CHRISTIAN NÖSTERER, BSc

Datawiz - A Scalable Sensor Data Framework

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Roman Kern

Knowledge Technologies Institute
Head: Univ.-Prof. Dr. Stefanie Lindstaedt

Graz, August 2019

Affidavit

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 indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Abstract

Nowadays there are more and more devices that are being connected to the internet, therefore it is important to provide a reliable bridge between them. Gathering/Routing the data is the foundation for many different business processes and is therefore highly important.

The goal of this thesis was to build a scalable infrastructure for sensor data that only uses open source components and is easy to use for users who provide sensor data. To make this system scalable, different container orchestrators were evaluated.

As a basis, the container orchestration tool Kubernetes was chosen. Additional system components for system maintenance were selected to improve the maintainability. Further components include a load-balancer, certificates for secure communication and monitoring. For the persistence of data, a solution was evaluated and included. The platform can be deployed to different IaaS providers via a Terraform script.

The web UI for users and application management is written in Java and based on the high performance web framework Vert.x. The performance was evaluated using current web frameworks as a reference point. Applications from categories such as data input, data output and data computation/processing can be consumed by users. For every application category there is at least one reference application configured.

On the data input category available MQTT servers were tested in regards to performance and the best suitable server solution was selected. The data output layer was evaluated and the best databases were used. For the data computation layer a HSTM based computational intelligence library was selected to showcase inter-connectivity between the components. The framework is extensible to include new applications to provide additional functionality to the users of the system.

The system was tested in full action with two sensor types for input and output. Additional hardware sensors can be included by providing a template and base-values. Code can then be uploaded to these sensors, based on the values the user provided. Thus the developed system allows and facilitates the setup of a full-blown scalable sensor data framework on multiple cloud provider.

Contents

List of Figures

1 Background

1.1 Internet of Things

As the internet is growing exponentially more and more devices are being connected. In June 2019 it was estimated that 57.3 percent (=4,4 Billion people) of the world's population are already connected [14]. To further connect technologies like artificial intelligence with the physical world, more and more sensor data devices are necessary. It is estimated that in 2020, nearly 31 billion devices will be connected to the internet globally [14].

Unfortunately as of August 2019, many of these devices cannot be interconnected and can only be accessed through their cloud provider. Examples include the AWS IoT button [1], which can only connect to the AWS cloud, Netatmo Smart Thermostat [2], which only works with Apple Cloud, Google and Alexa. Further examples are The Honeywell thermostat [3], which uses a proprietary cloud service and the smart cams from Xiaomi [4], which exclusively communicates through their mobile app with their cloud service in China.

Since all these sensors and actuators use different service management and updates will become extremely complicated. That means it is impossible to use a different data collector service nor interact directly in the data flow. Some devices, like the Hunter Hydrawise [5] offer API access, but this further

[1]https://aws.amazon.com/de/iotbutton/ (Accessed on: 2018-09-30)
[2]https://www.netatmo.com/de-de/energy/thermostat/ (Accessed on: 2018-09-30)
[3]https://www.honeywellstore.com/store/category/thermostats.htm (Accessed on: 2018-09-30)
[4]https://www.mi.com/us/index.html (Accessed on: 2018-09-30)
[5]https://www.hydrawise.com/ (Accessed on: 2018-09-30)

complicates IoT connectivity as data flow is not direct and every device uses different API access methods.

The current world wide web was built upon open standards, which fueled its growth and collaboration between different companies/services.

There will be over 75 billion devices connected in 2025 [18], these challenges have to be solved. These devices will generate a massive amount of traffic and it will be important, that the data collector scales well and is preferably open source.

Despite these clustered environments, and since some devices may contain sensitive private data they will have to comply with data regulations, like the DSGVO from Europe [12]. However this does not affect all sensor data, since only personalized data is subject to it. As it is likely that more and more health-related data will be gathered, security of data transportation has to be considered.

Even though most proprietary devices do not provide open communication, there are promising communication protocols emerging. This leads to two important concepts of how the device is managed and how the data is transported.

1.1.1 Data Transportation

Data transportation is the means of how data passes from a source (sensor,..) to the destination (usually a server). The transportation can be uni-directional or bi-directional.

This means the TCP[6] connection is closed and reopened again when the new data is ready to be sent. (usually after a delay). This is not in real-time and often more advanced protocols are necessary to enhance performance and stability and keep the protocol overhead even smaller. Uni-directional protocols are for example HTTP[7] via a REST [8] request.

[6]https://tools.ietf.org/html/rfc793 (Accessed on: 2018-09-30)
[7]https://tools.ietf.org/html/rfc2616 (Accessed on: 2018-10-04)
[8]https://tools.ietf.org/html/rfc7231 (Accessed on: 2018-10-10)

There are more lightweight protocols which are meant to combat these issues and can also provide bi-directional communication. The underlying transportation layer is different and can be Bluetooth, TCP or any other means of transmission. Protocols like Bluetooth provide some features, like bluetooth Low Energy (BLE) while still needing a gateway on edge to act as a broker for them. Still if sensor data does not need to be bi-directional it can be very useful as it needs fewer memory and less energy consumption, which can be critical for current embedded devices because they are usually thin in both. Notable mentions, which provide the transportation on the wireless layer include ZigBee and BLE. There are several protocols that are based on TCP, such as Message Queue Telemetry Transport (MQTT)[9] and Constrained Application Protocol (CoAP)[10]. Several protocols are suitable for IoT sensor data like MQTT, CoAP or more complete protocols like IoT Bacnet [11]. These protocols are based on the pub/sub model, where a client/server can subscribe to various topics and clients can publish data on these topics. These are more suitable protocols for IoT applications since they additionally can guarantee data delivery through various service levels. CoAP for example has two message types confirmable and non-confirmable, while MQTT has several service levels according to QoS. But while CoAP relies on a one-to-one communication between a server and a client, this is in contrast to MQTT, as it can be used with many clients and servers.

1.1.2 MQTT

As MQTT relies on the pub/sub model, a client/server can subscribe to various topics and clients can publish data on these topics. An example of this pattern can be seen in Figure 1.1, where three clients connect to a broker. In this example client one publishes sensor data and receives commands whereas client two only subscribes and client three can actively issue commands based on the topics. The message payload however has to be negotiated upfront, whereas in CoAP they can be exchanged later.

[9] `http://docs.oasis-open.org/mqtt/mqtt/v5.0/cs02/mqtt-v5.0-cs02.html` (Accessed on: 2019-05-10)

[10] `https://tools.ietf.org/html/rfc7252` (Accessed on: 2018-10-15)

[11] `http://dingo-iot.io/iot-bacnet` (Accessed on: 2019-05-10)

For MQTT there are three service levels:

- QoS 0 (At most once) - best effort delivery is not guaranteed.
- QoS 1 (At least once) - messages are assured to arrive but could be duplicated.
- QoS 2 (Exactly once) - messages arrive exactly once.

[12]

The latest version of MQTT 5.0 (Standard 02) was release on February 11, 2019 and has enhancement for scalability, and server disconnect notifications when there are problems regarding the connection. There is also an addition to MQTT sensor devices called MQTT-sn, which is more suitable for sensor devices. To also support devices, which do not rely on TCP for transportation, there is an addition to the protocol called MQTT-sn, where the sn stands for Sensor Network. This way it is possible to use UDP to do the transportation. Not all MQTT based brokers support MQTT-sn, but if devices such as Zigbee are needed inside the network, one must make sure that it is supported.

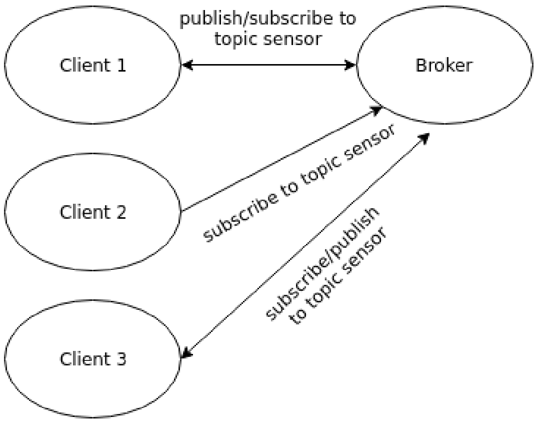

Client 1
publish/subscribe to topic sensor
Broker
Client 2
subscribe to topic sensor
Client 3
subscribe/publish to topic sensor

Figure 1.1: MQTT publisher/subscriber model example with three connected clients

In 1.1, the MQTT publisher/subscriber model is visible, with three connected clients.

[12]https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (Accessed on: 2019-05-10)

1.2 Orchestration

It was quite cumbersome to deploy an application to a production system, as different libraries/dependencies have to be considered. Portability is possible but takes a lot of space and often the entire Virtual machine (VM) with all it's additional services has to be exported. When making changes to a running VM, changes can be hard to track because it involves the whole operating system.

Often even a specific Linux distribution is necessary. Considering critical tasks such as updates and high availability, addressing these issues can get complicated to handle. To combat some of these issues mentioned above, fully virtualized solutions where used to provide isolation on various domains (data, storage, memory, CPU, network). Current major solutions are VirtualBox, Hyper-V, KVM, Xen and VMware.

High Availability and Fault Tolerance could be taken care of but migration from one system to another was still complex and costly as the running state of the application would have to be replicated.

With integration of cgroups into the Linux kernel in version 2.6.24, resource isolation for memory, CPU, disk and network were introduced. Further separation was achieved with Linux namespaces, which provide PID, network, mount, UTS and user name space isolation. Isolated applications which still can be run on the kernel are called containers. FreeBSD/chroot jails provided many of the features but where not distribution independent.

This development lead to Orchestration, which is the process for applications to run containers as an application packaging mechanism.

It should be a lightweight portable run-time, capable of developing, testing and deploying applications to a large number of servers and the capability to interconnect containers [25]. Furthermore It should be an abstracted pool of resources and applications.

Containers should run autonomously, should be self-healing and should not need any direction interaction. This is in contrast to previous applications, where the state is not declarative but imperative.

This is also the foundation of scalable software: Which Bondi, André B defined as:

> Scalability is the ability of a system to expand in a chosen dimension without major modifications to its architecture. Load scalability is the ability of a system to perform gracefully as the offered traffic increases. [7]

There are two types of scaling solutions vertical and horizontal. Whereas vertical scaling is the approach of adding more capacity to a resource, like CPU, memory, network. Horizontal Scaling is meant to add more physical machines to expand the capacity of a cluster. But in contrast to vertical scaling, orchestration has to be applied, since the load has to be managed and balanced.

It was described by Byers, Charles Calvin and Salgueiro, Gonzalo and Clarke and Joseph Michael[8] as:

> Orchestration is the process whereby the resources of a complex network are allocated, configured, and managed

In Orchestration systems, one declares how many containers should be running and not where, nor do you need to specify any hardware specifics. Constraints regarding CPU, network, memory can and should still be declared but they are not manually assigned. Therefore vertical scaling is more flexible as it expands beyond the capacity of a single machine.

It is in essence the pets vs cattle analogy, which was first mentioned by Bill Baker in his presentation "Scaling SQL Server" [5]. Pets are named and should be nursed back when unhealthy, whereas cattle is counted in numbers and can be shot when they become unhealthy.

In a presentation by Gavin McCance, named CERN Data Centre Evolution, on slide 17 Gavin states

> "future application architectures should use cattle but pets with strong configuration management are viable and still needed." [23]

Provider	**Searches**	**News**	**Stackoverflow**	**issues**	**stars**
Kubernetes	10.800.000	197.000	10.744	2.202	44.506
OpenShift	3.700.000	41.900	5.321	311	6.311
AWS ECS	2.310.000	9.050	1.072	-	-
Dcos+Mesos	1.860.000	6.590	1.474	-	6.027
Docker swarm	848.000	3.860	490	246	5.351
Nomad	53.100	297	46	397	4047

Table 1.1: Different Orchestration provider number of Google Searches[13], Google.com/news articles[14], Stackoverflow.com questions[15] and open Github.com issues[16]. This can provide the popularity and maturity of the several orchestrators. The data was collected on 14.9.2018.

Essentially this is one of the problems orchestration tries to solve, by maintaining its servers where cattles can die anytime and scale-up/down easily when necessary. VMs could be used to empathize this concept, but still portability, performance and resource allocation is more practicable utilizing containers. The popularity of current orchestration solutions can be seen in Table 1.1. This clearly shows that Kubernetes is the most popular solution. As this solution was used in building the open sensor data system, it will be described in detail in the following pages, starting with the underlying container solution, named Docker.

In Table 1.1 one can see different orchestration providers with their respective Google searches [13], number of Google news articles[14], asked questions on Stackoverflow [15], the number of open Github.com issues [16]and times it has been stared by developers on Github[17]. This gives a broad overview of their popularity as well as how mature the provider is.

[13]`https://www.google.com/search?q=keyword` (Accessed on: 2018-09-30)
[14]`https://www.google.com/search?q=keyword&tbm=nws` (Accessed on: 2018-09-30)
[15]`https://stackoverflow.com/questions/tagged/keyword` (Accessed on: 2018-09-30)
[16] `https://github.com/reponame/issues` (Accessed on: 2018-09-30)
[17]`https://github.com/reponame` (Accessed on: 2018-09-30)

1.2.1 Docker

Docker describes by saying it "... powers millions of applications worldwide, providing a standardized packaging format for diverse applications" [17]. It was first announced on March 21, 2013 at PyCon by Solomon Hykes and it has captured more and more market share since. [18] Rated with over 51.k stars, Docker is by far the most popular Linux container runtime. The second most popular interface in contrast, LXC has only 2.3k stars.

Listing 1.1 : Docker Instructions

```
FROM − sets the base for the image
RUN − executes a command
ENV − sets an environmental variable
LABEL − sets a label information for the image
EXPOSE − exposes ports form the container
ADD/COPY − adds/copies a file to the container
CMD − run−time instruction for the program to execute
SHELL − overwrites the default shell
ENTRYPOINT − run−time instructions for the binary to run
ONBUILD − build−time instructions
ARG − build−time arguments
STOPSIGNAL − system call when a container exits
HEALTHCHECK − health checks for the running container
VOLUME − exposes a volume from outside the container
WORKDIR − sets the working directory of the container
USER − UID and GID to be used for the running container
```

Many solutions have been integrated with Docker to provide more specialized services on-top. The Docker application is used to create Docker images to compress applications in a Dockerfile, which includes all the necessary dependencies to run the application. In Listing 1.1 all available instructions are briefly described. A base image can be used as a starting point for an image, many major Linux distributions(Debian, Ubuntu, Fedora) have their image hosted on Dockerhub [19] and are updated and maintained by

[18]https://www.youtube.com/watch?v=wW9CAH9nSLs (Accessed on: 2018-10-01)
[19]https://www.dockerhub.com (Accessed on:2018-10-01)

their creators. Even commercial Linux distributions like RHEL offer their container images as a base image [20], but they need subscription.

Docker's paradigm is similar to that of Java, write once run anywhere[21], but in contrast to Java, Docker containers can be used with many more Linux based application and not only applications written in Java [22].

[20]https://access.redhat.com/containers (Accessed on:2019-03-15)

[21]https://tech-insider.org/java/research/1996/0123.html (Accessed on: 2018-09-30)

[22]It is still possible to containerize Java applications though, and be independent of their runtime environment or replace them depending on the need of license/performance/feature set of the JVM.

Listing 1.2 : Sample Dockerfile

```
# basic example for, which runs a python app
FROM python:alpine3.8 # alpine image for python
MAINTAINER Christian Noesterer version: 1.0
COPY . /hostnameapp
WORKDIR /hostnameapp
RUN pip install -r requirements.txt
EXPOSE 8000
CMD python ./index.py
```

In Listing 1.2 we see a simple Dockerfile, which can be used to run a Python application, after installing the relevant requirements from the requirements.txt.

The generated image can be stored in a registry to be distributed. Many applications provide maintained images, which are then pushed to Dockerhub[23]. This is also the official registry where the Docker daemon searches and retrieves images. Images can be either declared private or public. Every image can also be given a tag which further identifies it. Common tags are latest, which are often used as an identifier for the latest version release, and the appropriate version numbers or names regarding that version. When no tag is specified, the latest tag is used by default. Images can be further identified by their Sha256 cryptography hash sum. A simple way to run a multi tiered container application, is to utilize docker-compose [24]). It uses the YAML format to define the used images/ports developments and is an easy way for testing, since multiple versions of the same application can be used.

[23]https://hub.docker.com (Accessed on:2018-11-11)
[24]https://docs.docker.com/compose/ (Accessed on: 2018-01-11)

1.2.2 Kubernetes

Kubernetes (from the Greek word kubernáō "to steer" [25]), also called k8s (8 characters), was first released on September 9, 2014 and most of the code is written in GOlang. It is an open source container orchestration system, which is currently being maintained by the Cloud Native Computing Foundation. The origin dates back to Google Borg [28]. Many core developers of Kubernetes where working with Google Borg [26] and wanted to bring this new concept to the open source community.

Though it is a complex system, every component has its purpose and is needed. Initially it was hard to setup, but the ecosystem has evolved since then and more and more resources are available. In the following pages, the focus will be on the core components, which are absolutely necessary for a Kubernetes cluster. Additional components that enhance the Kubernetes even further (and are sometimes essential for stability and maintainability) are then briefly described in 1.2.2.6.

The CNCF[27] describes Kubernetes as:

> "... as an open source system for managing containerized applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications. [28]

These individual machines are called nodes (see Figure 1.2) and can host a variety of different services. Depending on the role of the node it can either be a master or a slave node.

[25] `http://www.perseus.tufts.edu/hopper/text?doc=Perseus:text:1999.04.0057:entry=kuberna/w` (Accessed on: 2018-01-11)

[26] `https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes` (Accessed on: 2018-10-11)

[27] Cloud Native Computing Foundation `https://www.cncf.io` (Accessed on:2019-02-10)

[28] `https://kubernetes.io/docs/home` (Accessed on:2018-09-30)

1.2.2.1 Nodes

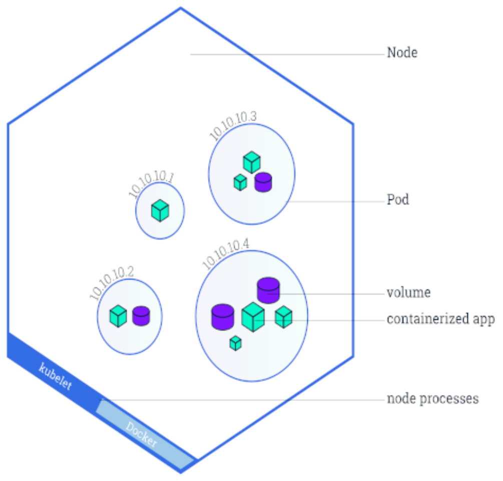


Figure 1.2: Kubernetes node [3]

Master nodes coordinate the state of the cluster. They include services such as:

- Kube-apiserver
  - exposes the Kubernetes API [29]
- Controller-managers [30]
  - this includes the replication/endpoint/namespace controller.
  - they update the current state to comply with the desired state.
- Kube-scheduler
  - check nodes which meet criteria (healthy,..)

