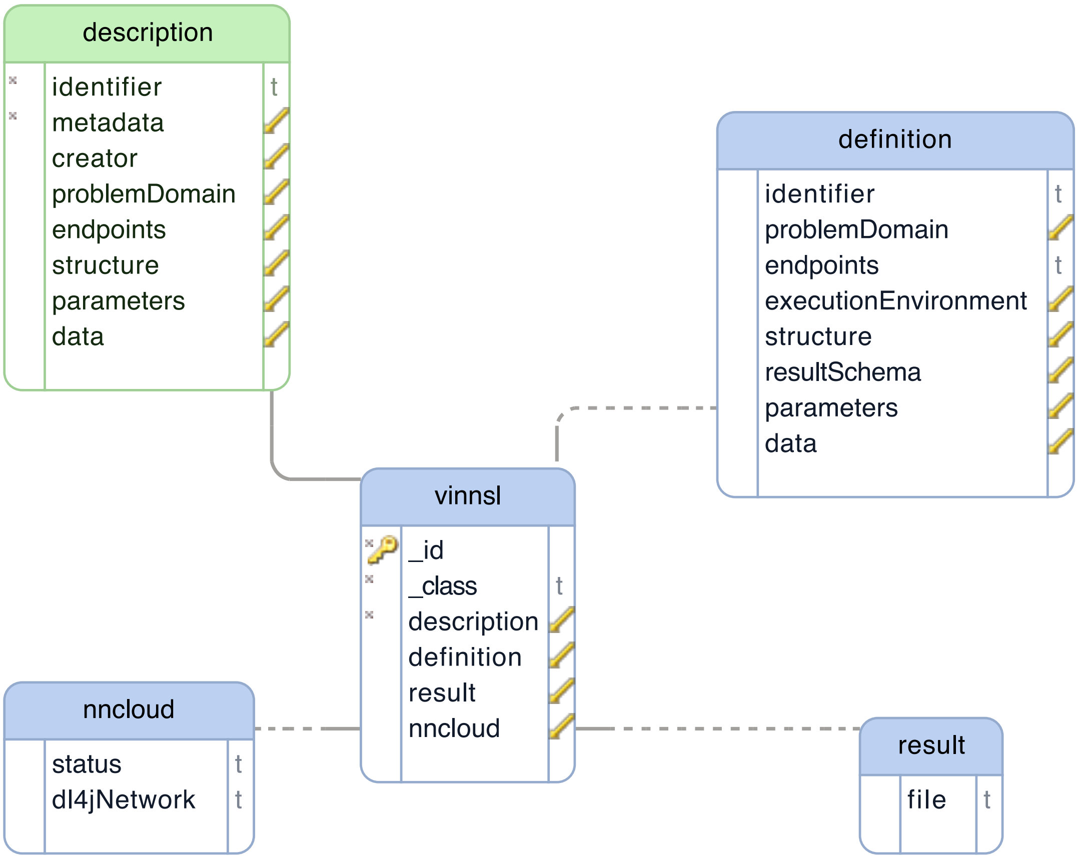

Figure 4.3: NoSQL Data Model

ing4J network. `description`, `definition`, `instance`, `training` and `result` represent the ViNNSL 2.0 Schema, generated from the provided XML Schema Definition files. See [Kop15] to get a listing and description on all provided properties of ViNNSL 2.0.

Figure 4.3 shows the data schema.

4.3.2 storage-service

The `storage-service` stores binary files and their metadata, either directly in the file system or inside a database. Each file needs to have a unique id, a filename, a content type and an upload date.

Attribute field	Description
id	a unique file id that can be referred to (f.ex in `vinnsl-service`)
filename	the original filename when uploaded
content type	the MIME type standardized in RFC 6838 (f.ex text/plain)
upload date	date and time of original upload
metadata	a field for arbitrary additional information

Example of stored file:


```
{
    "_id" : ObjectId("5ab4e69c8f136a16bf81f093"),
    "filename" : "iris.txt",
    "aliases" : null,
    "chunkSize" : NumberLong(261120),
    "uploadDate" : ISODate("2018-03-23T11:35:56.700Z"),
    "length" : NumberLong(2700),
    "contentType" : "text/plain",
    "md5" : "f0e89bd71f7bb9e584e685aeb178a5aa"
}
```


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 *Deeplearning4J* models. It provides a wrapper of the *Deeplearning4J* 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`.

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.

