

# DIPLOMARBEIT

Gesamtprojekt

## **NetBat – Entwicklung eines anpassbaren Netzwerk Monitoring Systems**



# NetBat

Marcus Edlinger

5AHEL

Betreuer: Prof. Dipl.-Ing. Robert Vogl

Rafael Haigermoser

5AHEL

Betreuer: Prof. Dipl.-Ing. Robert Vogl

Anna Victoria Krupa

5AHEL

Betreuer: Prof. Dipl.-Ing. Robert Vogl

Kooperationspartner: conova communications GmbH

ausgeführt im Schuljahr 2019/20

---

Abgabevermerk:

Datum: 03.04.2020

übernommen von:

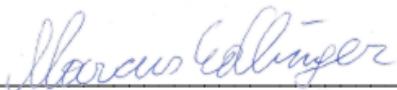
<b>htl</b> bildung mit zukunft	HÖHERE TECHNISCHE BUNDESLEHR- UND VERSUCHSANSTALT Salzburg
	Elektronik und Technische Informatik

## Eidesstattliche Erklärung

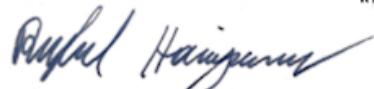
Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Salzburg, am 03.04.2020

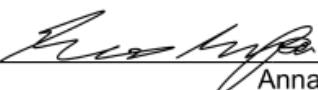
Verfasserinnen / Verfasser:



Marcus Edlinger



Rafael Haigermoser



Anna Victoria Krupa

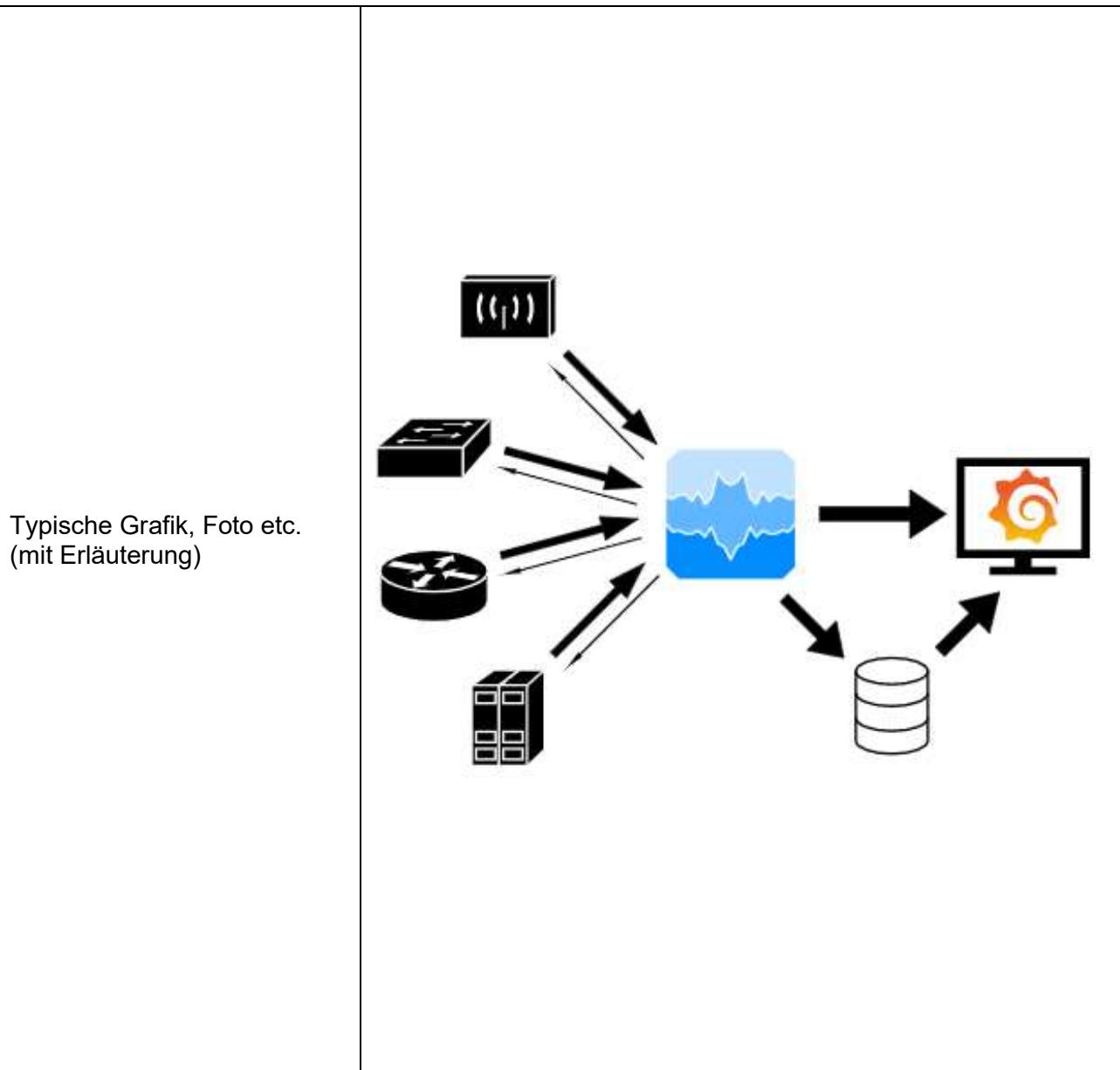
**DIPLOMARBEIT  
DOKUMENTATION**

Namen der Verfasserinnen / Verfasser	Marcus Edlinger Rafael Haigermoser Anna Victoria Krupa
Jahrgang Schuljahr	5AHEL 2019/20
Thema der Diplomarbeit	Entwicklung eines anpassbaren Netzwerk Monitoring Systems
Kooperationspartner	conova communications GmbH

Aufgabenstellung	Die Komplexität moderner Netzwerke erfordert kontinuierliche Verfügbarkeit. Um Probleme frühzeitig zu erkennen, muss das System permanent überwacht und sein Zustand mitgeschrieben werden.
------------------	---

Realisierung	Der zentrale Server liest mittels diverser Protokolle Messwerte von den Netzwerkgeräten ein und speichert diese in einer Datenbank. Ein Webserver zeigt diese Informationen grafisch an, überwacht die einkommenden Daten und alarmiert bei Ausfällen oder Überlast. Um repräsentativ zu testen, wird ein Testnetzwerk mit unterschiedlichen Netzwerkkomponenten aufgebaut und möglichst realistisch belastet.
--------------	--

Ergebnisse	Ein Netzwerküberwachungssystem, welches Betriebsdaten von Netzwerkgeräten ausliest, abspeichert, überwacht und in ansprechender Weise darstellt.
------------	--



Teilnahme an Wettbewerben, Auszeichnungen	
--	--

Möglichkeiten der Einsichtnahme in die Arbeit	Bibliothek der HTBLuVA Salzburg Infoserver der HTBLuVA Salzburg
--	--

Approbation (Datum / Unterschrift)	Prüferin / Prüfer	Direktorin / Direktor Abteilungsvorständin / Abteilungsvorstand
---------------------------------------	-------------------	--

## DIPLOMA THESIS

### Documentation

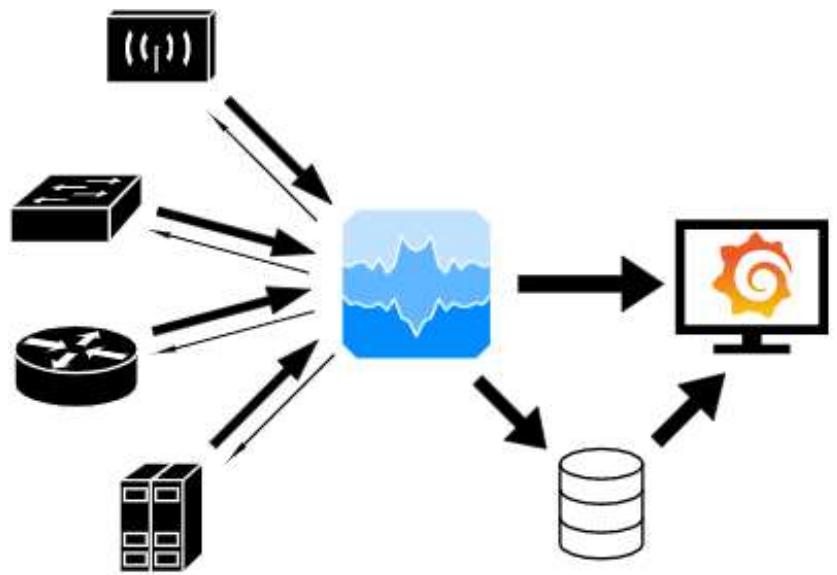
Author(s)	Marcus Edlinger Rafael Haigermoser Anna Victoria Krupa
Form Academic year	5AHEL 2019/20
Topic	Development of a customizable network monitoring system
Co-operation Partners	conova communication GmbH