[29] https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.12/ (Accessed on: 2018-10-30)
[30] https://kubernetes.io/docs/concepts/overview/components/ (Accessed on: 2018-10-30)

Worker nodes run the following services:

- Kubelet
  - receives requests to run a container
- Etcd
  - a highly available distributed key value store to store all the configuration for the cluster

Additionally some services should run on all nodes. These include:

- Kube-proxy
  - a network proxy, for networking services
- a OCI[31] compliant container runtime
  - The most popular is Docker but rkt, runc and OCI compliant runtimes can be used too

1.2.2.2 Pods

Pods is an abstraction for a group of applications running inside a container. [32] As seen in Figure 1.2, pods run inside a node.

They are ephemeral and, if needed a persistent storage volume can be attached to them in case, for example, the pod dies unexpectedly, is being updated or is rescheduled somewhere else inside the cluster.

[31]OCI Image Format - `https://github.com/opencontainers/image-spec` (Accessed on:2018-10-01)

[32]`https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/` (Accessed on:2019-07-10)

1.2.2.3 Objects

Kubernetes objects are the representation of the objects inside a Kubernetes cluster. Available objects include pods, services, deployments, daemonsets, namespaces and ingresses. They will also be defined in the following chapter regarding the Kubernetes API (1.2.2.4).

These are declarative and written in either JSON or in the YAML[33] format.

Listing 1.3 : Sample Service in YAML

```
kind: Service
apiVersion: v1
metadata:
  name: sample-service
spec:
  selector:
    app: sample-service
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

The following items can be declared:

- object specifications,metadata (can be used to query and select and abstract further – for example via a service)
- abstractions for objects running on the system
- resource limitations
- policies (restart policies, upgrades, fault-tolerance)
- object specific definitions [34]

[33]YAML Ain't Markup Language - It is a human-readable format, often used for configuration. `http://yaml.org/` (Accessed on: 2018-10-10)

[34]`https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/` (Accessed on: 2018-10-30)

1.2.2.4 API

The Kubernetes API [2] provides full access in managing the cluster. It can be accessed via kubectl, using any HTTP client or utilizing a frontend UI in the Kubernetes-dashboard. It is a full REST endpoint to manipulate the different types of objects in the cluster.

Supported authentication modes are client certificates, bearer tokens, an authentication proxy, HTTP basic auth (Bearer Token) or simple password authentication. Additional authentication modes can be used via a additional proxy such as LDAP, SAML, Kerberos or an alternate x509 schemes.

Services can also get access to the cluster via the api-server, first issuing a CertificateSigningRequest, where service, namespace, groups and usages[35] [36] are defined.

Listing 1.4 : Rbac authorization object with list access to the namespace object

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: datasense-access
rules:
- apiGroups: [""]
  resources: ["namespaces"]
  verbs: ["list"]
```

All authenticated requests are authorized based on either a Role Based Access Control[37] or Attribute Based Access Control [38].

[35]https://tools.ietf.org/html/rfc5280#section-4.2.1.3 (Accessed on: 2018-10-30)

[36]https://tools.ietf.org/html/rfc5280#section-4.2.1.12 (Accessed on: 2018-10-30)

[37]https://kubernetes.io/docs/reference/access-authn-authz/rbac/ (Accessed on: 2018-10-30)

[38]https://kubernetes.io/docs/reference/access-authn-authz/abac/ (Accessed on: 2018-10-30)

Requests from kubelets can be further restricted when utilizing the NodeRestriction plugin [39]. Read/write/auth related operations can also be further restricted [40].

In the resources section the resources are defined, while relevant access methods for example "get" and "list" can be defined in the verbs field. An example of this is shown in Listing 1.4.

There are APIs for:

- Workloads (Container, Cronjob, Daemonset, Deployment, Job, Pod, Replicaset, Replicationcontroller and Statefulsets)
- Discovery and Load Balancing (Endpoints, Ingress, Service)
- Config and Storage (Configmap, Secret, PersistentVolumeClaim, Storageclass, Volume and Volumeattachment)
- Discovery Events, ClusterOperations (Node, Roles, Rolebinding, Binding, ApiService) [41]

These can either be write operations (create, patch, replace and delete), read operations (read, list, watch), status operations (patch, read, replace) or misc operations (read, replace, patch) for each of these APIs. A comprehensive list of all the endpoints can be found at the official documentation [41].

[39]`https://kubernetes.io/docs/reference/access-authn-authz/node/` (Accessed on: 2018-10-30)

[40]`https://kubernetes.io/docs/reference/access-authn-authz/authorization/` (Accessed on: 2018-10-30)

[41]`https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.15` (Accessed on:2019-06-20)

1.2.2.5 Data Persistence

For data to persist, it is important to store it somewhere safe. Several things have to be taken into consideration, like how many copies of the data are kept, if the data has to be highly available, and which storage backend to use. In Kubernetes, as mentioned before data is only ephemeral but persistence can be reached utilizing a PersistentVolume. This is important since docker containers, which are used as pods can be relocated anytime from one node to another. This could be either in the case of not enough resources or a node failure.

A pod can claim a volume by requesting a PersistentVolumeClaim.

There are several plugins that support different types of volume providers. Some of them are Flocker, NFS, iSCSI,... [42].

As of Kubernetes Version Version 1.9, a Container Storage Interface was defined [43], which makes integration for volume providers easier as well as providing a full lifecycles implementation for these volumes (creation, deletion,...) [44].

1.2.2.6 Applications

Optional but very useful services which should run on any Kubernetes cluster are:

- fluentd [45]
  - a logging agent, which aggregates log files from services running inside the cluster.

[42]AWSElasticBlockStore, AzureDisk, AzureFile, CephFS, Cinder (OpenStack block storage), GCEPersistentDisk, Glusterfs, NFS, Quobyte Volumes, RBD (Ceph Block Device), StorageOS, VsphereVolume, iSCSI [4]

[43]`https://kubernetes.io/blog/2018/01/introducing-container-storage-interface/` (Accessed on: 2018-10-30)

[44]`https://github.com/container-storage-interface/spec/blob/master/spec.md` (Accessed on: 2018-10-30)

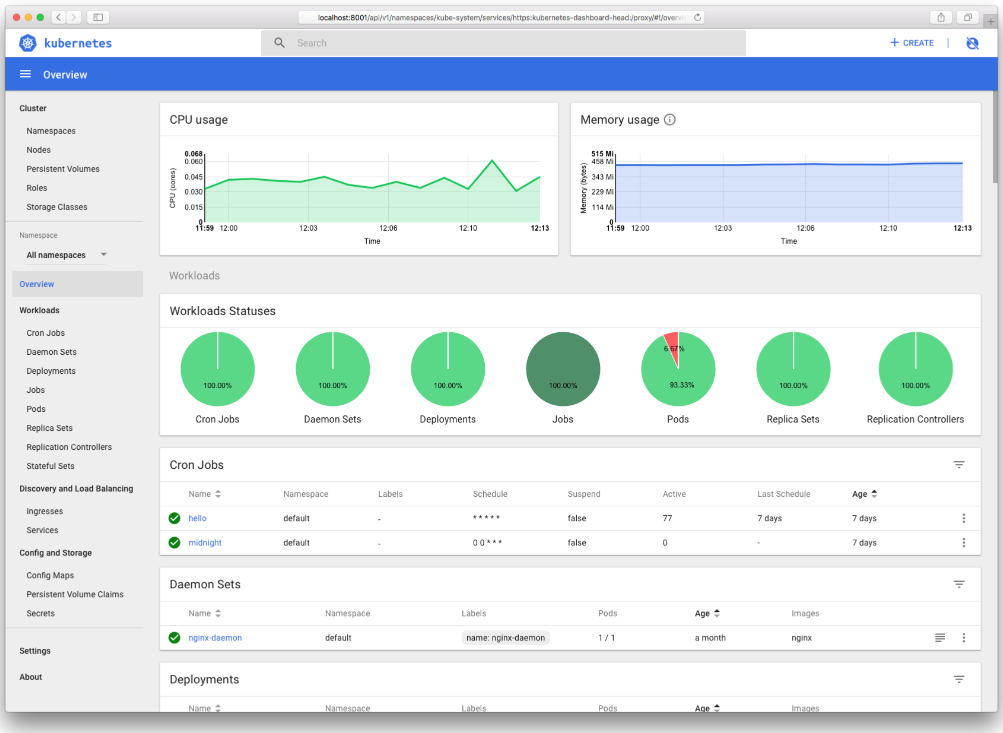
[45]`https://www.fluentd.org` (Accessed on: 2018-10-30)

  - It is often used in combination with Elasticsearch [46]

- nginx-ingress [47]
  - the ingress controller based on NGINX, which makes it easy to deploy a load balancer and manage ingress resources

[46]Elasticsearch is a search engine based on Lucene - https://www.elastic.co/products/elasticsearch (Accessed on: 2018-10-30)

[47]https://github.com/kubernetes/ingress-nginx (Accessed on: 2018-10-30)



Cron Jobs

	Name	Namespace	Labels	Schedule	Suspend	Active	Last Schedule	Age
✓	hello	default	-	* * * * *	false	77	7 days	7 days
✓	midnight	default	-	0 0 * * *	false	0	-	7 days

Daemon Sets

	Name	Namespace	Labels	Pods	Age	Images
✓	nginx-daemon	default	name: nginx-daemon	1 / 1	a month	nginx

Deployments

Name	Namespace	Labels	Pods	Age	Images

Figure 1.3: Kubernetes dashboard [3]

- Kubernetes-dashboard
  - is a simple web UI to control and manage a Kubernetes cluster.
  - it shows the workload (CPU/Memory usage) for individual nodes, as well as specific services running inside the cluster.
  - a screenshot of the latest version as of July 11, 2019 (v2.0.0-beta1) can be seen in Figure 1.3
- Registry [48]
  - provides mechanism to store and retrieve container images.
  - in some cloudproviders, like AWS,GKE,... this is often integrated into their proprietary Kubernetes cloud system.

[48] https://hub.docker.com/_/registry/ (Accessed on: 2018-10-30)

  - two additional registry services are Quay[49] and Dockerhub[50].
  - they provide supplementary options like: container vulnerability scanning, auto-deployment and more extensive authentication/authorization options.

- Kube-dns [51]
  - provides DNS to be used inside the cluster for name resolution of services.
- Heapster
  - is a monitoring solution best used in combination with Grafana, which visualizes the metrics and provides analytics & dashboards.

As Kubernetes has even been called the New Linux [19], a package manager has to exist to maintain packaged software. This includes the full lifecycle of a software in their production environment (install, update, delete). Besides that a central repository should exist where all the packages are safely stored, preferably in conjunction with a cryptographic signature to avoid tampering. The current and only package manager, which satisfies these constraints is called Helm [52]. It has two components the client (helm) and the server component (Tiller).

Packages are called charts. There is an official repository, where as of July 14, 2019, 278 packages are available. [53] Furthermore there is an incubator[54] repository where as of July 14, 2019, additionally 60 charts are available.

1.2.2.7 Installation

Installation was cumbersome at the beginning when Kubernetes was still in an early alpha version (pre version 1.4). Since the introduction of Kubeadm

[49]`https://quay.io/` (Accessed on: 2018-10-30)
[50]`https://dockerhub.com` (Accessed on: 2018-10-30)
[51]`https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/` (Accessed on: 2018-10-30)
[52]`https://helm.sh` (Accessed on: 2018-10-30)
[53]`https://github.com/helm/charts/tree/master/stable` (Accessed on: 2018-10-30)
[54]`https://github.com/helm/charts/tree/master/incubator` (Accessed on: 2018-10-30)

in version 1.4.0, Kubernetes provided an easier way in setting up clusters. It was designed to provide a minimum viable platform to build upon.

Many optional applications described in 1.2.2.6 are not included but should gradually build upon.

There are many more options to deploy Kubernetes ready cluster, like with Kubespray a tool, which is used to deploy a production ready cluster to major cloud platforms like GCE, AWS [55] [56].

Other automated deployment tools for Kubernetes include: Kubespray[56], Kops[57], hypercube [58], kube-aws[59] and kubicorn[60].

Listing 1.5 : Sample Terraform provider configuration

```
provider "aws" {
  access_key = "test"
  secret_key = "test"
  region     = "us-west-1"
}
```

One solution which is exceptionally useful in this context is Terraform [61].

As of 14.July 2019 over 106 cloud providers can be used to deploy an Terraform configuration. A simple configuration can then be used to define the authentication steps needed for that particular cloud provider, as seen in Listing 1.5. It is a infrastructure as code utility to roll out new server configurations. It can be used to create, update, destroy or modify the cluster and can even provide tooling for generating a graphical representation of the cluster.

Some automated deployment tools (Kops, Kubespray) can even generate Terraform output.

[55] AWS, GCE, Azure, OpenStack, vSphere, Oracle Cloud Infrastructure or baremetal
[56] `https://github.com/kubernetes-sigs/kubespray` (Accessed on: 2018-10-10)
[57] `https://github.com/kubernetes/kops` (Accessed on: 2018-10-10)
[58] `https://github.com/kubernetes/kubernetes/tree/master/cluster/images/hyperkube` (Accessed on: 2019-02-20)
[59] `https://github.com/kubernetes-incubator/kube-aws` (Accessed on: 2018-11-20)
[60] `https://github.com/kubicorn/kubicorn` (Accessed on: 2018-11-20)
[61] `https://terraform.io` (Accessed on: 2018-10-30)

What is further in need is some kind of configuration management software like (Ansible [62], Puppet[63] or Chef[64]) to deal with the underlying OS and keep the nodes updated. This can also depend on which Linux distribution Kubernetes is deployed on.

[62]https://www.ansible.com/ (Accessed on: 2018-11-05)
[63]https://puppet.com/ (Accessed on: 2018-11-05)
[64]http://www.chef.io/ (Accessed on: 2018-11-05)

1.2.2.8 Network

1.2.2.9 Security

CVE-2018-1002105 was one of the first major security flaws in Kubernetes. It was possible to gain access to the cluster through the API server and perform a privilege escalation. The cloud native foundation published a security best practice on January 14, 2019 to enhance security to defend against unauthorized data access.

These are the following advices:

- Upgrade to the Latest Version
- Enable Role-Based Access Control (RBAC)
- Use Namespaces to Establish Security Boundaries
- Separate Sensitive Workloads
- Secure Cloud Metadata Access
- Create and Define Cluster Network Policies
- Run a Cluster-wide Pod Security Policy
- Harden Node Security
- Turn on Audit Logging [10]

Furthermore the underlying container platform should also be secured properly. In a recent hack on January 14, 2019 [65], security researchers gained access to the underlying host kernel of a popular platform named play-with-docker.com, this was due to improper configuration the privileged container. In [1] Brendan Michael Abbott described in chapter 4.1 more methods how to properly secure Container Images.

[65]https://threatpost.com/hack-allows-escape-of-play-with-docker-containers/140831 (Accessed on: 2019-01-20)

1.3 Databases

When many sensors are utilized and they sent their values in a very short time frame it is necessary to be able to aggregate this data or infer new knowledge through artificial intelligence. Additionally, it has to be stored somewhere, if the processing is not done in real-time. For intelligent algorithms more data means more data process, which in turn can than be used to improve the algorithms efficiency and predictive capabilities.

This poses many challenges such as clustering to gain high availability, storage of data, searchability, processing capabilities and how to deal with unknown data structures.

For structured data storage, first there were structured databases such as SQL, which was first introduced by IBM in 1974 [9] provided an easy way for data manipulation but still needed a database schema upfront to be able to work with data. Problems arise if data is not known or unstructured, which can be in the case of sensor data, when, for example multiple vendors are used and therefore data cannot be manipulated.

To the rescue NoSQL solutions re-gained popularity in the last few years due to the fact that it can deal with large amount of data and is also designed to be schema-less, often at the cost of disk space (as data is often stored in JSON). Furthermore NoSQL databases often provide features like sharding, horizontal scalability, like with MongoDB for example.

Contrary to what one would believe NoSQL is even older than SQL, as it has been around since the 1960s [22], but the difference is that disk space was more precious back then. As consistency of NoSQL databases and partition tollernace is not an issue with modern NOSQL solutions availability of data at all the time is. These principle can be explained according to the Brewers CAP Theorem, which holds true for all distributed systems currently known. It states that it is not possible for a distributed system that it simultaneously has two out of the three desired quantities, namely consistency, availability and partition tolerance. This holds also true for any database and depending on the need a trade off has to be made.

1.3.1 NoSQL

NoSQL can have different features as there are different use cases and business needs. On November 9, 2009, Steven Yen gave a presentation and declared nine different types of taxonomy of databases [11].

These are the following:

- key-value-cache
- key-value-store
- eventually-consistent key-value-store
- ordered-key-value-store
- data-structures-server
- tuple-store
- object-databases
- document store
- wide column store

[11]

For sensor data the most popular ones are either wide column stores/key-value stores or document stores, since they have the most features relevant to storing and sensor data. More IoT Specific NoSQL solutions often incorporate features such as data compression/data cleanup features, connectors, resampling, aggregators and much more.

Timestamped data is one of the things any of these systems has to deal with in regards to sensor data. More optimized solution exist in this context when dealing with time series data called time series databases.

1.3.2 Time Series Databases

Time series databases (TSDB), are a wonderful way when dealing with timestamped data, since they integrate many features that make it easier to deal with this kind of data. Often data can be further aggregated with avg, sum or rate functions.

Some offer compression for efficient storage and different kind of periodicity for the granularity of data. Usually they offer additional visualization to arrange the aggregated data. Downsampling of large amounts of data can furthermore increase data storage efficiency and make it easier so that the data can be used in calculations. In 3.8.3 different kinds of time series and NoSQL databases were evaluated, which can fit in the context of this thesis. Several different Query languages can also be used to consolidate and aggregate data. While this is often somehow similar to SQL it still is very unique to each time series database.

1.4 Microservices

For orchestration to work, an application has to be prepared to meet certain components to be integrated. One document which describes some profound steps in meeting these criteria is "The twelve factor App" by Adam Wiggins.

It is based on user experiences and observations on a wide variety of software-as-a-service apps in the wild.

Adam Wiggins, who co-founded Heroku [66], developed a modern methodology which is independent of the programming language and describes some core principles needed for any application to survive in the cloud area. Many of these factors are essential for Kubernetes, as otherwise scaling up and down can be impossible. Its twelve factors are described in the Twelve-Factor App, which as of December 21, 2018, has been translated into 13 languages and is also found online [67].

[66]`https://heroku.com` (Accessed on: 2018-10-30)
[67]`https://12factor.net` (Accessed on: 2018-12-10)

1. Codebase - one codebase with revision control and many deploys (production, staging, etc.). Local and production should not differ.
2. Dependencies should be explicitly declared and dependencies isolated.
3. Config - should be stored in the environment - a strict separation of code vs config is enforced.
4. Backing Services - backing services are treated as attached resources.
5. Build, release, run - separated build and run stages.
6. Processes - the process is stateless and persistent data is stored in a backend.
7. Port binding - services are exported via portbinding.
8. Concurrency - scaling out via the process model.
9. Disposability - robustness and fast startup and handled graceful shutdown.
10. Development /production parity - keep deployment and production as similar as possible.
11. Logs - logs should be treated as streams, the collection/storage should not be the concern of the application.
12. Admin processes - admin/management tasks are one one-off processes.