1 Create, Read, Update, Delete

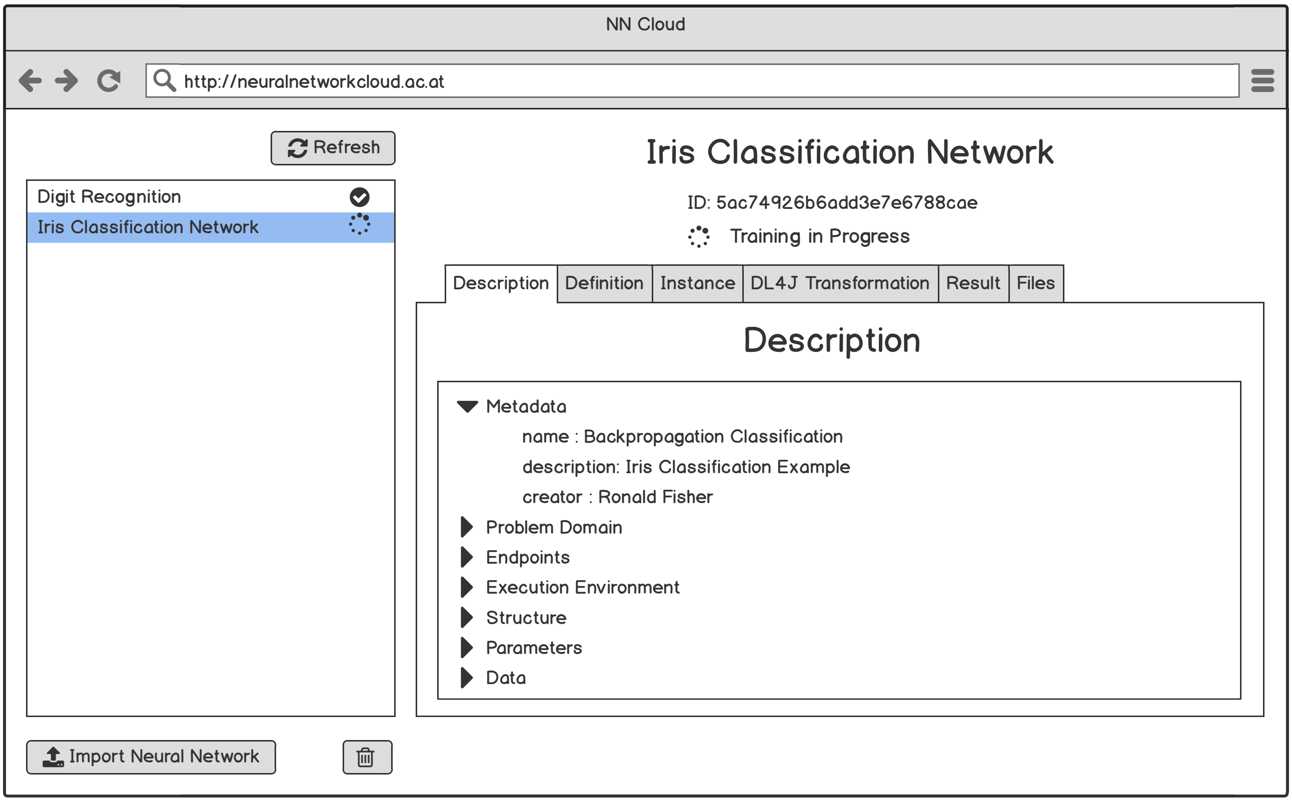

Figure 4.4: User Interface Design for vinnsl-nn-ui

4.5 User Interface Design

Based on the mockup in section 3.1.1, a user interface design has been created, that will later be implemented as a web application. Buttons to import, delete a neural network and to refresh the user interface have been added to the design.

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

4.6 Service Discovery and Load Balancing

Service Discovery is the process of finding out how to connect to a specific service. This applies within the cluster, which is typically firewalled from the internet. As Kubernetes allows services to be scaled, there is also a logic that knows and decides how network traffic is routed. This is called *Load Balancing*. Figure 4.5 shows an overview of the microservices, their endpoint URL and the domain name service. External access to specific services is managed by *Ingress*.

4.6.1 Kubernetes DNS-based Service Discovery

`kube-dns` is the Kubernetes add-on that starts a pod with a DNS service and configures the kubelets to resolve DNS names over this service. It listens on port 53, the standard DNS port. Services in a cluster are assigned a DNS A record derived from their service metadata name specified in the *ServiceSpec*. [Kubd]

The following code snippet is an extract of the *ServiceSpec* for the `vinnsl-service` defining the metadata name:

```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "vinnsl-service",
    ...
  }
}
```

Structure of the Hostname

The full hostname record is composed of the zone, kind, namespace of the cluster and the metadata name of the service.