Assignment of Tasks	The complexity of modern networks demands continuous availability. The status of Systems has to be monitored and logged without interruptions to detect problems and allow fast intervention.
---------------------	---

Realisation	The central server utilizes a variety of protocols to read status information from network devices and stores them into a database. A webserver displays this data graphically, compares it with nominal values and alerts the user on failure or overload. A test network is set up and stressed realistically to gain representative test results.
-------------	--

Results	The result is a network monitoring system that reads, stores and monitors status information from network devices and displays it in an appealing way.
---------	--

Illustrative Graph, Photo  
(incl. explanation)



Participation in Competitions  
Awards

Accessibility of  
Diploma Thesis

HTBLuVA Salzburg library  
HTBLuVA Salzburg info server

Approval  
(Date / Sign)

Examiner

Head of College  
Head of Department

## Vorwort

Oft genug haben wir in den letzten Jahren festgestellt, dass die meisten Menschen Netzwerktechnik nicht gerade spannend finden und nur wenige Bekannte und Verwandte sich über eine „kurze“ Einführung in die Grundlagen dieser Technik freuen. Da wir als Techniker gelernt haben, mit Zahlen zu jonglieren, fällt uns oft nicht einmal auf, dass wir unsere Zuhörer schon längst verloren haben.

Aufgrund dieser Erfahrungen haben wir uns entschlossen, ein Tool zu schaffen, welches Daten ansprechend grafisch darstellt, um, etwas übertrieben formuliert, der ganzen Menschheit zu zeigen, wie schön Zahlen und insbesondere jene im Zusammenhang mit Netzwerktechnik sein können. Zum Glück sind wir nicht die ersten, die eine solche Idee gehabt haben, und haben daher die Möglichkeit, uns die Errungenschaften vieler anderer Projekte zu Nutze zu machen. Wir freuen uns, dass als Resultat unserer Arbeit ein Stück Software entstanden ist, welches einerseits Faszination und Interesse auslöst und andererseits die Administration eines Netzwerks erheblich erleichtert, da Techniker Probleme schneller erkennen und dadurch früher lösen können.

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderwärtige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint, soweit es für die Aussage erforderlich ist.

## Danksagung

Wir bedanken uns bei Prof. DI Robert Vogl für die Betreuung unseres Projekts. Zusätzlich möchten wir uns auch bei der conova communications GmbH bedanken, die uns großzügig mit Servern und Netzwerkhardware unterstützt, ein Büro zur Verfügung gestellt und bei technischen Problemen weitergeholfen hat. Für das Team „Network & IT Security Solutions“ nennen wir namentlich DI Roman Schnatter, Armin Fisslthaler BSc und Philipp Kendlbacher BSc, denen wir besonderen Dank aussprechen wollen. Im Weiteren bedanken wir uns auch bei dem Team „Linux & Open Technology Solutions“ und allen anderen conova Mitarbeitern, die wir im letzten Jahr schätzen gelernt haben.

Außerdem möchten wir uns bei folgenden Personen für das Korrekturlesen unserer Arbeit bedanken: Prof. Mag. phil. Mario Wegscheider, Prof. Mag. rer. nat. Eveline Edlinger Bakk. phil. und Prof. Mag. theol. Georg Haigermoser.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>14</b>
1.1 Ausgangslage . . . . .	14
1.2 Zielsetzung . . . . .	14
<b>2 Organisation</b>	<b>16</b>
2.1 Projektteam . . . . .	16
2.2 Individuelle Aufgabenstellung und Zeitplan . . . . .	17
2.2.1 Marcus Edlinger . . . . .	17
2.2.2 Rafael Haigermoser . . . . .	18
2.2.3 Anna Victoria Krupa . . . . .	19
<b>3 Datenauslese</b>	<b>20</b>
3.1 Verwendete Netzwerkprotokolle . . . . .	20
3.1.1 ICMP . . . . .	20
3.1.2 SNMP . . . . .	21
3.1.3 HTTP/HTTPS . . . . .	21
3.1.4 Munin . . . . .	22
3.2 Besondere Datenquellen . . . . .	22
3.2.1 NetBat Logo . . . . .	22
3.2.2 Befehlsausgabe . . . . .	22
<b>4 Das Datenbanksystem</b>	<b>24</b>
4.1 Wahl von InfluxDB . . . . .	24
4.2 Datenbankschema . . . . .	25
4.3 Downsampling . . . . .	25
4.3.1 Ansatz 1 . . . . .	26
4.3.2 Ansatz 2 . . . . .	27