[29]

The same local and production deployment environment can be achieved in a Kubernetes system, utilizing a declarative installation/run routine using Dockerfiles.

Dependencies can be declared in the Dockerfile using the specific package manager for a particular programming language (npm for Node.js, pypi for Python, CPAN for Perl, ...).

System-wide declaration would be possible if many dependencies are needed (Xorg, QT,..) but is strongly discouraged. As applications lifecycle are often integrated into a continuous integration system many different versions of the same app with different configuration are needed. Persistent data can be stored in a Kubernetes application using Persistent Volumes [68].

[68] `https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.15/#persistentvolume-v1-core` (Accessed on: 2019-03-15)

Ports are exposed in a Dockerfile using the EXPOSE command. Afterwards this can be used in a Kubernetes service to either expose it internally or outside of the cluster using a load balancer.

In a Kubernetes application this can be achieved using ConfigMaps [69] . Concurrency in Kubernetes can be handled via a Deployment, ReplicaSet, StatefulSets or Daemonset as processes are stateless and data is shared using a backend.

Kubernetes deployments can be started quickly as their are not occupying any additional resources and are not full blown VMs.

Many lifcycle methods can trigger events in the case of a container creation or termination. [70]. Graceful shutdown can be handled using the PreStop hook, this command is called immediately after the container is terminated. The command is blocking and should let the container cleanup any leftovers before a graceful shutdown.

Sometimes a simple sleep command can be used, but this is not encouraged as it is non-deterministic and often leads to more bugs. If the application was started using a process manager like for example NGINX or Apache their respective graceful shutdown periods should be used.

Containerized applications should use STDout/STDErr to communicate their state as logs can be collected and aggregated in a Kubernetes system. A logging driver, which is a set of applications can be used to enhance logging in a Kubernetes cluster. In a live environment The EFK stack (Fluentd[71], Elasticsearch and Kibana [72]) can be used to collect, search and visualize the collected log data.

Admin/Management tasks can be run using Jobs [73], which can be run in

[69]https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.15/#configmap-v1-core (Accessed on: 2019-03-15)

[70]https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/ (Accessed on: 2019-03-15)

[71]https://www.fluentd.org/ (Accessed on: 2019-03-15)

[72]https://www.elastic.co/products/elastic-stack (Accessed on: 2019-03-15)

[73]https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.15/#job-v1-batch (Accessed on: 2019-03-15)

parallel if needed. A CronJob [74] can be used to reschedule recurring jobs.

In Kubernetes Deployment livenessProbe, readinessProbe and a lifecycle method can be used to recreate services depending on their availability.

1.5 Web Frameworks

Since the first website was posted on August 6, 1991 by Tim Berners-Lee [75] the web changed in many different areas.

As of June 14, 2019, 201 different web frameworks are listed by [76], this is in stark contrast to the web 1.0 where there was not even JavaScript around. Since March 28, 2013, TechEmpower run 18 benchmark rounds in regards to performance on these web frameworks.

Tests include:

1. JSON Serialization
2. Single Query
3. Multiple Queries
4. Fortunes
5. Data updates
6. Plaintext

What stands out is the fact that all of the top 10 frameworks are processing requests in a reactive manner, which means they have a way of processing requests in a non-blocking way. This is in essence the functional asynchronous pattern.

[74] `https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.15/#cronjob-v1beta1-batch` (Accessed on: 2019-03-15)

[75] `https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt` (Accessed on: 2019-03-15)

[76] `https://www.techempower.com` (Accessed on: 2019-07-23)

2 State of the Art

To find related research, a search was carried out via the meta paper search engine Google Scholar. The following keywords were used "Kubernetes Iot", "container IoT", "internet of things container", "iot orchestration", "paas iot container", "IoT orchestration" and "blockchain IoT".

Additionally a manual reverse reference searches from the initial research was performed.

From the over 59560 results, the most recent from the year 2018 and 2019 (76960 results) were considered, to further restricting the results only the first 50 search results per keyword were considered. The relevant/related scientific papers were structured into Weakly related and papers with a strong correlation to this thesis. Furthermore research was conducted to see, what commercial solutions/open source are out there and how they are related to Datawiz.

Weakly Related - Benchmarking:

Weakly related papers were [27], which is a study of Linux Containers and their ability to quickly offer scalability for web services focuses on scaling and benchmarking of web services. MLN(openstack) was benchmarked against Kubernetes utilizing HTTP (Apache 2.4.7) and SQL (MySQL 5.5) traffic. Benchmarking was done using a custom Python script. It showed that Kubernetes is 2x faster than mln(openstack). It has however no strong IoT context.

Weakly Related - Edge computing:

In [20] Asad Javed created a Container-based IoT Sensor Node based on Raspberry Pi and the Kubernetes Cluster Framework,which focuses on a container based IoT platform based on the Raspberry Pi. It used five

Keyword	**Results total**	**Results 2018**	**Results 2019**
Kubernetes IoT	1260	802	343
container IoT	21200	8210	3280
"Internet of things" container	26200	11500	4400
Paas IoT container	3400	1510	568
IoT orchestration	10600	4780	1930
blockchain IoT	14300	9500	4560

Table 2.1: Different Orchestration provider number of Google scholar results.

Raspberry Pi 2 with a network switch, collecting temperature and camera data and aggregating the data using Apache Kafka framework. A test regarding fault-tolerance and high availability was performed in the end. While Raspberry Pi is still one of the cheapest microcomputer available, it still is way more expensive than cheaper microcontroller like ESP8266.

Another implementation using the Edge Computing Architecture was performed by Kristiani Endah et al. [21] . They evaluated three layers, Cloud side, Edge side, and Device side and created a Edge Gateway to increase performance and reduce bandwidth.

Strongly Related - Iot platforms:

GENESIS is strongly related to the work as it provides a full platform [13] for orchestration and deployment of IoT systems. It uses a code generator and Node-RED containers to facilitate data flow management. As a storage engine it uses CouchDB for storage but it is not cloud based as they deploy their components microprocessors like the Raspberry Pi. FRED [6] is a platform, hosting multiple Node-RED instances with a provided MQTT service and storage utilizing InfluxDB. In the paper in [6] however, they do not provide more detailed information regarding their orchestrator, as it was only mentioned as an open research question for them. In his Master's Thesis, Mikko Yliniemi developed a Node.js based system for monitoring air quality called Co2.io [30]. Docker Swarm was used to scale out the system and in the end an evaluation of gathered air quality data was performed.

Platform	IoT Applications	device generator	open source	IOT specific	Intelligence
Datawiz	6	yes	yes	yes	via Ludwig/Nupic
IBM Bluemix	11	no	no	no	via Watson
Thingspeek	4	no	no	yes	no
Fred	3	no	no	yes	no
Tag.IO	6	yes	no	yes	yes
Ubeac	4	yes	no	yes	yes
Devicehive	7	yes	yes	yes	audio/video analysis
Mainflux	9	yes	yes	yes	-

Table 2.2: Different commercial IoT platforms features VS Datawiz

Strong related - Open Source/Commercial IoT platforms

The following commercial projects/open source projects were evaluated, based on their feature set seen in Table 2.2 While commercial projects like IBM Bluemix offers additional features that are not IoT specific, it is proprietary and therefore not possible to use it on an existing cluster/server infrastructure. Mainflux offers advanced features like multiple protocols, and ACL and deployment via Docker/Kubernetes. It is also part of the Linux Foundation and actively developed.

3 Method

3.1 Preliminary work

First I started to implement the web application using the Meteor framework, this offers some nice features for real-time showcasing but does not scale well. After upgrading the Meteor library[1] it did not compile anymore due to some screwed up dependencies and issues with the evolving react framework emerged.

React[2] needed a data store to make it extensible and since this took many hours of working through but in the end this simply defied the case of being offering a framework which is extensible. The persistence layer of NoSQL on a storage level was moreover not ideal, as it posses risks to data consistency and database design.

When the thesis began, CoreOS was completely open source and one of the distributions which made it easy to use Kubernetes and although the setup was quite complex, with not many ways to automate it got a full one node cluster working. However since this was still partially a vendor lock-in, unfortunately CoreOS was then acquired by RedHat Inc. on January 30, 2018 [3] and documentation was not updated anymore.

This created additional problems since Kubernetes did not have the most recent version, therefore many things did not work as expected. One of them was the package manager helm, but also ingresses evolved in many ways, which broke things.

[1]`https://www.meteor.com/` (Accessed on: 2019-02-20)

[2]`https://reactjs.org/` (Accessed on: 2019-02-20)

[3]`https://redhat.com/en/blog/faq-red-hat-acquire-coreos` (Accessed on: 2018-03-30)

Additionally since CoreOS was only ”Free for use” for up to 10 nodes this was not acceptable as a requirement, since it is technically not open source anymore and scaling beyond that limit simply does not work with the open source version.

It was than decided to solve this vendor lock-in by utilizing Terraform, so that it can be independently deployed and provide a more vendor neutral approach.

3.2 Overview

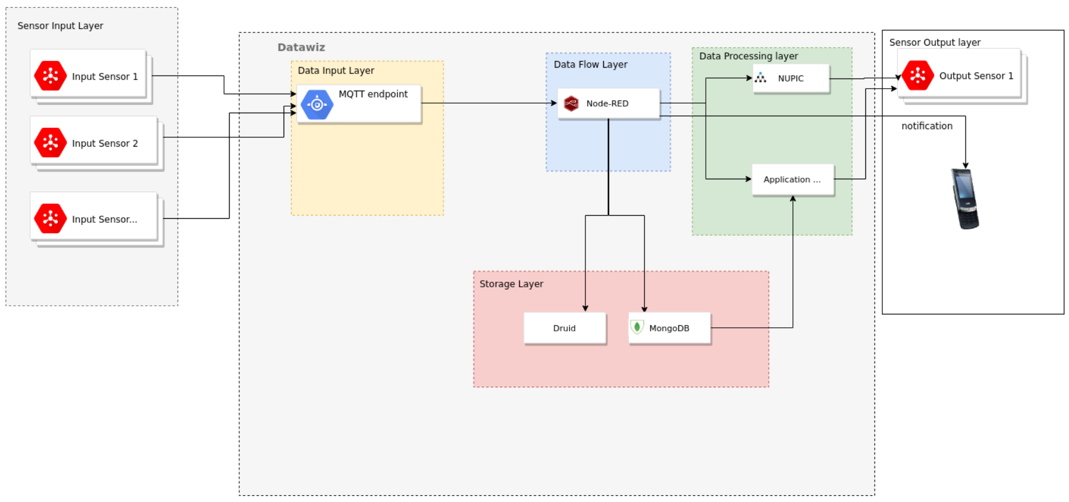


Figure 3.1: Datawiz - High level overview

Datawiz [combination of Data and Wizzard] is an approach to provide a IoT based sensor data framework based on open source technologies. In the core it is based on three different parts:

- Kubernetes with a chosen storage driver and several components to enhance system maintainability
- The UI - a web application based on the Act framework to provide sensor and application management
- Various selected evaluated user applications to help provide a full data flow from the sensor input layer over to the data flow processing layer, a data persistence layer and a data processing layer.

For each of the layers evaluated applications and reference hardware is provided to fully show the integration into Datawiz.

Datawiz is a framework especially for sensor data to provide the following:

- a code generator for various sensors to be integrated into the data Flow
- isolation for different users
- integrate open source applications for data flow integration and processing
- provide a database backend to save aggregate data
- provide open source computing applications
- provide performance tested data transport layer integration (HTTP, MQTT)

All basic parts will be scalable and failsafe, so that in an event of a server failure data will not get lost. This is guaranteed with a storage layer which provides high availability and failover.

Furthermore the included parts are: a load balancer, the WebUI, a database and a MQTT server with exceptions of some user running applications (the data is persistent though). It is easily scalable, as it will be built on top of Kubernetes and use its features to harness this power.

In production environments deployment and operations are often separated. This can get even more complicated if the production and development environment is managed by different people.

To make Datawiz easily deployable the existing Terraform script from Stefan Prodan was adopted. [4]

[4]`https://github.com/stefanprodan/k8s-scw-baremetal` (Accessed on: 2019-02-20)

3.3 Setup

3.3.1 Local development environment

For local development k3s[5] was used, since it is rather lightweight and facilitates testing/deployment and development on the local machine.

A Three Node Kubernetes cluster can be created with commands seen in 3.1.

Listing 3.1 : K3s creation of development cluster

```
kind create cluster --name datawiz --config config
```

This setup still had some issues, like the multi server setup and for the docker in docker approach the storage layer could not be tested well. This ranged from permission issues to kernel module support on the host system and many more. Furthermore whenever the host system was connected to a new network, the clusters network configuration was messed up and at least the multi-server approach did not work anymore.

That meant at least for evaluation purpose that more hardware or a cloud based setup was needed to evaluate various scenarios and test the performance of the different components.

3.3.2 Remote Setup

The remote setup is done via Terraform on the Scaleway cloud[6] . The Terraform script from Stefan Prodans [4] served as a basis.

Four bare metal instances were create with one master and three worker nodes. The nodes have the following hardware specifications:

- 4x nodes (One master node and three worker nodes) scaled up to nine worker nodes

[5]`https://k3s.io/` (Accessed on: 2019-04-30)
[6]`https://cloud.scaleway.com` (Accessed on: 2019-04-30)

- CPU: 64bit Intel(R) Atom(TM) CPU C2550 @ 2.40GHZ
- Memory: 8GB DDR3 1600Mhz
- OS: Ubuntu 18.04.2 LTS [7]
- 50 GB SSD (for OS/ephemeral Docker image storage)
- 300MBit network connection
- 25GB additional disk for storage for storage driver

The system was then used to load test the cluster under several scenarios. This is further described in chapter 4.

After approximately five minutes the master node of the cluster was up and ready with all the kernel modules loaded and ready to install Datawiz onto it. The creation of additional node took five to six minutes and can be done simultaneously.

If a node is not responding to the Kube API Server in a time frame of 5 minutes (which is the default value), the pods on this node are migrated to another node. This threshold however can be configured to suit the needs of the cluster. For example, if a quicker failover for user based applications is needed the threshold should be set lower.

The following kernel modules are needed to be enabled for user file storage and the persistent volume claims inside the cluster.

- target_core_mod
- tcm_loop
- target_core_file
- configfs

To show detailed cluster utilization of memory/network and CPU and many more metrics Grafana in combination with Prometheus was used. Prometheus is a monitoring and alerting solution, which supports a wide range of server metrics. Additionally, it supports more features, like alerting and visualization as it is also a time series database but these features were not used for server monitoring purposes. Grafana is a popular dashboarding solution used in combination with Prometheus to show these metrics.

[7] `http://releases.ubuntu.com/18.04` (Accessed on: 2019-04-30)

There are official helm charts for both software [8] [9]. Grafana's store provides a wide range of dashboards submitted by their users. In the context of Kubernetes it is a great way to give a good overview of cluster utilization of all the various server metrics, as well as per Node utilization.

Besides it is possible to even get detailed data from a per Pod basis. Of particular interest is the dashboard from mjsjinsu [10], which captures all these metrics from Kubernetes. There is even an official Kubernetes addon from Grafana, which captures even more metrics [11].

[8] `https://github.com/helm/charts/tree/master/stable/grafana` (Accessed on: 2019-04-30)

[9] `https://github.com/helm/charts/tree/master/stable/prometheus` (Accessed on: 2019-04-30)

[10] `https://grafana.com/grafana/dashboards/6772` (Accessed on: 2019-04-30)

[11] `https://github.com/grafana/kubernetes-app` (Accessed on: 2019-04-30)

3.4 Components

Datawiz is an approach to create a open sensor data framework, which is solely built upon open source components. It is built upon various open source components that let the user user choose from a range of applications to integrate their sensors and provide a full dataflow from input to output.

This includes a reference hardware (ESP8266 (input, output)), a data flow management solution (Node-RED), data transport mechanisms and a processing application to generate implicit knowledge.

name	performance	high availability	package	released
ceph	14.2 MB/s	yes	via rook.io	2012
longhorn	16.5 MB/s	yes	yes	23.3.2018
openebs	13.9 MB/s	yes	yes	7.4.2017
local (dd)	200.2 MB/s	-	-	
remote (ssh dd)	33.4 MB/s	-	-	

Table 3.1: Various data storage solutions for Kubernetes.

3.5 Storage Layer

Different storage layer solutions exist to provide persistent storage for application run inside the cluster as well as for user based applications. This includes applications which request storage via a Persistent Volume Claim reclaim the space needs for any application data. Application data include database storage from the NoSQL and SQL databases as well as the data of user specific applications such as Node-RED or a CI application. Based upon further investigation storage solutions, which can be seen in Table 3.1 were evaluated.

First Longhorn[12] was further examined, since it showed promising performance but unfortunately for every volume there has to be a pod running and because many user applications are running on the cluster this seemed not the best solution for that use case.

Additionally after a loadtest, volumes became unhealthy and they could not be fixed automatically and needed manual interventions. Since it is one of the youngest projects evaluated, it looks promising in regards to configuration needed, but it was not usable under these conditions. Ceph, on the other hand, is quite mature in that regards and was used to scale to 16 exabyte and beyond according to Red Hat [13].

[12]`https://github.com/longhorn/longhorn` (Accessed on: 2019-04-30)
[13]`https://docs.ceph.com/docs/master` (Accessed on: 2019-04-30)

Framework	Single Query (tm)	Single Query (cl)	Multiple Queries	Update	Language	Github *
Vert.x-web	656.272	12.219	1338	653	Java	608
Actix	572.115	17.744	1287	763	Rust	2916
swoole	461.819	8.208	671	186	PHP	13576
cpoll	409.617	11.219	-	197	C++	-
Jooby	357.775	7.915	780	634	Java	1056
undertow	349.116	9.097	779	623	Java	2421
Ulib	260.154	12.133	1036	1033	C++	789
micronaut	54,850	7990	1076	407	Java	2426

See the items in S.1 S.2 S.3 S.4 S.5 S.6 in section 1.4 for a detailed explanation of the performed test cases.