Name	Description
zone	the cluster domain (default using minikube: *cluster.local*)
kind	kind of pod (default for services: *svc*)
ns	namespace (default using minikube: *default*)
hostname	hostname from service metadata name

Example The vinnsl-service running on a local minikube cluster gets the following DNS record name: `vinnsl-service.default.svc.cluster.local`.

Service Discovery

Using the Kubernetes DNS service a microservice instance (kubelet) can now lookup other services by using DNS Queries.

Example For example the tool `nslookup` can query the DNS service for the IP address of the `vinnsl-service` within the cluster.

```
/ # nslookup vinnsl-service
Server:     10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:       vinnsl-service
Address 1: 10.102.84.122 vinnsl-service.default.svc.cluster.local
```

In this example the service is reachable at the IP address *10.102.84.122*.

External Access and Load Balancing

External Access from outside the cluster to specific services is managed and provided through the *Ingress* API object. The associated implementation is called *Ingress controller* and is obligatory. Currently there are two official implementations: `ingress-gce` and `ingress-nginx`. [Kuba]

Minikube runs the `ingress-nginx` implementation as default and also provides basic load balancing by configuring a `nginx` [2] web server. Kubernetes configures `nginx` to use the *least-connected* load balancing mechanism, which means that the *next request is assigned to the server with the least number of active connections* [ngi].

2 http://nginx.org/

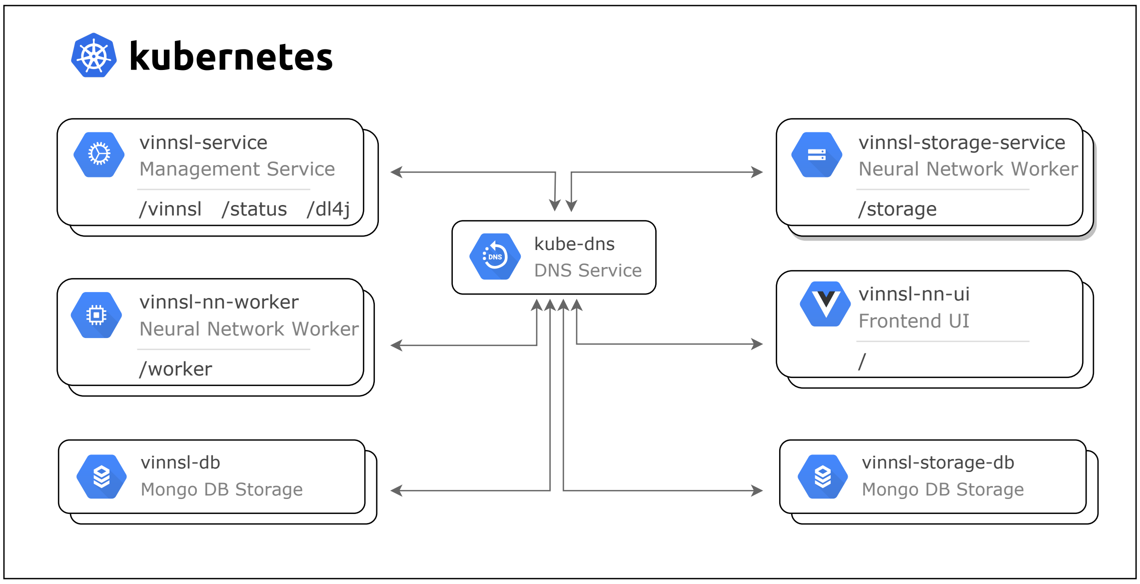

Figure 4.5: Service Discovery with kube-dns

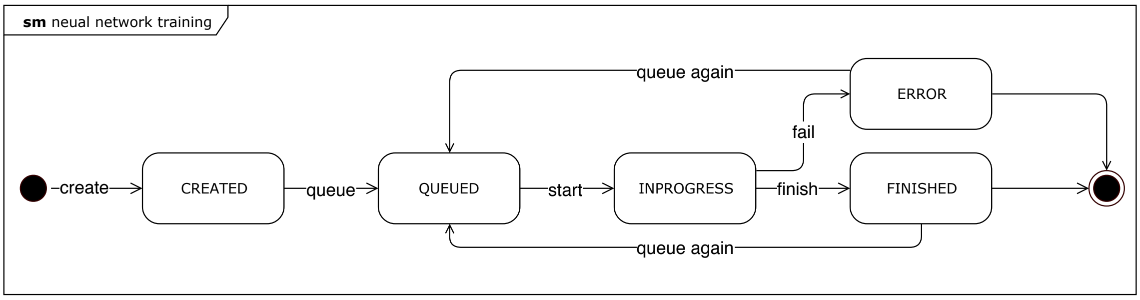

Figure 4.6: State Machine of a Neural Network

4.7 Neural Network Objects State

The state of neural network objects is saved in the `NnCloud` object. When the object is instantiated the default value is `CREATED`. When the network is queued, the worker service gathers all the necessary data from the vinnsl and vinnsl storage service and changes the state the `QUEUED`. During the network training, the worker changes the state to `INPROGRESS`. As soon as the training is finished, the worker service uploads the results and updated network state to the storage service and subsequently changes the state to `FINISHED`. Trained networks can be queued for retraining: in that case the state returns to `QUEUED`. If errors occur during the training process the state will be set to `ERROR`.