Table 3.2: Top web frameworks sorted by Single Query

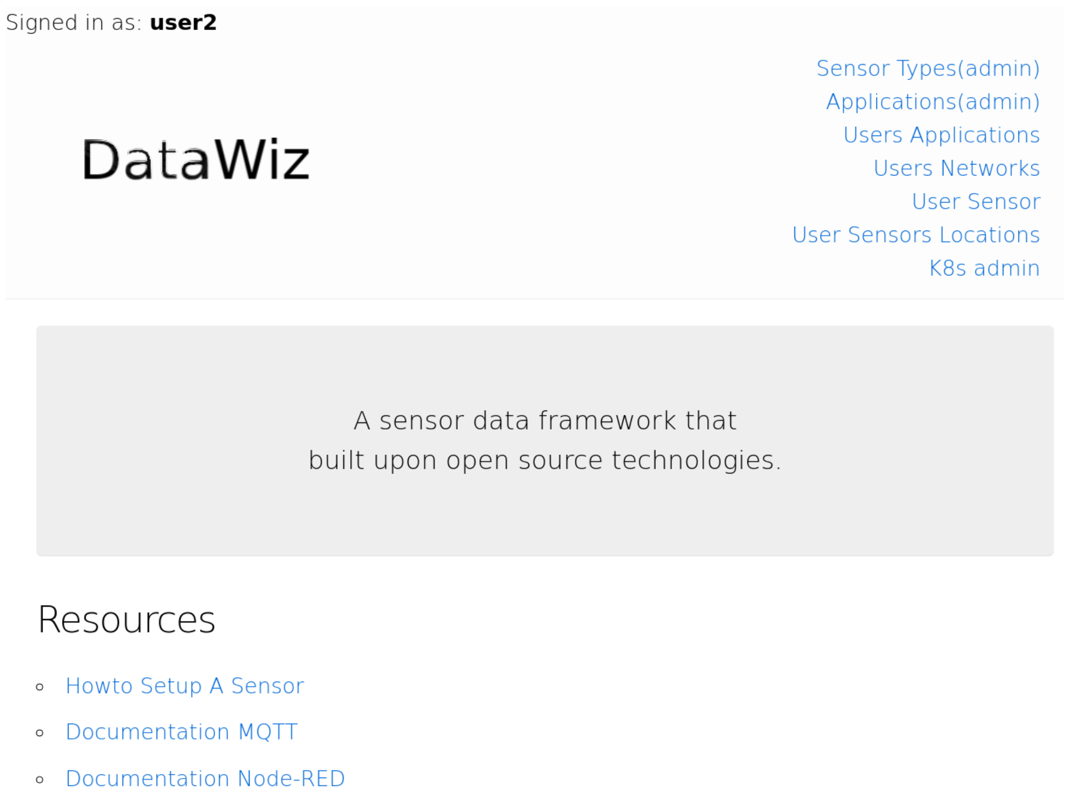
3.6 User Layer - Use Cases & Requirements

The web application is written in Java based upon Vert.x-web[14], which itself is based upon Eclipse Vert.x[15]. JSON Web Token [16]are used to provide session information to be used across multiple instances. Authentication/authorization of provided resources is further based upon the token. Vert.x uses an event loop, to handle requests in a reactive way, meaning it can handle incoming requests simultaneously without blocking each other.

The UI (main shown in Figure 3.4) is a screenshot of the main view of the application. This test was performed on one machine with the hardware specifications described in section 3.3.2. The benchmark was conducted on May 18, 2019 and it took a total time of 84 hours 18 minutes and 36s for benchmarking program by TechEmpower[17] to complete. As seen in Table 3.2 the Vert.X framework performs quite well on all different performed tests.

[14]https://vertx.io/docs/vertx-web/js/ (Accessed on: 2019-05-11)
[15]https://vertx.io/ (Accessed on: 2019-07-09)
[16]RFC7519 https://tools.ietf.org/html/rfc7519 (Accessed on: 2019-05-11)
[17]https://github.com/TechEmpower/FrameworkBenchmarks (Accessed on: 2019-07-09)

For the database access it uses JDBC [18], which provides database-independent configuration. As the database the MySQL drop-in MariaDB is used to provide fault-tolerance and high-availability.


Signed in as: **user2**

DataWiz

Sensor Types(admin)
Applications(admin)
Users Applications
Users Networks
User Sensor
User Sensors Locations
K8s admin

A sensor data framework that
built upon open source technologies.

Resources

- Howto Setup A Sensor
- Documentation MQTT
- Documentation Node-RED

Figure 3.2: Main user interface overview

The database schema consists of three parts (the user schema,the sensor schema, the application schema).

[18]JSR 221: JDBC 4.0 API Specification `https://jcp.org/en/jsr/detail?id=221` (Accessed on: 2019-04-30)

3.6.1 User schema

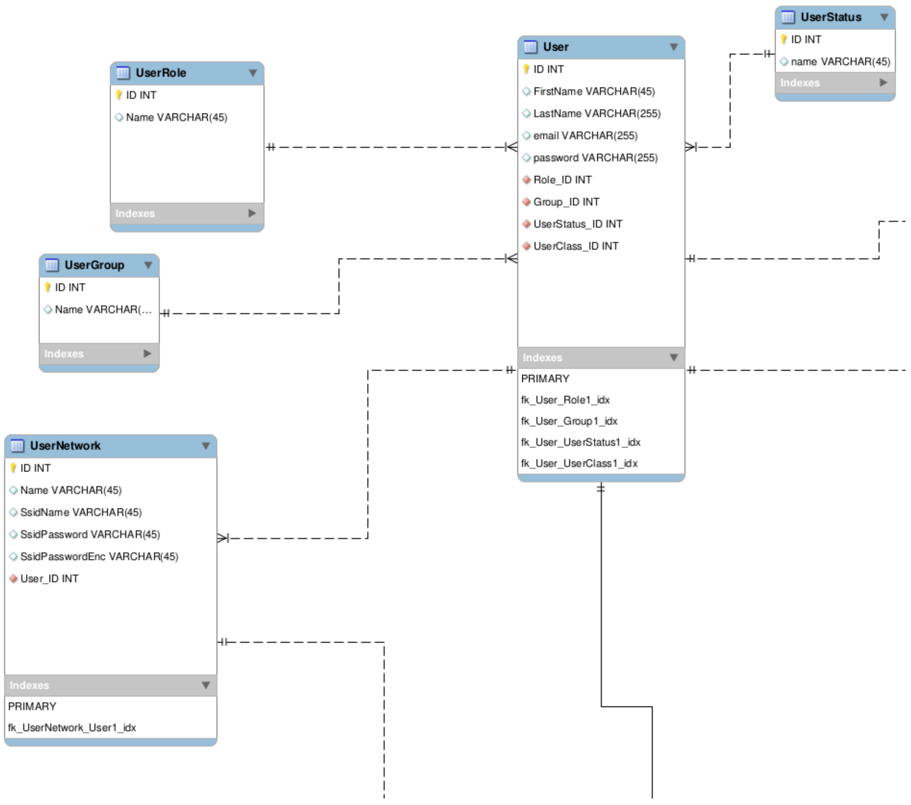


Figure 3.3: Datawiz - Database User Schema

The User database schema consists of:

- User Role - what is the role
- User class - what class of resources a user get
- User Group - to which group does they belong
- User Status - what is their status (pending, active, deleted,...)
- User Network(s) - which wireless networks did this user create
- User Application(s) - which applications did the user create

3.6.2 Application schema

The application schemas consists of all the various parts necessary for the applications a user has and their respective values.

The application database schema consists of:

- UserApplication - the user specific application data
- Application - the Application (Status, Category, Comments)
- ApplicationValues - the user entered values for an application
- ApplicationTemplateValues - the variables for an application

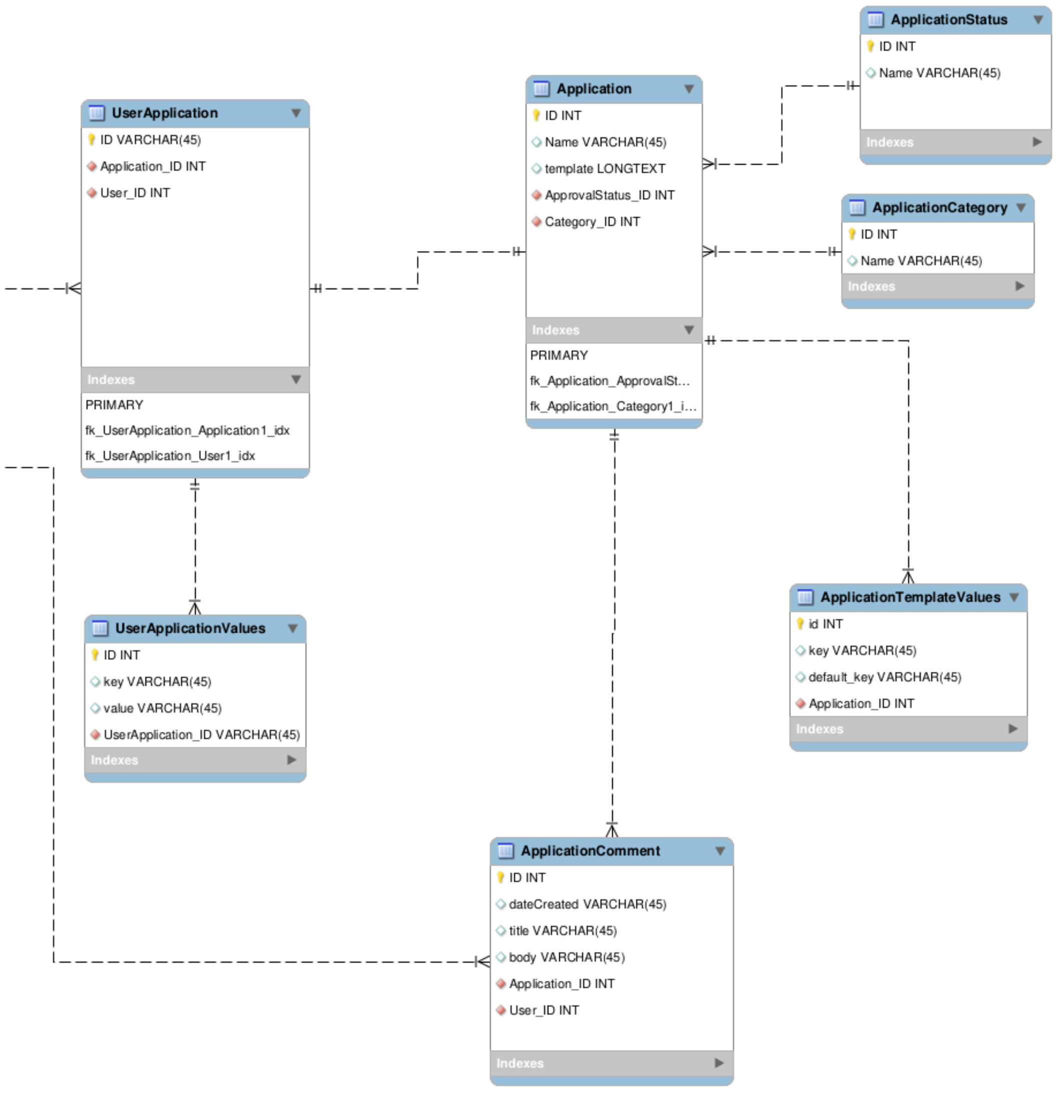


Figure 3.4: Datawiz - Application Schema

3.6.3 Sensor schema

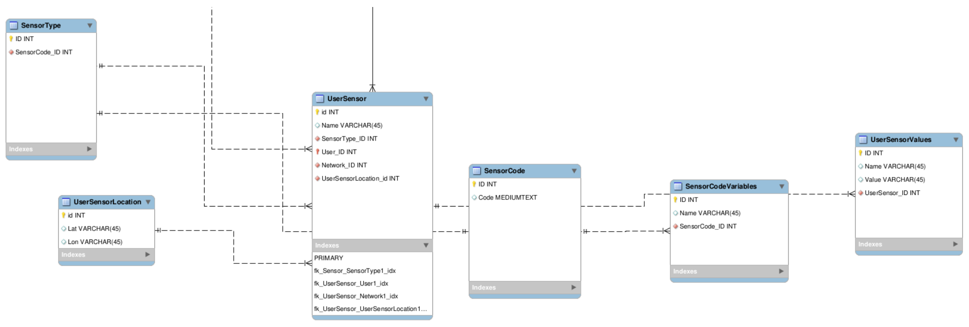


Figure 3.5: Datawiz - Sensor Schema

The sensor schemas includes of all the various parts necessary for the sensors a user can have and their respective values. The sensor database schema consists of:

- SensorType - which Sensor
- SensorTypeCode - the codetemplates for the sensor
- SensorCodeVariables - the user entered values for a sensor
- UserSensorLocation - the location data of the sensor

3.6.4 Use Cases

The following pages will describe the components of the UI which consists of the following:

- User Registration
- Group Creation/Management
- Sensor Registration
- Sensor Location
- Network configuration
- Sensor Configuration
- Application Configuration

All actions are CRUD[19] based.

[19]create, read, update and delete

3.6.5 User View

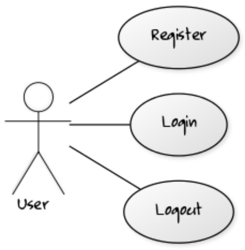


Figure 3.6: Datawiz - User

A user can perform basic actions, which consist of every login system. These are Register, login and logout (3.6). After this basic flow is completed, they can join a group. Furthermore he can start to add new sensors and applications.

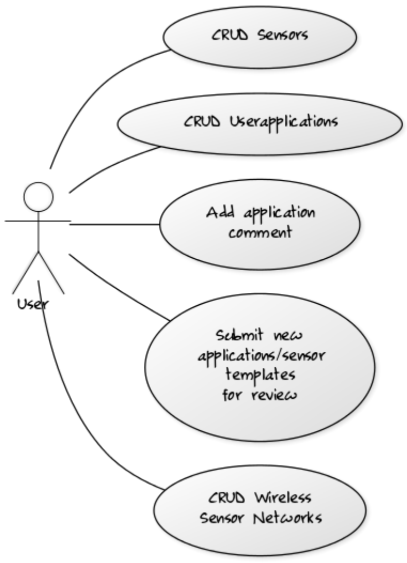
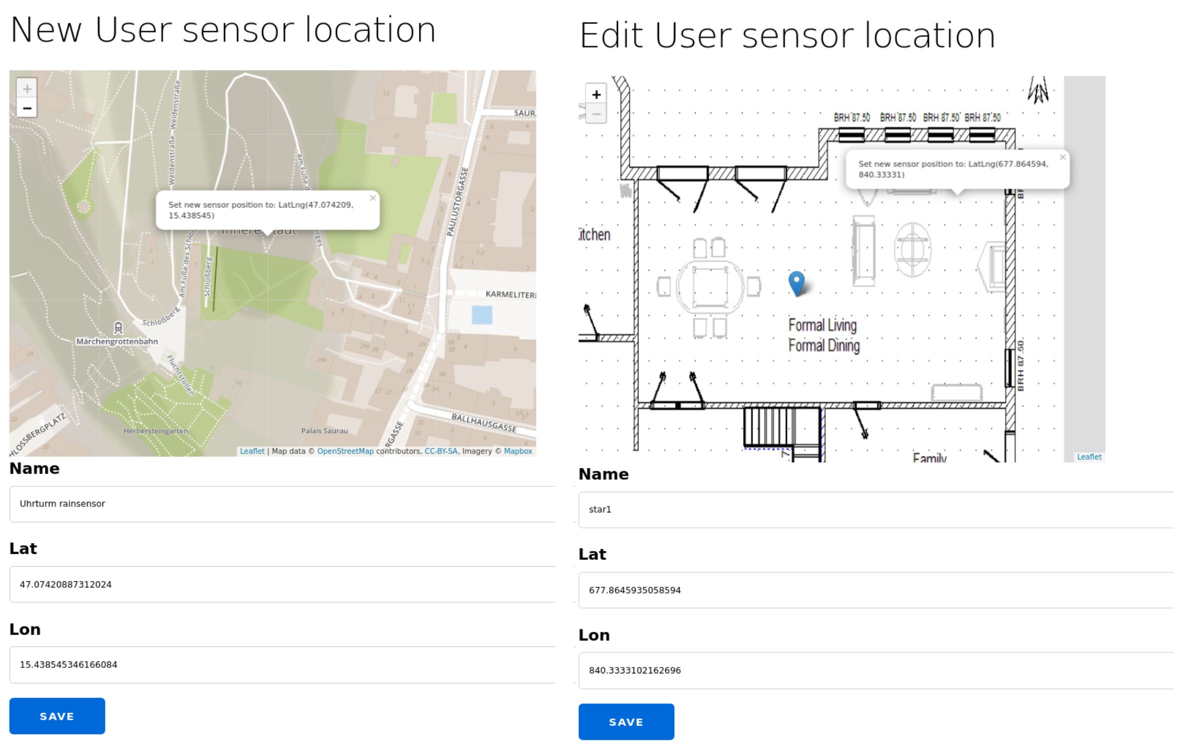

CRUD Sensors
CRUD Userapplications
Add application comment
Submit new applications/sensor templates for review
CRUD Wireless Sensor Networks
User

Figure 3.7: Datawiz - User

Users further can CRUD sensors. They can also CRUD userapplications based on existing templates and add comments to an existing application template. Furthermore after submitting a new application/sensor template, it can be reviewed by an administrator.


Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox
Formal Living
Formal Dining
Leaflet
SAVE
SAVE

New User sensor location

Set new sensor position to: LatLng(47.074209, 15.438545)

Name: Uhrturm rainsensor

Lat: 47.07420887312024

Lon: 15.438545346166084

Edit User sensor location

Set new sensor position to: LatLng(677.864594, 840.33331)

Name: star1

Lat: 677.8645935058594

Lon: 840.3333102162696

Figure 3.8: Datawiz User Sensor Location - outdoor (new view) and indoor (update view)

After adding the sensor it is possible to add location data to each of these sensors. A user can pick GPS location data of these sensors or upload a custom indoor map to pick the location.

The view can be seen in 3.8 and is based upon the open source library Leafletjs[20]. A user can then assign the sensor type to that created sensor and depending on the sensor template he can fill in the user chosen values. The key/value pairs from the sensor includes Wireless ID and password (created earlier) as well as some fixed constants like sensor sensitivity, re-try rate and more, all depending on the the template chosen.

[20]https://leafletjs.com/ - a JavaScript library for interactive maps (Accessed on: 2019-04-30)

3.6.6 Admin View

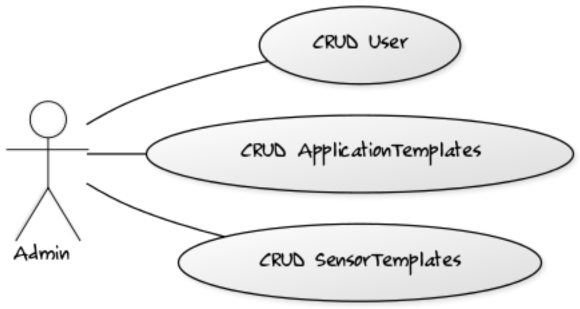


Figure 3.9: Datawiz - Admin

Administrators can perform regular user actions. Additionally they can perform CRUD user management actions, add a new application template and add new sensor types. They can also CRUD applications comments and CRUD new user classes.

3.7 Hardware

Datawiz integrates templates for two hardware sensors, which support WLAN and the MQTT protocol.

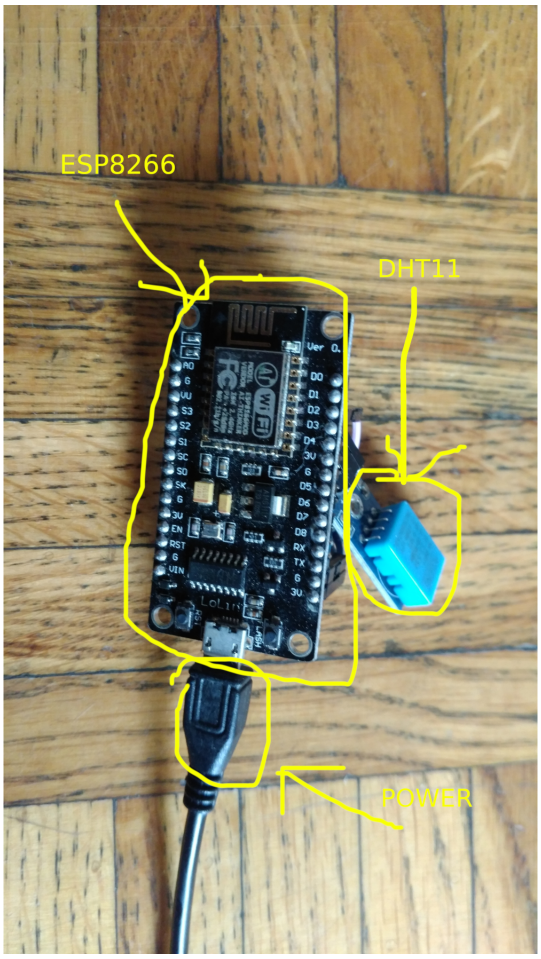


Figure 3.10: ESP8266 with attached DHT11 temperature/humidity sensor

First the ESP8266 (as seen in 3.10) sensor by Espressif Systems[21], is a low cost[22] 32-Bit microcontroller that supports a wide range of sensors (all

[21]https://www.espressif.com/ (Accessed on: 2018-09-30)

[22] prices as of July 15, 2019 were 5 euro/ESP8266 devloper mcu board - price information

powered by 3.3V). Though WLAN is not strictly necessary to connect sensors to MQTT, the ESP8266 is a good reference hardware since it does not need any other bridge to act as an intermediate between the source of the sensor and the destination, which is configured via Node-RED.

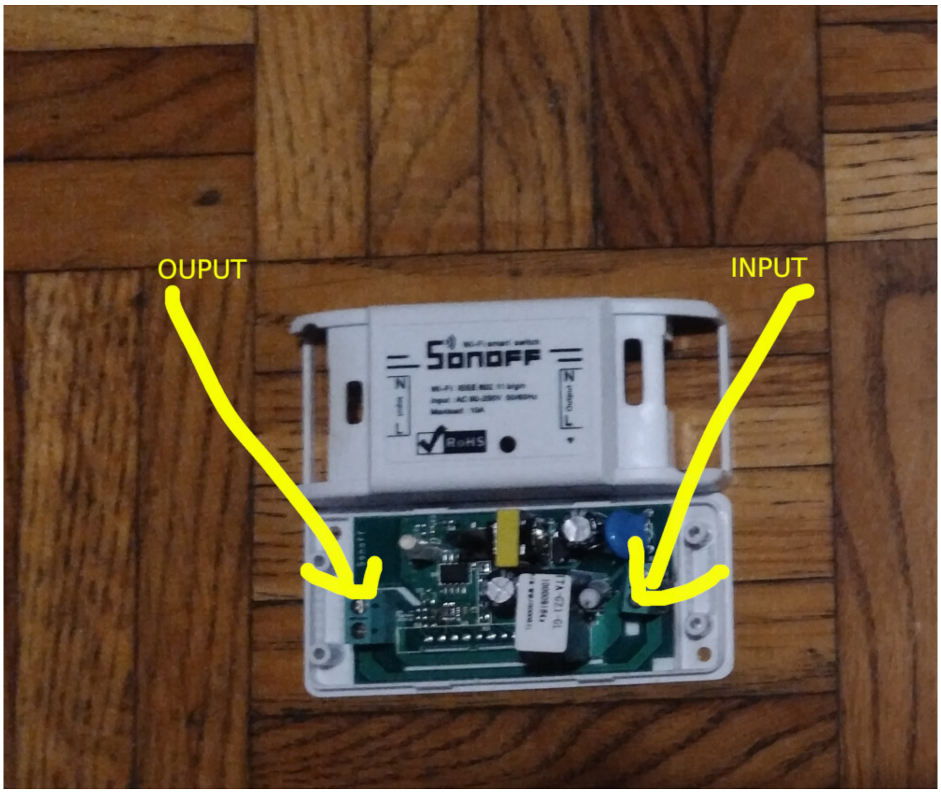


Figure 3.11: Sonoff Basic Switch

Second the Sonoff Basic WiFi Wireless Switch by Itead[23], which is also based upon the ESP8266 and allows switching 220V and can act as an output device.

While in the initial setup of the Sonoff switch, it only connects to proprietary cloud services, it is possible to utilize the alternative Tasmota Firmware[24],

gathered from `http://www.amazon.de`

[23]`https://www.itead.cc/sonoff-wifi-wireless-switch.html` (Accessed on: 2018-09-30)

[24]`https://github.com/arendst/Sonoff-Tasmota` (Accessed on: 2018-09-30)

which allows it to connect to MQTT. These should be seen as a reference implementation for the architecture and more sensors could be added by the administrator.

According to Phonegg [25],as of May 18, 2019, there are 26 Android phones which integrate environmental sensors like thermometer and hygrometer. For Android smartphones there is also a sensor template available in Datawiz. It can use any sensor gathering app, for example the Sensor Node app [26]. This simple application turns any Android phone into a fully integrated sensor that can be used inside Datawiz.

In the wearables market the gadgetbridge[27] is a great project which allows many vendor locked smart wristbands to connect to these devices and transmit data to an Android phone. Many of these wristbands provide data such as: sleep tracking, and heart rate and blood pressure monitoring. At the current state, MQTT is not integrated to the platform but its in one of their open issues [28].

3.8 Applications

Through application templates, Datawiz integrates multiple different application available to users. They are available through the applications tab and offer different configurable options, depending on the application. All application must provide all necessary Kubernetes configuration files to be integrated into Datawiz. These include but are not limited to:

- the namespace (if it is intended to be separated)
- a deployment class
- the ingress route
- a service class

[25]`https://www.phonegg.com/list/302-Cell-Phones-with-Humidity-Sensor` (Accessed on: 2019-03-30)

[26]`https://play.google.com/store/apps/details?id=com.mscino.sensornode&hl=en` (Accessed on: 2019-03-30)

[27]`https://gadgetbridge.org`

[28]`Githubexportofhealthdataissue-https://github.com/Freeyourgadget/Gadgetbridge/issues/553` (Accessed on: 2019-03-30)

- optionally necessary secrets to run the service

Listing 3.2 : Node-RED application with secrets

```
        image: registry.datawiz.host/nodered:v9
        resources:
          requests:
            memory: "128Mi"
            cpu: "250m"
        env:
          - name: USER
            valueFrom:
              secretKeyRef:
                name: secret
                key: USER
          - name: PASSWORD
            valueFrom:
              secretKeyRef:
```

In Listing 3.2 you can see the Node-RED service application, it is configured to get its Configuration file from a secret to provide authentication and authorization to the service. For other programs which also need additional services such as an SSH service to provide configurability for the users, it is possible to include secrets via a path.

This would for example allow a service with a ssh service to included authentication/authorization based upon a public key, which the user uploaded before in their profile.

Listing 3.3 : Scheduling priority for an applications

```
apiVersion: scheduling.k8s.io/v1beta1
description: low priority
kind: PriorityClass
metadata:
  name: low-priority
value: -1000000
```

Optionally deployment specific configuration could also be applied like a PriorityClass, which can be seen in Listing 3.3. Depending on the separation needs of the service a namespace is way to further separate services and it could be either user on the service level (clusterable applications) or for user-based applications, which run separately like Node-RED. In Listing 3.4 you can see a the namespace applied to a userapplication.

Listing 3.4 : Namespace for an application

```
kind: Namespace
apiVersion: v1
metadata:
  name: userapplication
  labels:
    name: userapplication
```

3.8.1 Data Transport

Data transport is one of the core components of Datawiz. As discussed in chapter 1.1.1, MQTT is very suitable for that kind of communication, also because of its different QoS levels. Different kinds of data can be submitted, which can either be numerical sensor data, image/video/audio data (when encoded for example in Base64), or simply plain text.

When data is transmitted it makes its journey through the data flow layer (Node-RED), where it can be enhanced/modified and re-routed. The data transport protocol MQTT, assures that according to the QoS level it reaches its destination, which is either the data storage, in the form of a database or further processing via CI based applications. In case of an output device it can trigger for example an alarm when a threshold is reached or used in conjunction with the reference hardware 3.11 to switch a 220 volts device.

There are many MQTT servers currently on the market, that are scalable and highly available, however not all provide all the features necessary. As a pre-requirements for the MQTT server component it should understand MQTT, provide all the QOS levels and provide clustering possibilities to be integrated into the Kubernetes cluster. Furthermore it should include authorization and authentication possibilities utilizing a database like MySQL or PostgreSQL. This is essential, since every user should only be allowed to access their own MQTT endpoint and MySQL and PostgreSQL should provide good clustering support on their own. Modeling sensor data can be dependent on what sensor someone has and what information is needed. Research has been conducted by Mehmood, Nadeem Qaisar et al.[24], which shows an approach to have a flexible schema for NoSQL databases.

From these requirements the following applications were included to be evaluated.

For setup the following projects were used:

- Kubernetes-vernemq for VerneMQ [32]
- Ekka for EMQ X [33]

[32]https://github.com/nmatsui/kubernetes-vernemq (Accessed on: 2019-03-30)
[33]https://github.com/emqx/ekka (Accessed on: 2019-03-30)

Server	stars [29]	watches [30]	fork [31]	Clustering
RabbitMQ	5825.0	5,825	1,703	only via addon / (QoS2 downgraded)
Mongoose	5439.0	5,439	1,598	no
EMQ X	4597.0	4,597	868	yes
mosquitto	2604.0	2,604	922	clustering only via bridge
emitter	1825.0	1,825	179	wraps MQTT over TCP/SSL
VerneMQ	1738.0	1,738	187	yes
moquette	1241.0	1,241	564	not implemented yet
MQTT.js	221.0	221	809	no
Apache ActiveMQ(mirror)	211.0	211	1,040	unofficial chart/setup broken
mosca	192.0	192	498	no

Table 3.3: Open source MQTT server

- Activemq-kubernetes for ActiveMQ [34]
- Emitter [35]
- GCP click to deploy - RabbitMQ [36]

EMQ X was chosen to be integrated since it offers many benefits is highly scalable/clusterable and can be extended via additional plugins such as:

- emqx_sn - provides MQTT-SN support
- emqx_auth_mysql - user authentication via MySQL
- emqx_coap - CoAP support
- emqx_retainer - retained messages

More plugins can be found on the Getting started page[37].

[34] https://github.com/padmaragl/activemq-kubernetes (Accessed on: 2019-03-30)
[35] https://github.com/emitter-io/emitter/tree/master/deploy/k8s (Accessed on: 2019-03-30)
[36] https://github.com/GoogleCloudPlatform/click-to-deploy/tree/master/k8s/rabbitmq (Accessed on: 2019-03-30)
[37] https://developer.emqx.io/docs/emq/v3/en/getstarted.html (Accessed on: 2019-07-20)

3.8.2 Data Flow Management

The application for data flow management in Datawiz is Node-RED. Node-RED is very slim, has a vast community, many addons and a lower memory footprint.

Some other options are software like Apache Nifi[38], which provide flow based programming.

As of July 16, 2019 Node-RED[39] has over 14.000 weekly downloads[40]. It is written in NodeJS, which was developed by IBM and allows easy visual flow based programming, which allow direct manipulation of the data flow.

With flow based programming, it is possible to connect different input/output and processing nodes. A flow of five sensors connected to Datawiz, can be seen in Figure 3.13. It allows users to easily manage the flow of data between the source and the sink. Other applications like ludwig[41] and Tensorflow [42] can act on the data stream.

As of July 16, 2019, it currently offers over 3400 addons to be integrated into the data flow via its official repository[43].

There is an official Docker image that was adopted to include useful addons required for integration with the other applications of Datawiz. At runtime, via Linux environment variables, credentials like username and password are injected to provide authentication and authorization for users of the given application. To be accessible outside of the cluster, an ingress rule is further applied giving the user a unique DNS name to access their Node-RED instance. The default image built for Node-RED also includes a sample flow file to start with. In the image there is the Node-RED-dashboard included to provide a live dashboard to show which sensors are connected to the system. It also provides historical sensor values in the form of a

[38]`https://nifi.apache.org` (Accessed on: 2019-03-30)
[39]`https://nodered.org` (Accessed on: 2019-03-30)
[40]Data gathered from `https://www.npmjs.com/package/node-red` (Accessed on: 2019-03-30)
[41]`https://uber.github.io/ludwig` (Accessed on: 2019-03-30)
[42]`https://github.com/tensorflow/tensorflow` (Accessed on: 2019-03-30)
[43]`https://flows.nodered.org/` (Accessed on: 2019-07-16)

time series chart via its dashboard addon. To smoothen values further out a widget is included in the data flow to take the median value of the last ten values transmitted. Besides the median function, it is also possible to aggregate/sum data, even to integrate an own function.

This is an easy way of pre-processing data but it can be further extended in a later stage in a computational intelligence application (see 3.8.4).

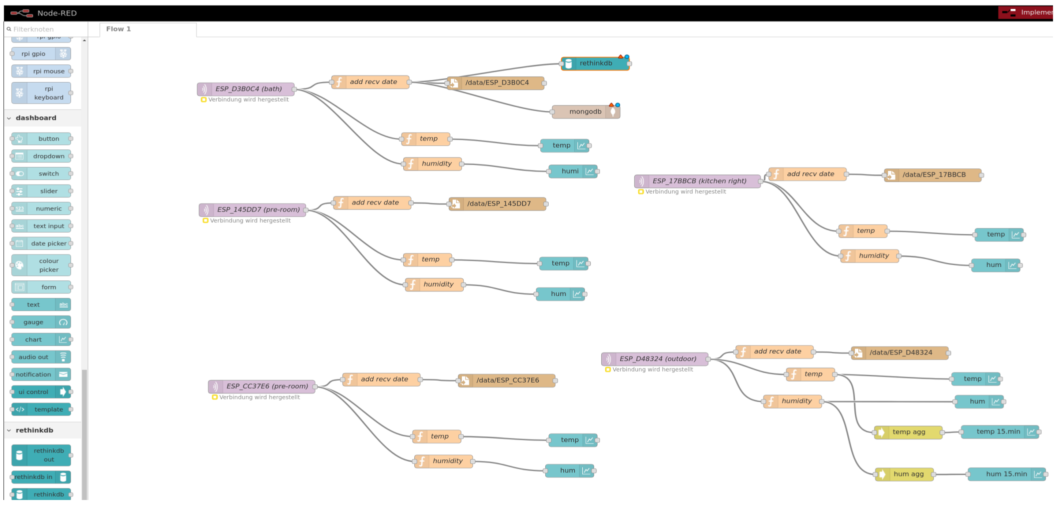


Figure 3.12: Node-red Dataflow with 5 data input/output nodes

3.8.3 Database

Received sensor data is best stored efficiently and flexible in regards to the data format. Sometimes time specific functions can help to provide useful functions to do calculations and compress the data. Yet since the data flow is managed by Node-RED [44], it is possible to write functions in the process which can perform functions like aggregation/median and filtering when the data is arriving.

name	**k8s cluster**	**authentication**	**module** [44]
Influxdb	no in OSE		yes
akumali	no	no	no
kairosdb	yes (unofficial)	via proxy	no
blueflood	no	no	no
khronus	yes	no	no
netflix/atlas	no	no	no
prometheus	yes	no	no
voltaire	no	no	no
btrdb-server	no	no	no
MongoDB	yes	yes	yes
RethinkDB	yes	yes	yes

Table 3.4: Different Database solutions.

The following time series database (as seen in Table 3.4 were evaluated to be included in the selection of a storage for sensor data. They should allow easy inclusion into the Kubernetes cluster either in the form of a HELM chart or as separate YAML scripts. In addition they should allow authentication/authorization on a user/database level so that the storage solution can be clustered and scaled.

[44]Node-Red modules - `https://flows.nodered.org` (Accessed on: 2019-06-30)

MongoDB and RethinkDB, were selected to be used as a sensor data storage layer.

Both of these have clustering functionality built-in and can store data, without needing a schema upfront. Furthermore they also provide authentication and authorization on a database level. Users then get a dedicated login to these databases.

3.8.4 Analytics and New Knowledge

Using computational measures such as statistics and artificial intelligence can be helpful in various scenarios to gain further knowledge from sensor data. It can be used to gain knowledge of the data from the sensor itself, or to detect anomalies or outliers either by incorporating them or by avoiding them depending on the business use case. If the sensor data is coming from connected physical machines it is possible to monitor and proactively intervene in case of a machine failure.

To provide analytics and to generate implicit knowledge templates are provided to plug into the data flow. Two methods were chosen, which are particularly interesting for sensor values, namely forecasting and anomaly detection. The connection is made via MQTT to provide input as well as output of the scores generated. If a certain threshold is triggered, a MQTT messages are sent and depending on the data flow another sensor could act upon it. In the test setup the Sonoff device was used to switch a connected 220 volt device.

The first CI application included is NUPic [45]. It uses a hierarchical temporal memory, which is an approach for intelligence gathering based upon the grid cells in the Neocortex [15]. Asides from other applications it can be used to predict temporal anomalies in time series data.

The second CI application is ludwig[46] by Uber, which is built on top of Tensorflow, a popular machine learning framework. It was chosen ton include two examples:namely one unsupervised learning algorithm and a

[45]https://github.com/numenta/nupic (Accessed on: 2019-03-30)
[46] https://eng.uber.com/introducing-ludwig (Accessed on: 2019-07-02)

supervised one. Ludwig in particular is very flexible since it supports many different types of data and simplifies the model and development process since it is rather abstract. It supports different types of input and output encoders, which include: text, category, numerical, binary, set, sequence, images and temporal time series features.

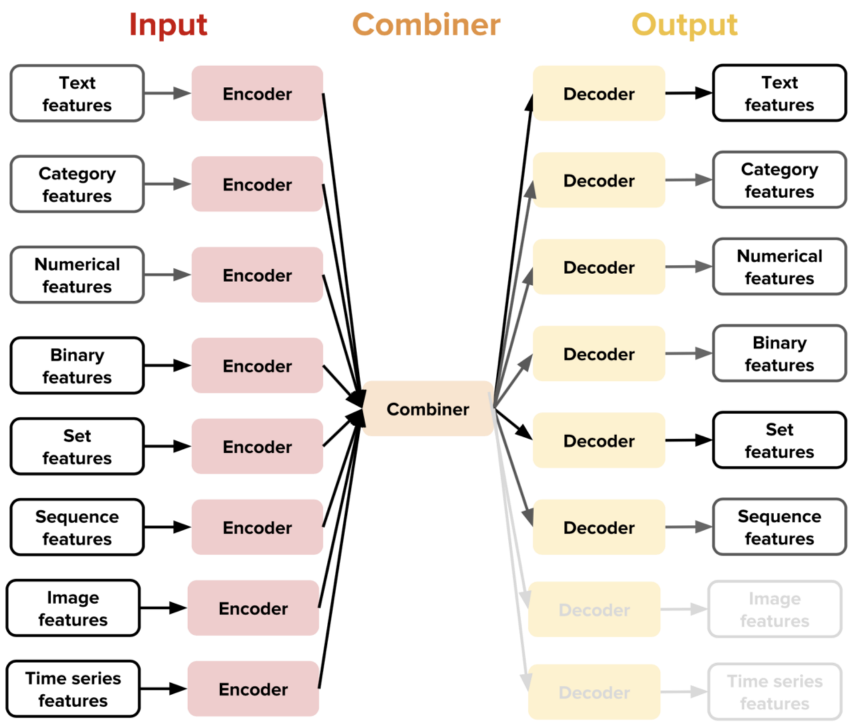


Figure 3.13: Ludwig scalar types [46]

For sensor data, all of them can be applied in various ways The time series feature is particularly interesting, as it allows the user to quickly make predictions based on their time series data for future values.