Figure 4.6 visualizes the state changes in a state machine.

5 Prototype Implementation

Following the specification, this section showcases an implementation of a prototype using microservices glued together by *Kubernetes*. This represents the execution stack for neural networks. Backend components are realized with *Java* and the *Spring Boot* framework and expose a RESTful API. The processing and training of neural networks is done by the *Deeplearning4J* framework. Database and file storage are powered by *MongoDB*. The frontend service is implemented using *Vue.js* and the *Twitter Bootstrap* UI framework, visualizing and consuming backend services.

5.1 Source Code

The source code of the implemented microservices is released on *GitHub*. The following table gives an overview of available services and their corresponding repository.

Name	Repository Link
vinnsl-service	https://github.com/a00908270/vinnsl-service
vinnsl-nn-ui	https://github.com/a00908270/vinnsl-nn-ui
vinnsl-storage-service	https://github.com/a00908270/vinnsl-storage-service
vinnsl-nn-worker	https://github.com/a00908270/vinnsl-nn-worker

The *ViNNSL* XSD schema specified in [Kop15] including (generated) examples is released on GitHub with permission from Dipl.-Ing. Thomas Kopica. JAXB class generation of the XML files is already included in the release with the intention of making it easier to include *ViNNSL* into new services.

Name	Repository Link
vinnsl-schema	https://github.com/a00908270/vinnsl-schema

5.2 Releases

Docker Contrainers ready for deployment in a *Kubernetes* cluster are released on *DockerHub*. The following table references the released repositories.

Name	Repository Link
vinnsl-service	https://hub.docker.com/r/a00908270/vinnsl-service/
vinnsl-nn-ui	https://hub.docker.com/r/a00908270/vinnsl-nn-ui/
vinnsl-storage-service	https://hub.docker.com/r/a00908270/vinnsl-storage-service/
vinnsl-nn-worker	https://hub.docker.com/r/a00908270/vinnsl-nn-worker/

5.3 Framework Dependencies

All services are written in *Java* and build using the *Apache Maven* build automation and dependency management tool.

5.3.1 Spring

Spring is a *Java* framework consisting of many modules, most importantly this project uses its feature so set up `RestController` instances that listen on specified endpoints.

Used in following services: `vinnsl-service`, `vinnsl-nn-ui`, `vinnsl-storage-service`, `vinnsl-nn-worker`

Spring Boot

Spring Boot is an extension to the framework that allows *Java* applications to run stand-alone by embedding a web server directly into the application. [Spra]

Used in following services: `vinnsl-service`, `vinnsl-nn-ui`, `vinnsl-storage-service`, `vinnsl-nn-worker`

Spring Data MongoDB

Spring Data provides an abstracted database access layer to MongoDB in form of a POJO (Plain Old Java Object). [Sprb]

Used in following services: `vinnsl-service`, `vinnsl-storage-service`

5.3.2 Swagger

Swagger is used to generate a live documentation of all web service endpoints in this project and allows to try out requests directly in the user interface.

Used in following services: `vinnsl-service`, `vinnsl-storage-service`, `vinnsl-nn-worker`

5.3.3 Fabric8

Fabric8 packs the generated executables from the build process into a *Docker* container that can run in a *Kubernetes* cluster. The process is described in detail in section TODO

Used in following services: `vinnsl-service`, `vinnsl-nn-ui`, `vinnsl-storage-service`, `vinnsl-nn-worker`

5.3.4 Deeplearning4J

Deeplearning4J is used by the worker service to train and evaluate neural networks.

A detailed introduction to Deeplearning4J can be found in Section 2.4.2.

Used in following services: `vinnsl-nn-worker`

5.4 Security

Ingress supports HTTPS encrypted connections. Authentication or restrictions are not implemented in the prototype.

5.5 User Interface

5.5.1 vinnsl-nn-ui (Frontend UI)

The `vinnsl-nn-ui` is a single page application (SPA) that displays all neural networks and their details in a web based frontend. Figure 5.1 shows a screenshot of the user interface.