Since model parameters need to be fine tuned the model should be tested first locally. This should serve as a basis and should provide an idea of how to integrate future applications. Also since the raw data can be accessed directly via the NoSQL database further analysis could be done offline.

4 Evaluation

Evaluation of the platform is important to show how well it can scale as well as where the limitations lie. Several scenarios were executed.

This included tests on how long the system needs to setup, as well as performance metrics and disaster recovery/failover scenarios. Horizontal scalability was evaluated to show how the system performs and the load of new users to scale up.

In case of reduced overload the scale down was also demonstrated. Upgradability of software to improve stability, provide features or to include necessary security fixes are important for any running system. As Node-RED instances are the building blocks for users to create their flows and it is the most memory hungry in regards as a per-user application. The applications on a per-user basis, needed the following memory requirements:

- Node-RED 122MB
- NupiC 188MB
- Ludwig 149MB

The other applications, which had user authentication already built-in had a smaller memory footprint but needed, at least initially the following requirements:

- RethinkDB 18MB initial empty database
- MongoDB 70MB initial empty database

To validate that approach Grafana (See chapter 3.3.2) in combination with Prometheus were used to show Used Memory, as well as CPU Usage along the cluster under different scenarios. The evaluation was performed on the cloud infrastructure described in section 3.3.2. First 3 nodes were included but it was scaled up to nine nodes plus one master node to also show scalability and failover of components.

Cluster creation time via the Terraform utility takes on average 5m29s[1]. Deletion of nodes takes on average only thirty seconds, as it is simply an API call to the cloud provider's API. If though the failover of the storage is calculated in it would take longer depending on how much data needs to be recovered.

On average this would be the configured timeout time plus the new pod creation time.

[1]average value on a 10x runs.

4.1 Initial Setup

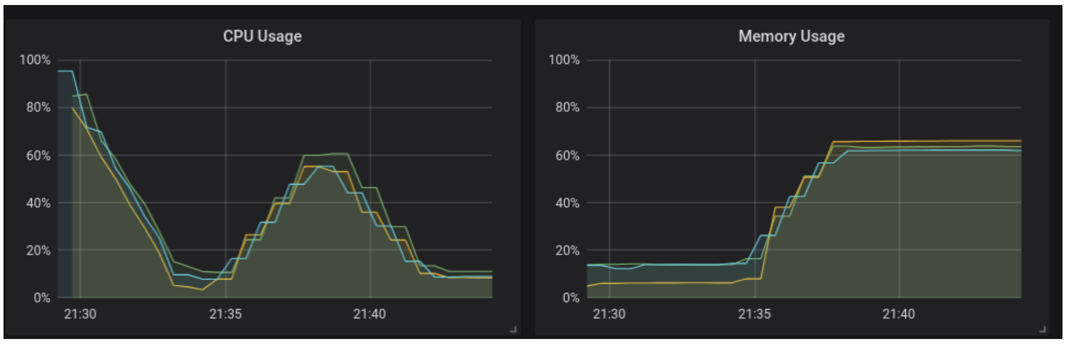


Figure 4.1: Scaling of 150 Node-RED instances CPU/Memory Usage

The scenario you can see in 4.1 demonstrates how the cluster performs after starting 150 new Node-RED instances. It peaks after five minutes in CPU usage with approximately sixty percent usage and memory shows an increase by fourty percent. This is the whole cluster usage, not only one single node. If this is reached in traditional system one could change the server configuration to scale up horizontally by including more CPU/Memory to the machine. Since Kubernetes is also capable of not only doing vertical scaling (using up all the available hardware attached to a node) but also doing horizontal scaling, this is a more scalable approach since one can only put in a limited number of hardware upgrade into a server. The full network usage and disk load and was included in the appendix. (5.2) Since this setup still has time to grow the number of Node-RED instances was increased to 200. This can be seen in Listing 4.2 and it increased to eighty percent cpu usage. User applications, means applications based on a per-user basis like Node-RED or CI-based ones like ludwig/NUPIC. They need a single instances/pod per user and cannot be used by multiple users at once. In contrast clustered applications, are able to internally use clustering to scale and can provide their services to many users by utilizing authentication to ensure that their is still a separation. The following paragraphs, describe up/down and upgradeability for these types of applications inside of Datawiz.

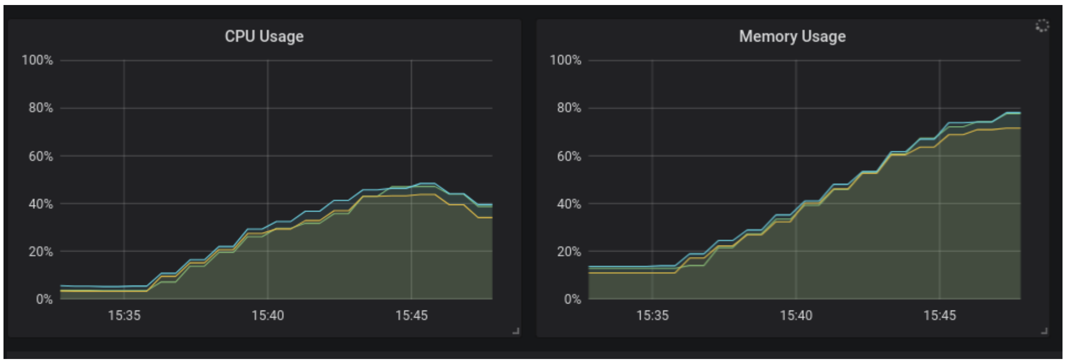


Figure 4.2: Scaling to 200 Node-RED instances CPU/Memory Usage

Listing 4.1 : Terraform apply

```
terraform apply \
 -var nodes=3 \
```

As seen in Listing 4.1, this simple Terraform command utilizes the cloud providers API to increase the number of nodes to four.

Assuming other smaller running services, the limit of 110 pods per nodes has to be considered - though this can be configured through the kube api server.

4.2 Scaling up

User Application:

As one can observe in Figure 4.6 and 4.8, at around 15:20 after the initial 150 new pods, the CPU/network stabilizes but memory still remains constant at around 60%-65%. A new node was added, utilizing the Terraform apply command with nodes=4, as seen in Listing 4.1. During the initialization phase, the CPU/network usage is quite high, because despite of all the services which are initialized like the core Kubernetes services, the disk is also prepared to be included in the Ceph cluster. Memory usage however stays first at zero percent and when adding 50 new Node-RED pods to the cluster it quickly gets distributed to the new node.

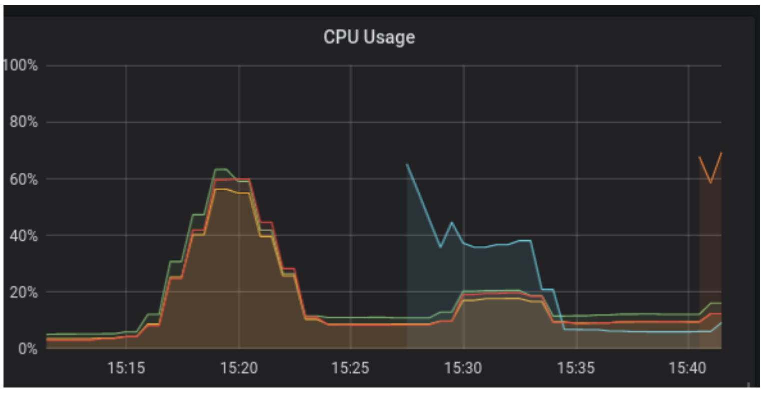


Figure 4.3: Scaling up Test- CPU Usage across nodes

Clustered Applications:

Scaling up a a clustered application like EMQ X deployment can be achieved by running the command seen in Listing 4.2. Based upon CPU utilization, automatic scaling could also be achieved utilizing the horizontal pod autoscaler (see Listing 4.3 for an example). The same works for every other application in the cluster.

Listing 4.2 : Scale deployment EMQ X

```
kubectl scale deployment emqx-chart --replicas=10
```

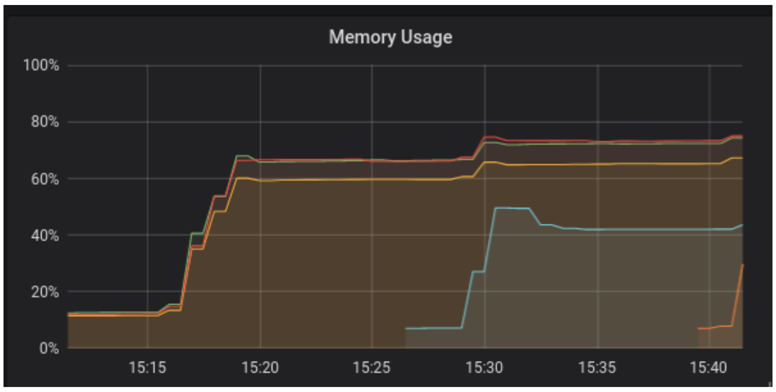


Figure 4.4: Scaling up Test - Memory Usage across nodes

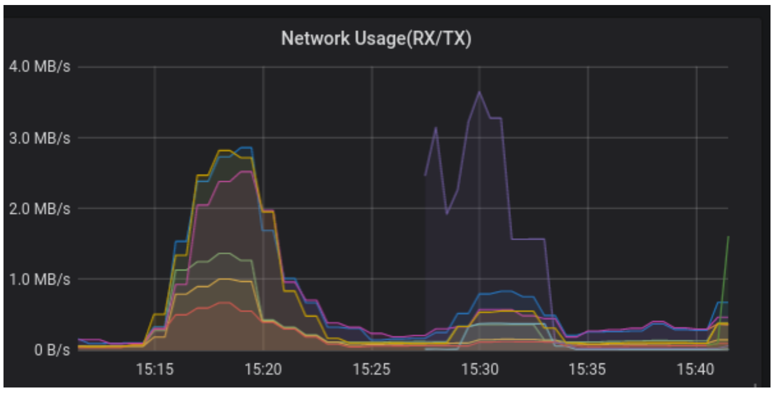


Figure 4.5: Scaling up Test - Network Usage across nodes

Listing 4.3 : Autoscale deployment EMQ X

```
kubectl scale deployment emqx-chart --replicas=10
```

4.3 Scaling Down/Node Failure

Scaling down a cluster is as important as scaling up. Therefore if scaling down is necessary this is possible as well, when for example the hardware gets faulty or the node is not responding.

Another case could be if the userbase is shrinking and not that many services are needed anymore.

User Application:

In this test four additional nodes were added to the cluster at around 15:40, which can be observed in 4.6 as the CPU spikes initially for the new node and after that at 15:50 when three additional nodes were added. After there was enough memory available in the cluster to account for a node failure, node four (4.7), which is the line in blue was killed with a hard shutdown. After the node becomes unresponsive, which can also be observed via the CLI utility, while running kubectl get nodes (see Listing 4.4), pods get redistributed to other healthy nodes.

Listing 4.4 : Node gets unresponsive

```
NAME                STATUS     ROLES    AGE    VERSION
datawiz-master-1    Ready      master   150m   v1.13.7
datawiz-node-1      Ready      <none>   143m   v1.13.7
...
datawiz-node-4      NotReady   <none>   29m    v1.13.7
datawiz-node-5      Ready      <none>   11m    v1.13.7
```

As seen in Listing 4.5 the pods get terminated and are at the same time recreated on different nodes. In Listing 4.6 ceph also noticed the node failure and depending on how much data is lost and how many copies should be stored the data gets recovered.

Listing 4.5 : Pods get terminated

```
user101-nodered-6478d75579-28x5h    1/1      Terminating
0         10m      10.36.0.42    datawiz-node-4    <none>
<none>
```

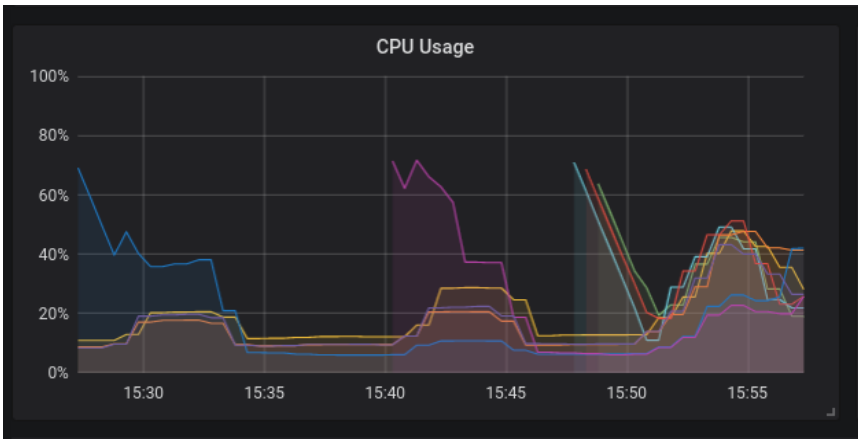


Figure 4.6: Scaling down Test- CPU Usage across nodes

```
user104-nodered-646dbd9475-zqmkv    1/1     Terminating
0           10m      10.36.0.43     datawiz-node-4    <none>
<none>
user11-nodered-775b4bfd59-v98zg     1/1     Terminating
0           12m      10.36.0.12     datawiz-node-4    <none>
<none>
```

Listing 4.6 : Ceph Health check

```
osd.1 reported immediately failed by osd.2
do_prune osdmap full prune enabled
Health check failed: 1 osds down (OSD_DOWN)
Health check failed: 1 host (1 osds) down (OSD_HOST_DOWN)
accept timeout, calling fresh election
```

If for some reason there is not enough space to house all containers on one node according to the priorityClassName, pods get evicted. As seen in Listing 4.7, the pods were evicted for user125, and simultaneously got recreated on another node.

Listing 4.7 : Pods get evicted

```
node-red       user121-nodered-c5965f948-wjlf2
0/1      Evicted              0           9m28s
```

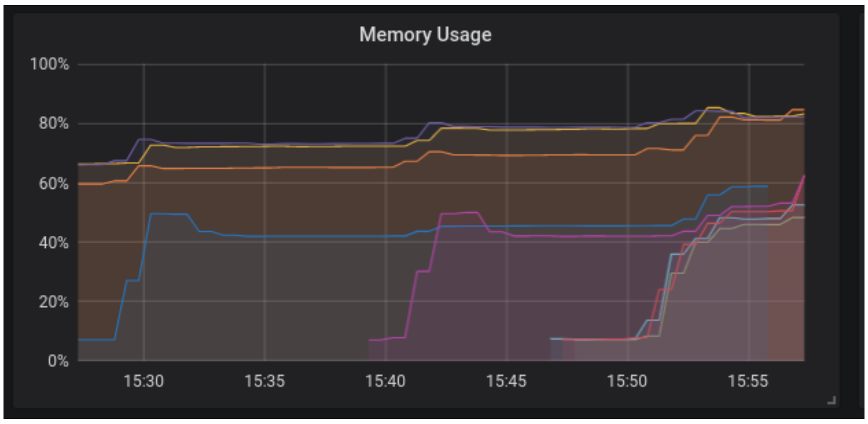


Figure 4.7: Scaling down Test - Memory Usage across nodes

```
node-red          user122-nodered-67bdd557f6-45gnw
1/1       Running                0          9m26s
node-red          user123-nodered-849bcf468-dcz6f
1/1       Running                0          9m25s
node-red          user124-nodered-ccd85f64-cgq98
1/1       Running                0          9m24s
node-red          user125-nodered-86b8b65d84-jtrh9
0/1       Evicted                0          9m23s
node-red          user125-nodered-86b8b65d84-xc2qm
0/1       ContainerCreating      0          2m41s
```

Clustered Applications:

Scaling down the number of pods for clustered applications, like for example the MQTT server the same command seen in Listing 4.2 can be used by reducing the number of desired pods. If the horizontal pod autoscaler is used, then this would also occur automatically. The pod is then removed from the EMQ X cluster and messages are routed to different pods.

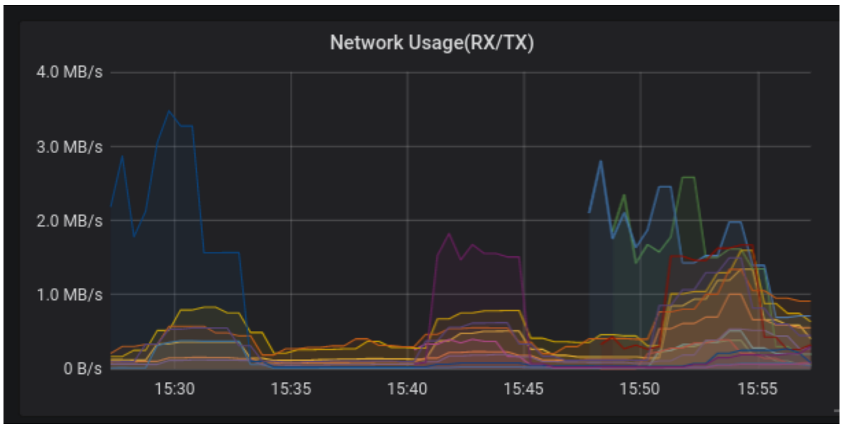


Figure 4.8: Scaling down Test - Network Usage across nodes

4.4 Upgradeability

Upgradeability is important to ensure all components are up-to-date for feature enhancement and get the latest security patches. For the helm based charts, the CLI interface, provides a way to upgrade these charts via helm upgrade.

For the user based applications, like Node-RED and the CI-based applications, a rollout of a new release can be performed in conjunction with a strategy. In this way the maximum unavailable pods can be configured. In this case, it is important to ensure that in the spec fields the replicas are set to a number greater than one, so that at least one instance is reachable all the time.

This process is currently not automated, but could be further improved utilizing a CI/CD based solution like Jenkins to automatically update the applications. A more simpler solution, Drone [2] could also be used, depending on the number of applications included.

If Kubernetes itself needs an upgrade, where the components like API Server, controller Manager, Scheduler, Kube Proxy, CoreDNS need to updated since it has to be assured that at least two master nodes available.

[2] `https://drone.io` (Accessed on: 2019-06-20)

When updating for a major version, detailed instructions are available from the official documentation (see for example the upgrade instructions from v1.12 to v1.13[3]). These instructions often break things and manual tasks have to be performed.

4.5 Data-aggregation

For evaluation purposes 5x ESP8266 sensors (see chapter 3.7, Figure 3.10.) with an attached DHT11 temperature/humidity sensor.