Architecture

The web application is a Javascript based frontend using the *Vue.js* and *Twitter Bootstrap* framework. The single main controller called `VinnslUI` provides methods to fetch a list of neural networks and their status. Additionally it queries for available files from the storage service and enables to connect them to a neural network.

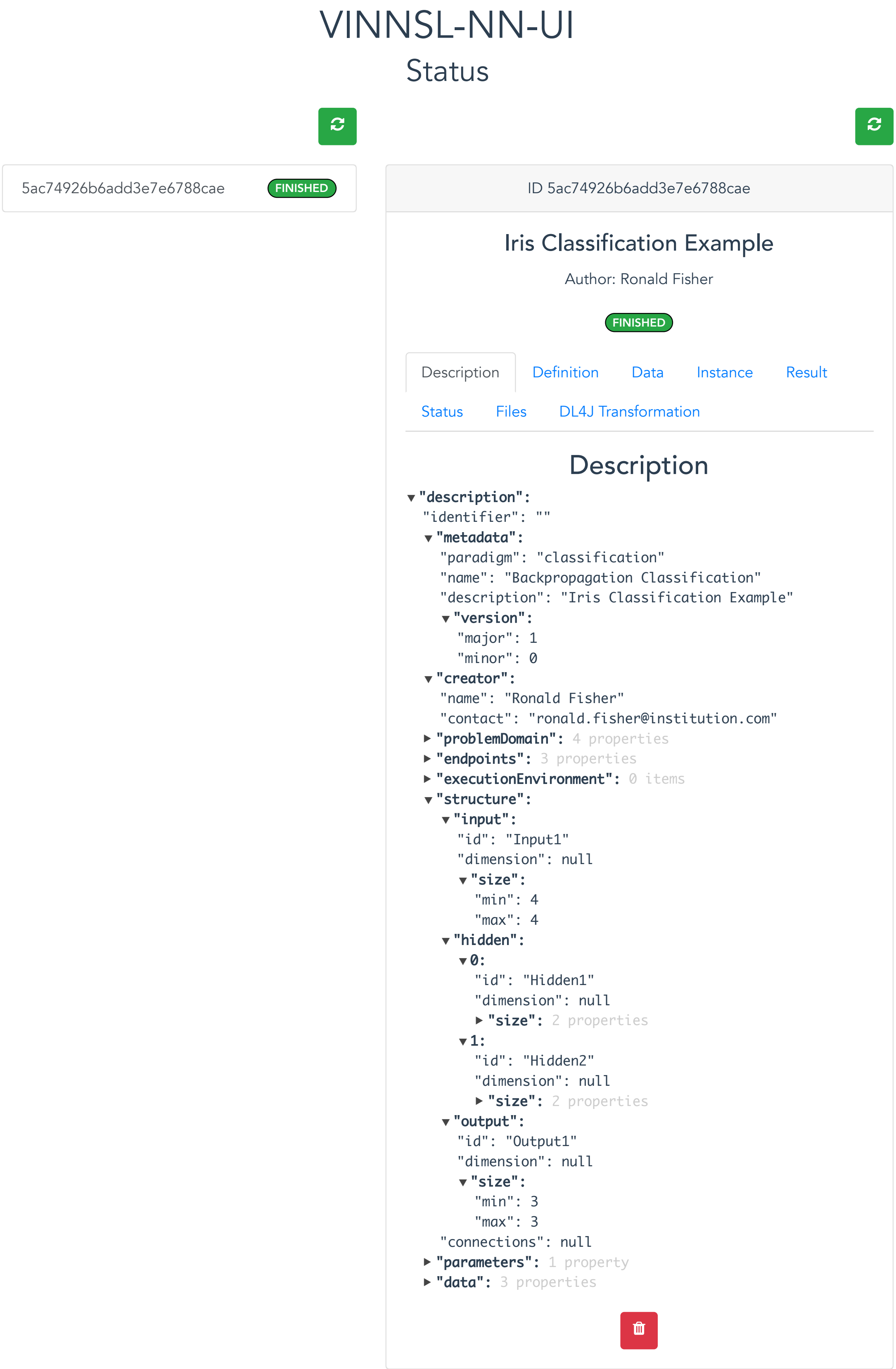

Figure 5.1: User Interface of Prototype

5.6 Class Diagram

5.6.1 vinnsl-service

5.6.2 vinnsl-storage-service

5.6.3 vinnsl-worker-service

5.6.4 vinnsl-nn-ui

The frontend service consists of one single controller named `VinnslUI`. The `getStatus()` method retrieves all and neural network ids and their status. This is cached in `vinnslList` . When selecting a neural network from the list, the neural network object is loaded by executing `getDetailsById()`. The response is stored in `currentVinnslItem`.

Figure 5.5 gives an overview of the used methods and stored variables.

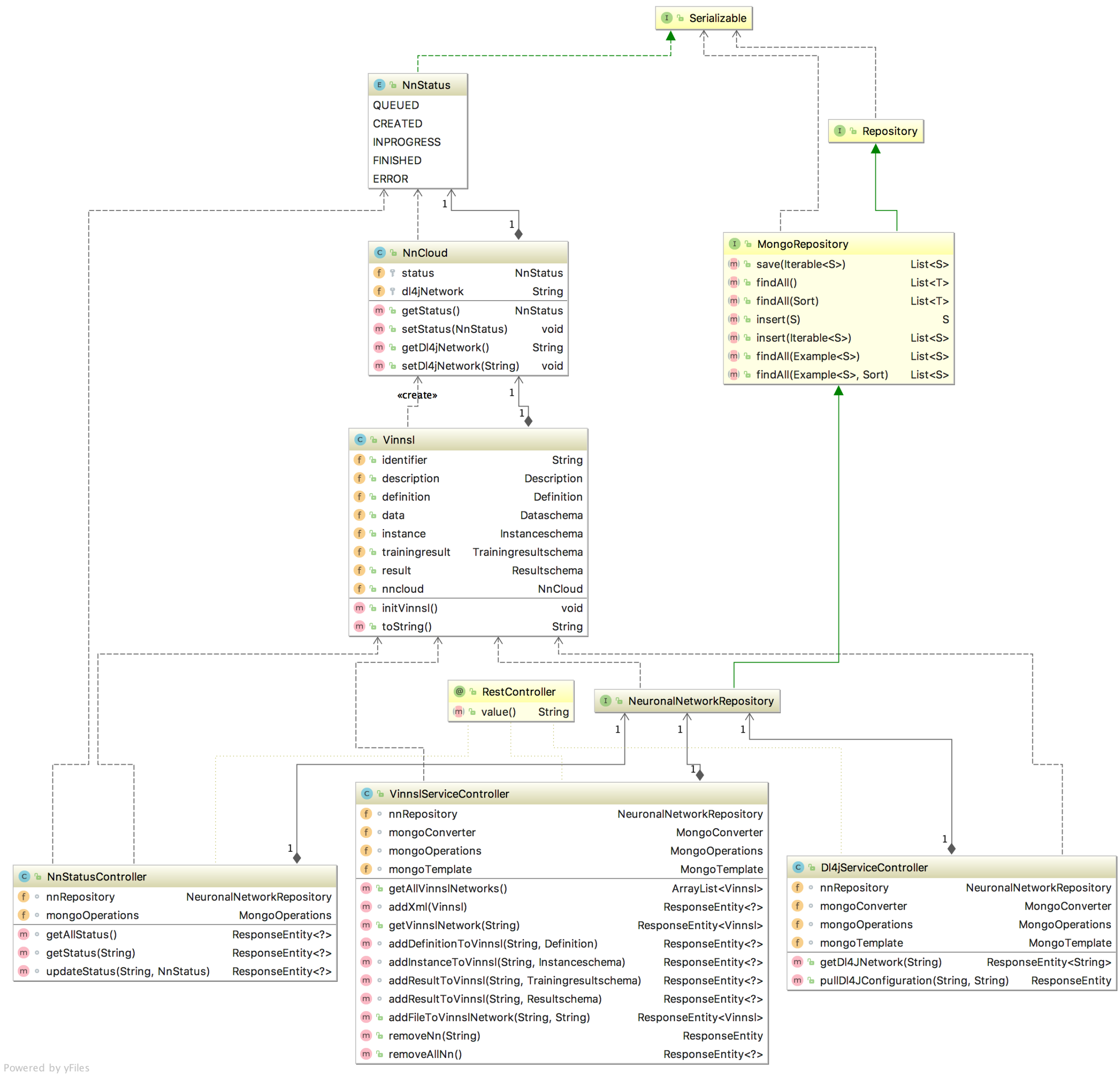

Figure 5.2: Class Diagram of vinnsl-service

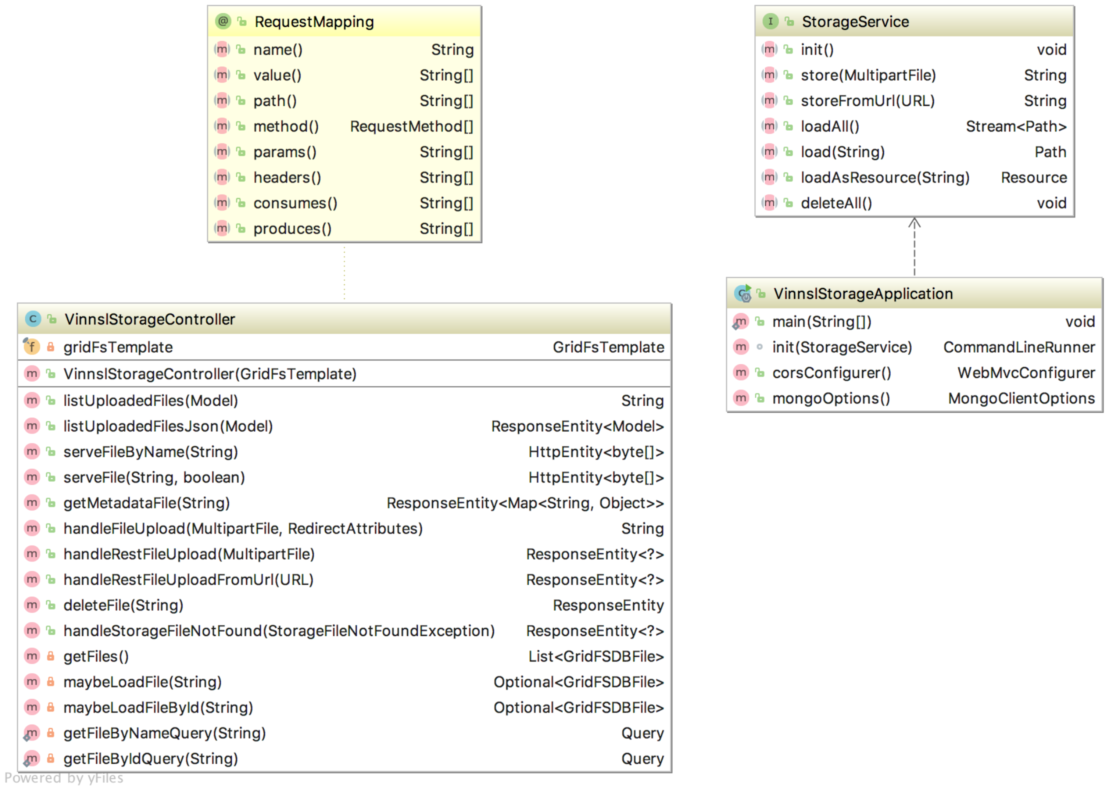

Figure 5.3: Class Diagram of vinnsl-storage-service

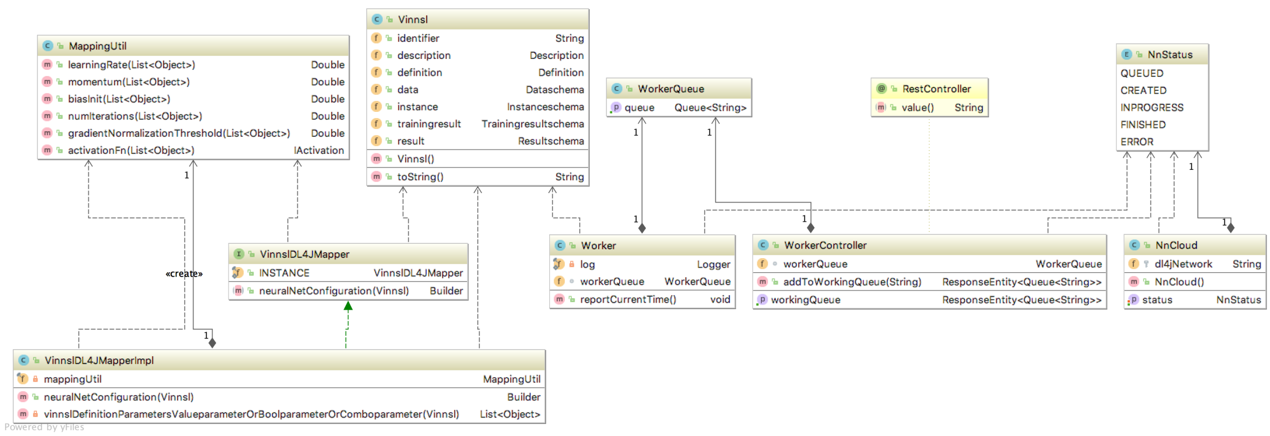

Figure 5.4: Class Diagram of vinnsl-worker-service

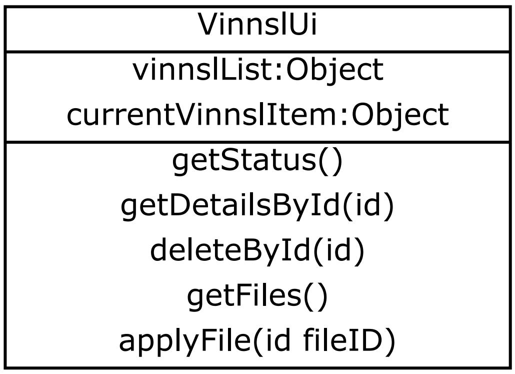

Figure 5.5: VinnslUI Vue Class

6 Prototype API Documentation

Base URL

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

6.1 vinnsl-service

6.1.1 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>
```

```xml
                <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
```

6.1.2 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": "{}
        }
    },
```

```
    ...
]
```