The code uploaded to the ESP8266 3.10 sensor, is based upon the example found in [4] and is pretty straightforward. Some variables were added and as the user utilizes the device code generator the variables are replaced with their respective values.

It gathers temperature/humidity data and send it to the configured MQTT server, that was set before. All variables in the template are replaced with the values which the user has configured.

It connects to WIFI, gathers all the sensor information and sends it to the MQTT server. MQTTs connection was not possible due to the limitation of memory of the microcontroller. It is however possible to validate at least the fingerprint of the MQTT server, but still this is not ideal.

Four sensors were placed indoors, while one sensor was located outdoors. The payload used was encoded as a JSON string, and can be seen in Figure 4.8.

Listing 4.8 : ESP8266 sensor sample

```
{"temperature":24,"humidity":19,"hostname":"ESP_145DD7","recv_date":"2019-03-22T14:14:30.681Z"}
{"temperature":24,"humidity":19,"hostname":"ESP_145DD7","recv_date":"2019-03-22T14:14:41.960Z"}
{"temperature":24,"humidity":19,"hostname":"ESP_145DD7","recv_date":"2019-03-22T14:14:53.236Z"}
{"temperature":24,"humidity":19,"hostname":"ESP_145DD7","recv_date":"2019-03-22T14:15:04.512Z"}
{"temperature":24,"humidity":19,"hostname":"ESP_145DD7","recv_date":"2019-03-22T14:15:15.790Z"}
```

[3]https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade-ha-1-13 (Accessed on: 2019-06-20)

[4]https://github.com/adafruit/DHT-sensor-library/blob/master/examples/DHTtester/DHTtester.ino (Accessed on: 2019-06-20)

The ESP8266 sensor were located in the following way:

- ESP_D48324 (outdoor)
- ESP_17BBCB (indoor kitchen)
- ESP_145DD7 (indoor pre-room)
- ESP_CC37E6 (indoor bath)
- ESP_D3B0C4 (indoor main room)

They were located in Graz (Styria).

Humidity and temperature was captured in the time frame from 17.3.2019 until 30.3.2019, with the default setting of one measurement per second.

The temperature and humidity charts for raw, resampled data can be found in Appendix (5.2). The attached DHT11 was not that reliable, which can be seen in the raw chart of sensor CC37E6 (.15).

To mitigate this, a function was included in Node-RED to calculate the median value over the last five minutes. In the end the outdoor sensor delivered the most promising data, and was used in chapter 4.6.

Listing 4.9 : Hostname ESP8266 + mqtt connect via username/password

```
WiFi.hostname().toCharArray(hostname, 50);
if ( client.connect(hostname, "user1", "user1password") ) {
...
```

The names represent their internal hostnames and this function is included in the template, as it is a good way to ensure uniqueness along the ESP8266 based sensors, without naming all of them beforehand. However they can be named and should be situated when creating the sensor to not lose track of them. As seen in figure 4.9 the client is then connected to the MQTT server with username and password provided.

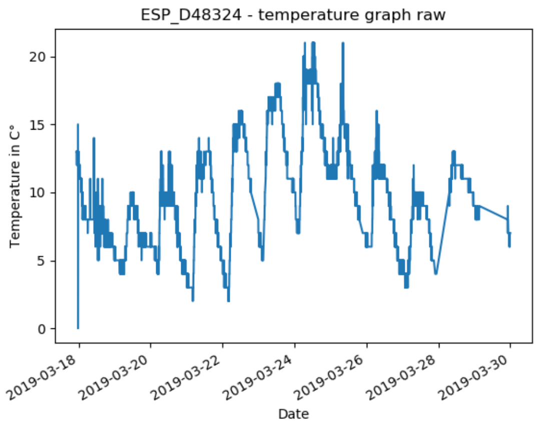


Figure 4.9: ESP_D48324 temperature graph raw

set	loss	mean_squared_error	mean_absolute_error	r2	error
train	0.471	0.1471	0.1309	0.0071	0.0098
validation	0.0093	0.0093	0.0488	0.0046	0.0047
test	0.0097	0.0097	0.0447	0.0056	0.0085

Table 4.1: Ludwig performance graph (trained on ESP_D48324 temperature graph raw)

4.6 Forecasting & Anomaly Detection

The forecasting algorithm was tested on the temperature data of the outdoor sensor (ESP_D48324 - see Figure .2 and .4 in the Appendix).

The dataset was split into training, validation and testset. With the last ten temperature values the algorithm was able to predict the next value with a mean absolute error value of 0.0488 on the validation set.

After the validation set did not improve anymore and the algorithm stopped after convergence, therefore the following performance was achieved.

Utilizing the default template included in Datawiz with an anomaly threshold of 0.8 the following datapoints, which can be seen in Figure 4.11 were gathered. Some of the datapoints does not some to have any correlation to

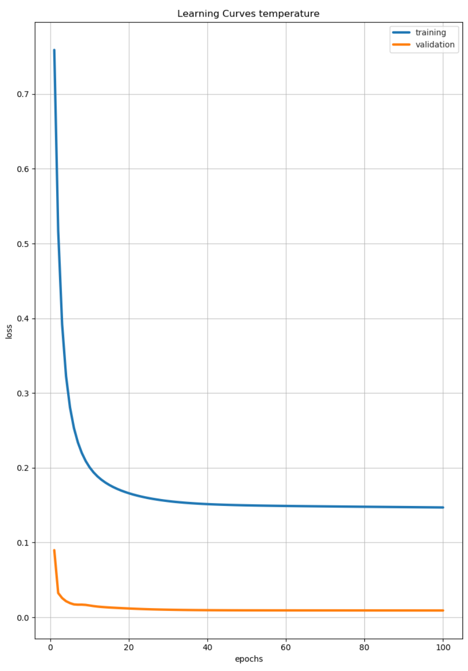


Figure 4.10: Learning curve Ludwig

the underlying data (point1,2,3) but this maybe due to the limitation of the dataset.

On March 24, 2019, a cold front appeared that was moving towards Graz, and the temperature dropped significantly afterwards. On that day an anomaly was detected. Parameter tuning could also be performed offline to further enhance the performance regarding the desired outcome. However since this is very specific in regards to which sensors and more data would be needed in any case this test was not further improved.

In a factory, sensors could be attached to different machines to detect a hardware failure early on and to prevent further damage by enabling/disabling a process on another machine.

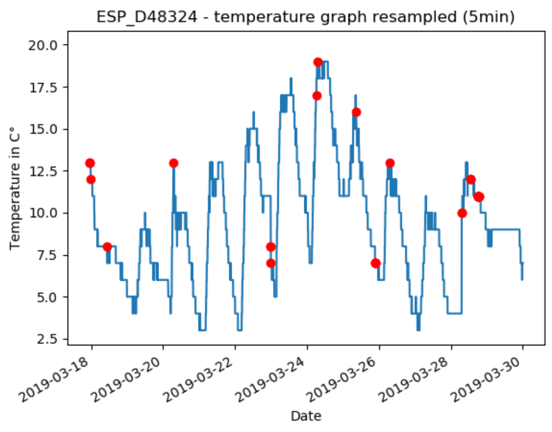


Figure 4.11: ESP_D48324 Temperature graph with NUPIC anomaly data points (0.35 threshold)

4.7 Limitations

General limitations of, which Kubernetes can scaled to are : [5] No more than 5,000 nodes/150,000 total pods/300,000 total containers and the 100 pods per node limit.

As of July 18.2019 there are over 2130 issues [6] open, many of which are still critical and hinder cluster stability. Furthermore if storage is attached, since it is a Read-Write-Once type of storage and the pod is still not terminated on one node, a new pod cannot be created by Kubernetes. This is a known limitation of all storage drivers, which use the Read-Write-Once type.

Regarding high availability an unstable node flapping state can occur, which I did multiple in the load test, when under high CPU/memory usage. That means a node is switching between Ready/NotReady state and failover does not occur.

The handling of addition/deletion of nodes need to be added to the load balancer, which is not handled automatically. This is due to the nature of

[5] `https://kubernetes.io/docs/setup/best-practices/cluster-large` (Accessed on: 2019-06-20)

[6] `https://github.com/kubernetes/kubernetes/issues` (Accessed on: 2019-08-07)

the setup, since the external load balancer is not connected via an API to the internal one (NGINX). If multiple nodes are already added in the load balancer still every NodePort forwards the port to the appropriate service. Appropriate API communication however can be added, is is sometimes already in place in multiple cloud provider solutions like GKE and AWS.

In regards to the storage layer the CAP theorem, holds true here too, as with the chosen NoSQL databases (RethinkDB and MongoDB) consistency will be the bottleneck. This is just a limitation to be aware of but, since writes are as important as reads for the sensor data, and because for data in one can always plug into the MQTT stream, this is only partially a problem.

Further described in the setup in 3.3, the network connection was a bottleneck.

Different sensors need different scenarios, like pre-filtering, data aggregation, dealing with null values and much more. Therefore with the Node-RED interface, a user can adopt their use case to use the necessary functions needed to ultimately get what they wanted out of their data. This can be implicit knowledge, dashboarding/visualization features or prevention of sensor failure via anomaly detection.

Time data of messages can be tricky, because of the sensors like the ESP8266 do not have an internal real-time clock.

5 Conclusions and future work

5.1 Conclusions

There are many applications necessary to run a scalable sensor data framework, and while it seemed trivial at first the whole ecosystem of possible candidates is very complex. As shown in chapter 4.5 more quality sensors are needed to gather data, otherwise afterwards processing is not possible and therefore no knowledge can be inferred. This thesis showed the potential of combining various open source projects that, are currently out in the market.

Maintainability is easier with this solution, but there are still many things to consider when running this system on a larger user scale. As described in chapter 4.4, upgradeability of the stack with a possible CI/CD would give flexibility ensuring that the shipped software is also up to date and security issues are handled. It was shown in section 4.2 and 4.3, that the underlying orchestrator (Kubernetes) performs well in handling additional nodes in the cluster to accommodate more users and applications in the system. Application templates could also be further optimized to make use of the horizontal AutoScaling feature based on CPU/memory consumption [1].

Although this was done with only one command, if the underlying cloud provider supports it this could be further improved with solutions like the cluster autoscaler [2]. While in old legacy applications sensor data was often transmitted via HTTP, it was shown in conjunction with the dataflow

[1] `https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/` (Accessed on: 2019-08-07)

[2] `https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler` (Accessed on: 2019-08-07)

solution Node-RED, how a powerful pub/sub schema can be used with modern IoT applications.

As the internet moves towards a more decentralized version of it, with cryptocurrencies arising like IOTA (which provide MQTT support at their core), an integration where one would connect Datawiz to one of this solutions can also be considered.

In 2019 there is also research going on [26], as it is more cost-effective and provides an even more scalable approach. As [16] pointed out, there are still many limitations, like Shipping Data to Code, hardware limitations and caching. They evaluated further in a case study that for model training AWS lambda was 21x slower than simply using AWS EC2 instances. Also it is not vendor neutral since most use their proprietary functions as a service solution. Some open source solution already exist like OpenFaaS [3] and it can for sure be leveraged in some regards.

[3]`https://github.com/openfaas/faas` (Accessed on: 2019-08-07)

5.2 Future work

To ensure inter-connectivity more sensors could be included, which would also mean to provide more templates for them and ensure more applications which could be used in that context. Although more than numeric values were tested in this thesis, image/audio/video data could also be transmitted through MQTT and than stored and processed.

More specialized applications would be necessary to ensure that knowledge can be inferred easily from these sources of data. While currently it is convenient to just use the default CI application settings, more fine granular optimizations are needed. This could possible be mitigated by allowing users to submit their own applications, and while this means more challenges like ensuring stability and security in a secured lab environment for a fixed number of authorized personal this could be an option.

If a more commercial settings is desired it is important to ensure that also users are billed accordingly to their usage data. Since system maintainability is important, a solution for log data aggregation/analysis could be included to ensure that all applications are running and working correctly.

With the system developed for this thesis is a broader distribution of it is possible due to easier handling, reliability, maintainability and scalability. It was shown and evaluated that it works with the ESP8266 sensor but more platforms/templates are to be included to further enhance the coverage.

The next step for Datawiz is to publish its components as open source with more potential use cases and user adoption.

Appendix

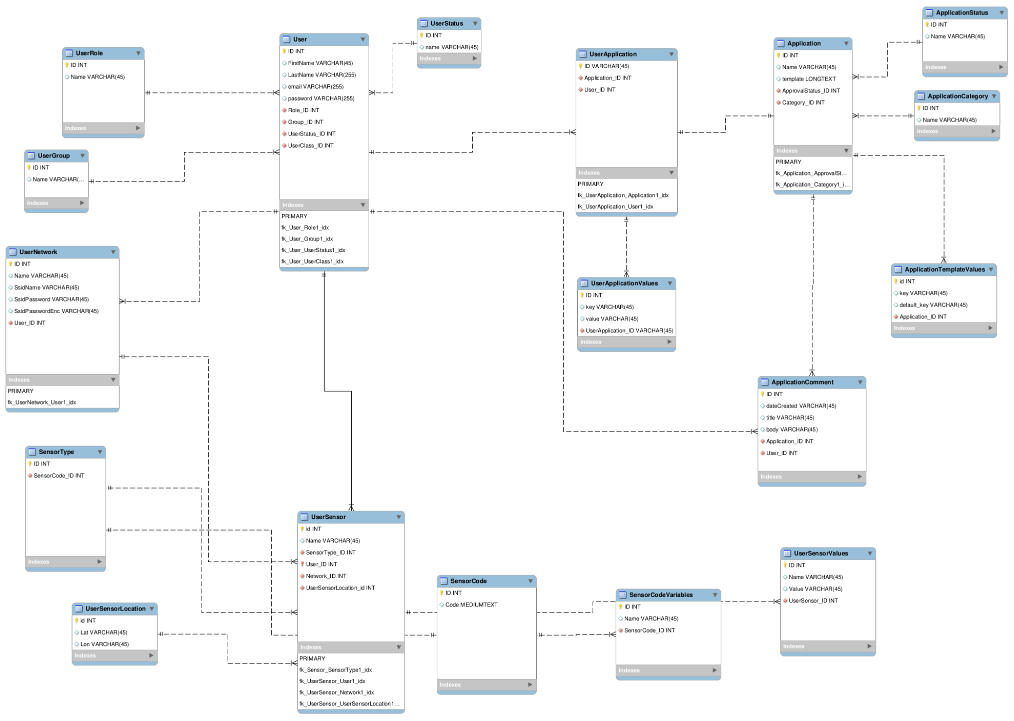


Figure .1: Database ER model Datawiz

Server	**stars**	**watches**	**fork**
RabbitMQ	5825	5825	1703
Mongoose	5439	5439	1598
EMQ X	4597	4597	868
Mosquitto	2604	2604	922
Emitter	1825	1825	179
VerneMQ	1738	1738	187
Moquette	1241	1241	564
MQTT.js	221	221	809
Apache ActiveMQ(mirror)	211	211	1040
Mosca	192	192	498
MqttWk	75	75	29
Jmqtt	69	69	35
Apache ActiveMQ Artemis	52	52	444
HBMQTT	36	36	131
RSMB	34	34	23
GnatMQ	29	29	76
Trafero Tstack	3	3	0