6.1.3 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

6.1.4 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>
```


6.1.5 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

6.1.6 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

6.1.7 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>
```

6.1.8 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>
```

6.1.9 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>
```

6.1.10 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>
```

6.1.11 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"
}
```


6.1.12 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

6.1.13 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

6.1.14 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

6.1.15 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

6.2 vinnsl-storage-service

6.2.1 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

6.2.2 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

6.2.3 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

6.2.4 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

6.2.5 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

6.2.6 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

6.2.7 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

6.2.8 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

6.3 vinnsl-worker-service

6.3.1 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

6.3.2 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

7 Use Cases

As demonstration of the implemented prototype this thesis features two use cases with practical relevance.

7.1 Iris Classification Example

Ronald A. Fisher published 1936 in his paper *The use of multiple measurements in taxonomic problems* [Fis] a dataset that is known as the *Iris flower data set*.

The data set [Fis] features 50 examples of three Iris species: Iris setosa, Iris virginica and Iris versicolor. A table lists four measured features from each sample: the length and the width of the sepals and petals.

This use case shall showcase the use of the implemented prototype to create a neural network, train and evaluate it, using this dataset.

7.1.1 Prerequisites

- Minikube installed and running
- Services from the Neural Network Execution Stack deployed in cluster
- Hostname `cluster.local` resolves to Minikube instance

7.1.2 Create the neural network

Request

```
POST https://cluster.local/vinnsl
```

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>
    <!--<valueparameter>learningrate</valueparameter>
```

```
      <valueparameter>biasInput</valueparameter>
      <valueparameter>biasHidden</valueparameter>
      <valueparameter>momentum</valueparameter>
      <comboparameter>ativationfunction</valueparameter>
      <valueparameter>threshold</valueparameter>-->
    </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>
```

Response

```
201 CREATED
```

7.1.3 Add ViNNSL Description to the neural network

Request

```
POST https://cluster.local/vinnsl/{id}/definition
```

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>
```

Response

```
200 OK
```

7.1.4 Queue Network for Training

Request

`POST https://cluster.local/worker/queue/{id}`

Response

`200 OK`

7.1.5 Evaluate

TODO

7.2 Hosted trained network

TODO

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

12.1 Deploy Neural Network Execution Stack

12.1.1 Local Machine

TODO

12.1.2 Cloud Instance

TODO

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

[BGO+16] Burns, Brendan; Grant, Brian; Oppenheimer, David; Brewer, Eric; Wilkes, John: Borg, Omega, and Kubernetes. In: *Communications of the ACM* 59 (2016), apr, Nr. 5, 50–57. `http://dx.doi.org/10.1145/2890784`. – DOI 10.1145/2890784

[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`

[Fis] Fisher, Ronald A.: The Use Of Multiple Measurements In Taxonomic Problems. In: *Annals of Eugenics* 7, Nr. 2, 179-188. `http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x`. – DOI 10.1111/j.1469–1809.1936.tb02137.x

[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

[Kuba] Kubernetes Authors, The: *Ingress*

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

[Kubc] Kubernetes Authors, The: *Kubernetes Concepts - Pods*. `https://kubernetes.io/docs/concepts/workloads/pods/pod/`, Abruf: 2018-05-12

[Kubd] Kubernetes Authors, The: *Kubernetes DNS-Based Service Discovery*. `https://github.com/kubernetes/dns/blob/master/docs/specification.md`, Abruf: 2018-05-31

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

[ngi] nginx: *Using nginx as HTTP load balancer*. `http://nginx.org/en/docs/http/load_balancing.html`, Abruf: 2018-05-31

[Por] Portworx: *Portworx Annual Container Adoption Survey 2017*

[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

[Spra] Spring: *Spring Boot*. `https://spring.io/projects/spring-boot`, Abruf: 2018-06-03

[Sprb] Spring: *Spring Data MongoDB*. `https://projects.spring.io/spring-data-mongodb/`, Abruf: 2018-06-03

[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