Table .1: Mqtt server

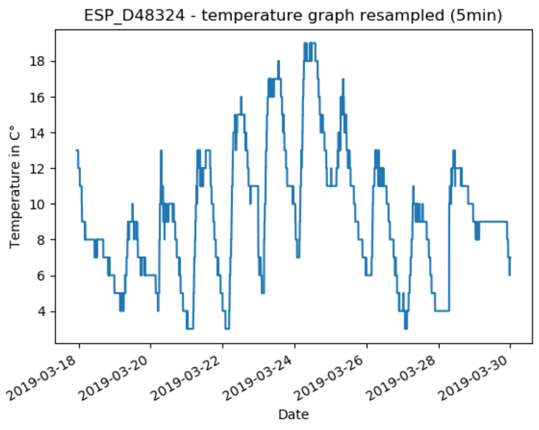


Figure .2: ESP_D48324 Temperature graph resampled

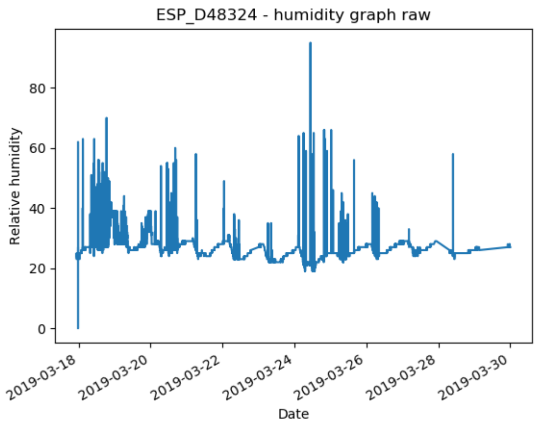


Figure .3: ESP_D48324 Humidity graph raw

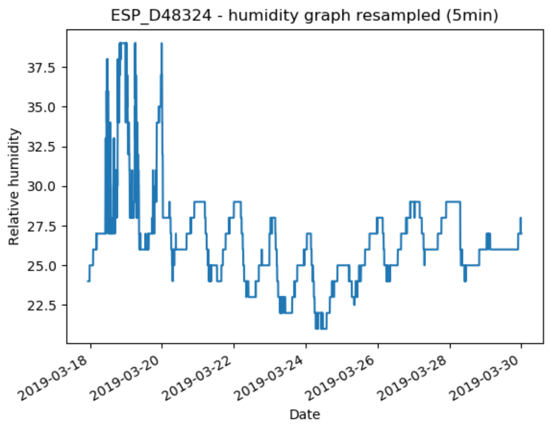


Figure .4: ESP_D48324 Humidity graph resampled

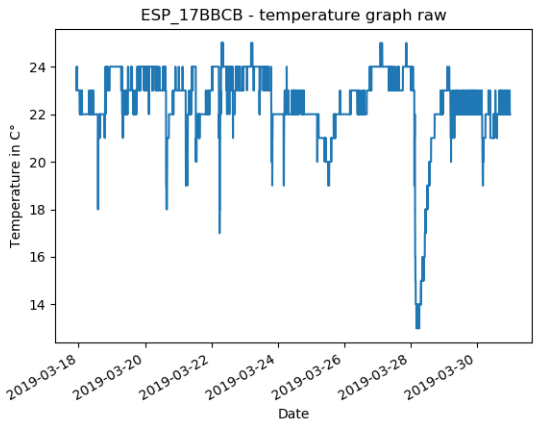


Figure .5: ESP_17BBCB Temperature graph raw

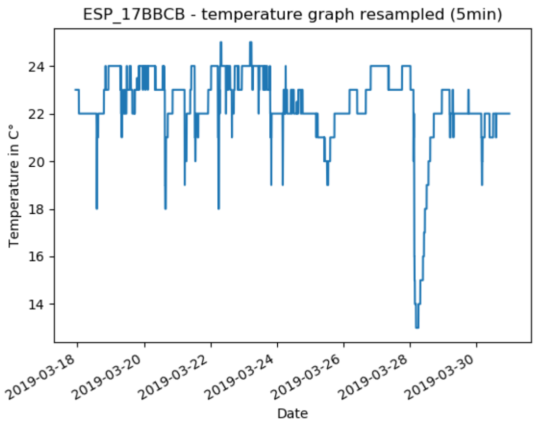


Figure .6: ESP_17BBCB Temperature graph resampled

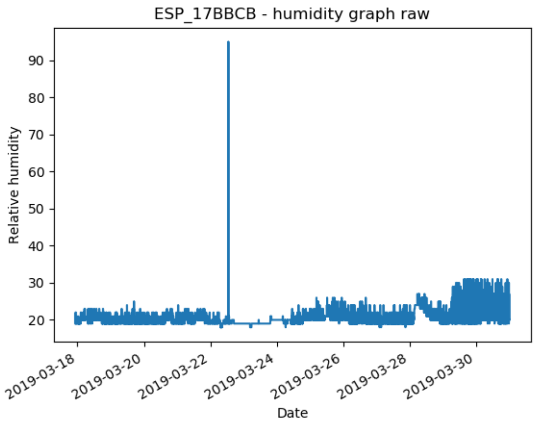


Figure .7: ESP_17BBCB Humidity graph raw

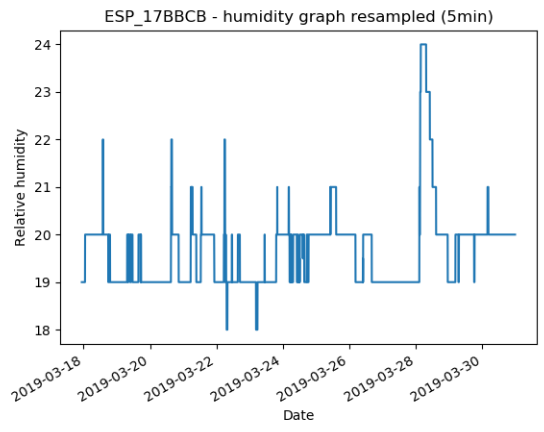


Figure .8: ESP_17BBCB Humidity graph resampled

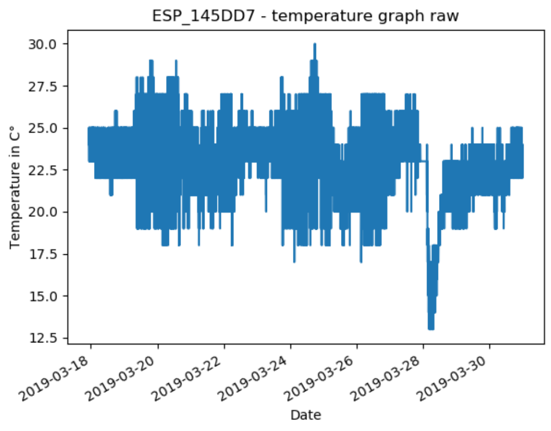


Figure .9: ESP_145DD7 Temperature graph raw

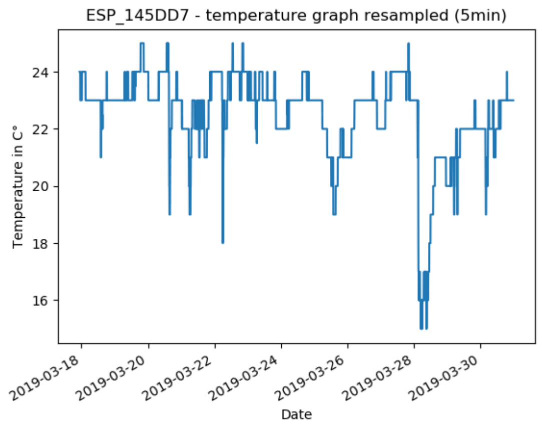


Figure .10: ESP_145DD7 Temperature graph resampled

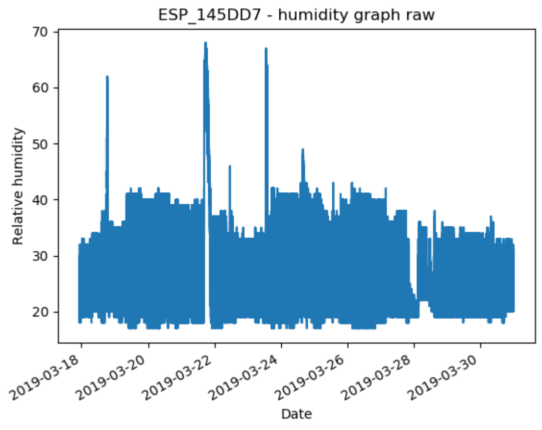


Figure .11: ESP_145DD7 Humidity graph raw

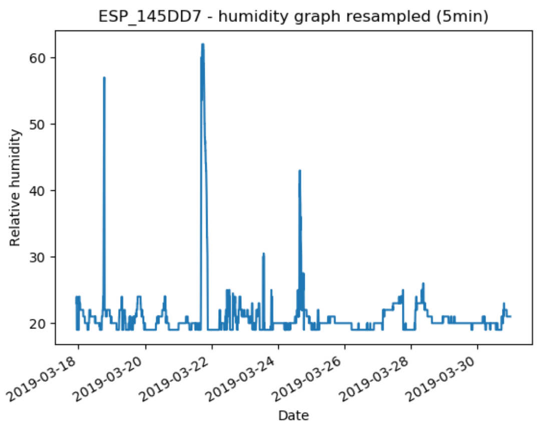


Figure .12: ESP_145DD7 Humidity graph resampled

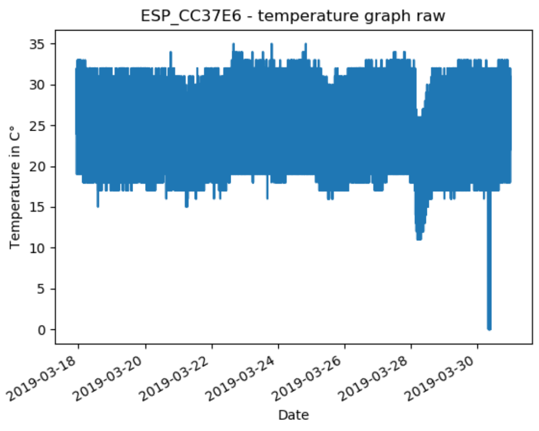


Figure .13: ESP_CC37E6 Temperature graph raw

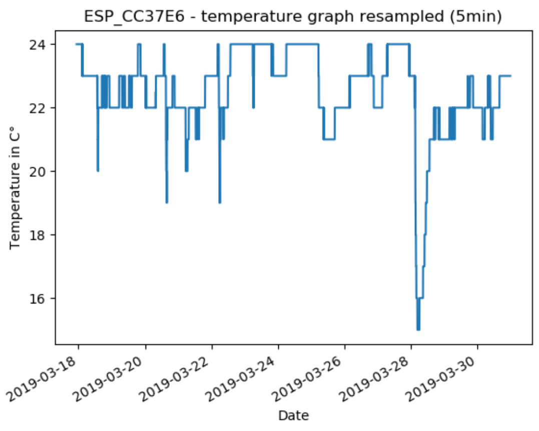


Figure .14: ESP_CC37E6 Temperature graph resampled

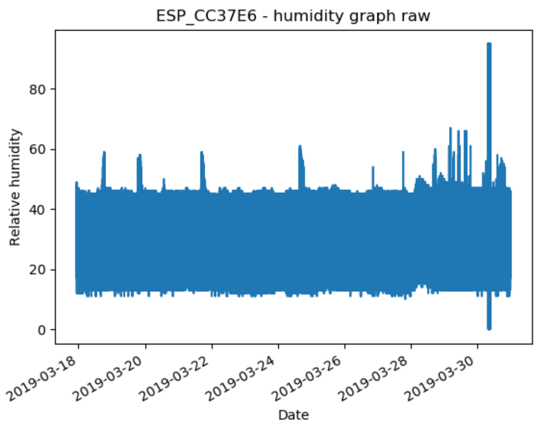


Figure .15: ESP_CC37E6 Humidity graph raw

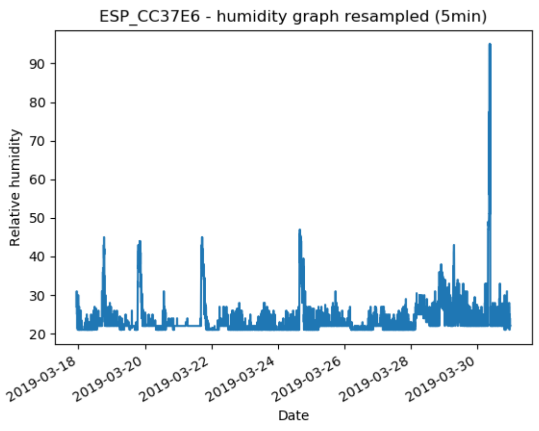


Figure .16: ESP_CC37E6 Humidity graph resampled

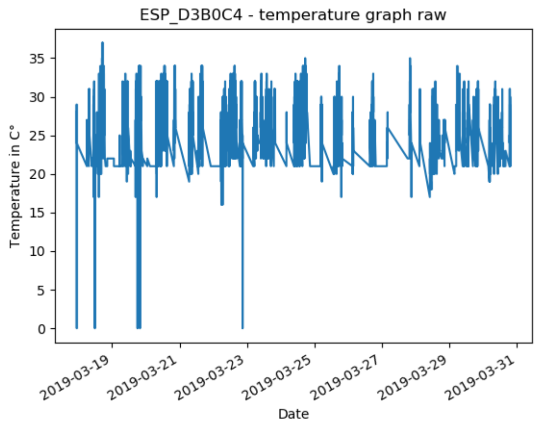


Figure .17: ESP_D3B0C4 Temperature graph raw

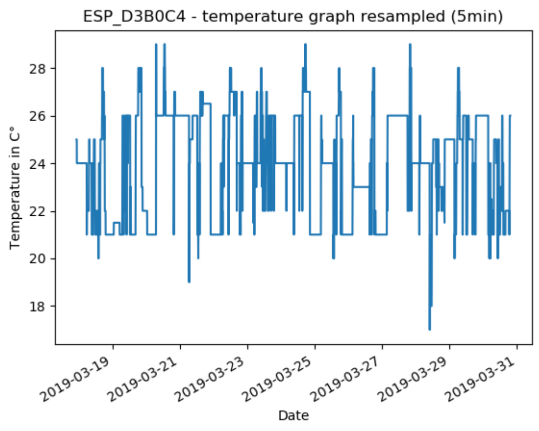


Figure .18: ESP_D3B0C4 Temperature graph resampled

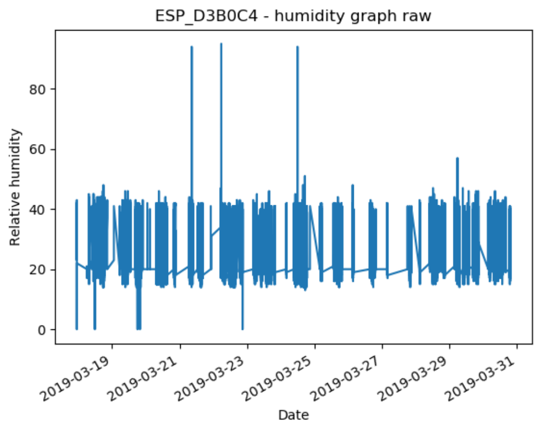


Figure .19: ESP_D3B0C4 Humidity graph raw

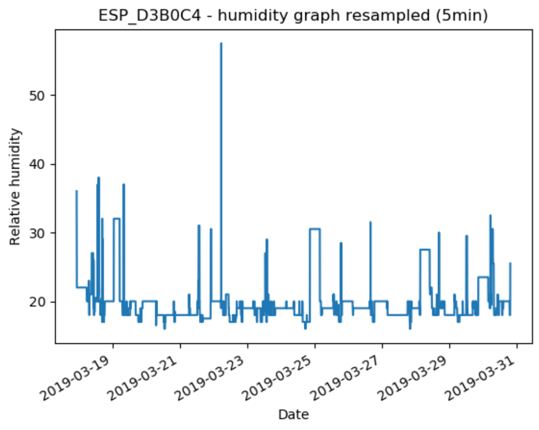


Figure .20: ESP_D3B0C4 Humidity graph resampled

Bibliography

[1] Brendan Michael Abbott. *A security evaluation methodology for container images*. 2017 (cit. on p. 23).

[2] The Kubernetes Authors. *Kubernetes API Reference Docs*. Nov. 2018. URL: `https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.12/` (visited on 06/11/2018) (cit. on p. 15).

[3] The Kubernetes Authors. *Kubernetes Components - Kubernetes*. May 2018. URL: `https://kubernetes.io/docs` (visited on 23/08/2018) (cit. on pp. 12, 19).

[4] The Kubernetes Authors. *Persistent volumes - Kubernetes*. 2018. URL: `https://kubernetes.io/docs/concepts/storage/persistent-volumes/` (visited on 25/09/2018) (cit. on p. 17).

[5] Glenn Berry. *Scaling SQL Server 2012*. June 2012. URL: `https://www.pass.org/eventdownload.aspx?suid=1902` (visited on 30/08/2018) (cit. on p. 6).

[6] Michael Blackstock and Rodger Lea. 'Fred: a hosted data flow platform for the IOT built using node-red'. In: *Proceedings of MoTA* (2016) (cit. on p. 31).

[7] André B Bondi. 'Characteristics of scalability and their impact on performance'. In: *Proceedings of the 2nd international workshop on Software and performance*. ACM. 2000, pp. 195–203 (cit. on p. 6).

[8] Charles Calvin Byers, Gonzalo Salgueiro and Joseph Michael Clarke. *Orchestration of cloud and fog interactions*. US Patent App. 15/371,038. June 2018 (cit. on p. 6).

[9] Donald D. Chamberlin and Raymond F. Boyce. 'SEQUEL: A Structured English Query Language'. In: *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. SIGFIDET '74. Ann Arbor, Michigan: ACM, 1974, pp. 249–264. DOI: `10.1145/800296.811515`. URL: `http://doi.acm.org/10.1145/800296.811515` (cit. on p. 24).

[10] cnf. *9 Kubernetes Security Best Practices Everyone Must Follow*. Jan. 2019. URL: `https://www.cncf.io/blog/2019/01/14/9-kubernetes-security-best-practices-everyone-must-follow/` (visited on 14/01/2019) (cit. on p. 23).

[11] cnf. *NoSQL is a horseless carriage*. Nov. 2009. URL: `https://de.slideshare.net/northscale/nosqloakland-200911021` (visited on 14/01/2019) (cit. on p. 25).

[12] Rat der Europäischen Union Europäisches Parlament. *EUR-Lex - 32016R0679R(02) - EN - EUR-Lex*. May 2018. URL: `https://eur-lex.europa.eu/legal-content/DE/ALL/?uri=CELEX:32016R0679R(02)` (visited on 23/08/2018) (cit. on p. 2).

[13] Nicolas Ferry et al. 'GeneSIS: Continuous Orchestration and Deployment of Smart IoT Systems'. In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2019, pp. 870–875 (cit. on p. 31).

[14] Miniwatts Marketing Group. *World Internet Users and 2019 Population Stats*. June 2019. URL: `https://www.internetworldstats.com/stats.htm` (visited on 30/06/2019) (cit. on p. 1).

[15] Jeff Hawkins et al. 'A Framework for Intelligence and Cortical Function Based on Grid Cells in the Neocortex'. In: *bioRxiv* (2018). DOI: `10.1101/442418`. eprint: `https://www.biorxiv.org/content/early/2018/10/13/442418.full.pdf`. URL: `https://www.biorxiv.org/content/early/2018/10/13/442418` (cit. on p. 61).

[16] Joseph M. Hellerstein et al. *Serverless Computing: One Step Forward, Two Steps Back*. 2018. eprint: `arXiv:1812.03651` (cit. on p. 80).

[17] Docker Inc. *Container Runtime with Docker Engine — Docker*. 2019. URL: `https://www.docker.com/products/container-runtime` (visited on 20/07/2019) (cit. on p. 8).

[18] *Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)*. Nov. 2016. URL: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ (visited on 22/09/2018) (cit. on p. 2).

[19] Joab Jackson. *World Internet Users Statistics and 2018 World Population Stats*. 2017. URL: https://www.slideshare.net/gmccance/cern-data-centre-evolution (visited on 09/12/2017) (cit. on p. 20).

[20] Asad Javed. 'Container-based IoT Sensor Node on Raspberry Pi and the Kubernetes Cluster Framework'. en. G2 Pro gradu, diplomityö. 2016-08-24, pp. 8 + 68. URL: http://urn.fi/URN:NBN:fi:aalto-201608263055 (cit. on p. 30).

[21] Endah Kristiani et al. 'Implementation of an Edge Computing Architecture Using OpenStack and Kubernetes'. In: *Information Science and Applications 2018*. Ed. by Kuinam J. Kim and Nakhoon Baek. Singapore: Springer Singapore, 2019, pp. 675–685. ISBN: 978-981-13-1056-0 (cit. on p. 31).

[22] N. Leavitt. 'Will NoSQL Databases Live Up to Their Promise?' In: *Computer* 43.2 (Feb. 2010), pp. 12–14. ISSN: 0018-9162. DOI: 10.1109/MC.2010.58 (cit. on p. 24).

[23] Gavin McCance. *World Internet Users Statistics and 2018 World Population Stats*. Nov. 2012. URL: https://www.slideshare.net/gmccance/cern-data-centre-evolution (visited on 01/10/2018) (cit. on p. 6).

[24] Nadeem Qaisar Mehmood, Rosario Culmone and Leonardo Mostarda. 'Modeling temporal aspects of sensor data for MongoDB NoSQL database'. In: *Journal of Big Data* 4.1 (Mar. 2017), p. 8. ISSN: 2196-1115. DOI: 10.1186/s40537-017-0068-5. URL: https://doi.org/10.1186/s40537-017-0068-5 (cit. on p. 56).

[25] Claus Pahl. 'Containerisation and the PaaS Cloud'. In: *IEEE Cloud Computing* 2 (June 2015), pp. 24–31. DOI: 10.1109/MCC.2015.51 (cit. on p. 5).

[26] Josef Spillner and Mohammed Al-Ameen. *Serverless Literature Dataset*. Zenodo dataset (3rd revision) at https://doi.org/10.5281/zenodo.1175423. Apr. 2019. DOI: https://doi.org/10.5281/zenodo.1175423 (cit. on p. 80).

[27] Ravn Steinholt. *A study of Linux Containers and their ability to quickly offer scalability for web services*. Mar. 2015. URL: https://www.duo.uio.no/bitstream/handle/10852/45127/1/Steinholt-Master.pdf (cit. on p. 30).

[28] *Large-scale cluster management at Google with Borg*. Bordeaux, France, 2015 (cit. on p. 11).

[29] Adam Wiggins. *The Twelve-Factor App*. 2017. URL: https://12factor.net/ (visited on 14/07/2019) (cit. on p. 27).

[30] Mikko1 Yliniemi. 'CO2.io : a scalable sensing infrastructure to monitor the effects of climate change on air quality'. en. MA thesis. 2018-09-06, pp. 49 + 10. URL: http://jultika.oulu.fi/files/nbnfioulu-201809062738.pdf (cit. on p. 31).