4.3.3	Ansatz 3	28
<b>5</b>	<b>NetBat-Server</b>	<b>30</b>
5.1	Programmiersprache und Runtime	30
5.2	Aufgaben des Servers	31
5.3	Aufbau	31
5.4	Kernbibliotheken	33
5.4.1	config	33
5.4.2	db	35
5.4.3	device	35
5.4.4	grafana	50
5.4.5	jstools	54
5.5	Modulare Datenquellen	55
5.5.1	ICMP	56
5.5.2	SNMP	57
5.5.3	HTTP/HTTPS	59
5.5.4	Munin	61
5.5.5	NetBat Logo	63
5.5.6	Befehlsausgabe	63
5.5.7	Implementierung weiterer Datenquellen	64
5.6	Datenbankabstraktion	66
5.6.1	InfluxDB	67
5.6.2	Konsolenausgabe	67
5.7	Globale Konfiguration	68
5.7.1	Kommandozeilenparameter	68
5.7.2	Environment-Konfiguration	68
5.7.3	Allgemeine Konfiguration	69
5.7.4	Globale Grafana-Konfiguration	70
5.8	Device-Konfiguration	71
5.8.1	Templates	73
5.8.2	Datenbank	74
5.8.3	Grafana	80
5.9	Fehlermeldungen	83

<b>6 Grafische Darstellung</b>	<b>88</b>
6.1 Allgemeines zu Grafana . . . . .	88
6.2 Globale Einstellungen . . . . .	88
6.3 Dashboards . . . . .	89
6.4 Panels . . . . .	89
6.5 Rows . . . . .	92
6.6 Playlist . . . . .	92
6.7 Topologieansicht . . . . .	92
6.8 Umsetzung im Testnetzwerk . . . . .	93
6.8.1 Ordner . . . . .	93
6.8.2 Dashboards . . . . .	93
6.8.2.1 Overview . . . . .	93
6.8.2.2 Topologie . . . . .	94
6.8.2.3 Devices . . . . .	95
6.8.2.4 NetBat Icon . . . . .	96
6.9 Templates . . . . .	96
6.9.1 grafana_global . . . . .	97
6.9.2 router/cisco/ASR-920-4SZ-A . . . . .	98
6.9.3 other/munin-generic . . . . .	99
6.9.4 other/netbat-icon . . . . .	103
<b>7 Alarmierung</b>	<b>104</b>
7.1 Regeln . . . . .	104
7.2 Bedingungen . . . . .	105
7.3 Fehlerbehandlungen . . . . .	106
7.4 Benachrichtigungen . . . . .	106
7.5 Beispiele . . . . .	107
7.5.1 RTT . . . . .	107
7.5.1.1 Vorbereitung . . . . .	107
7.5.1.2 Alarmkonfigurationen für RTT . . . . .	108
7.5.2 Festplattenauslastung . . . . .	112
7.5.2.1 Alarmkonfigurationen der Festplattenauslastung . . . . .	112

<b>8 Das Testnetzwerk</b>	<b>117</b>
8.1 Anforderungen an das Netzwerk . . . . .	117
8.2 Netzwerktopologie . . . . .	118
8.2.1 Router und Firewall . . . . .	119
8.2.1.1 Barracuda . . . . .	119
8.2.1.2 Cisco . . . . .	120
8.2.1.3 VyOS . . . . .	122
8.2.1.4 Laptop des Administrators . . . . .	124
8.2.2 Server (VMs) . . . . .	125
8.2.2.1 Grafana . . . . .	125
8.2.2.2 Proxy . . . . .	127
8.2.2.3 Server und Client . . . . .	129
8.2.2.4 Kraken . . . . .	131
8.2.2.5 Testen mit GUI . . . . .	134
8.2.2.6 NetBox . . . . .	135
8.2.2.7 Salt master . . . . .	137
8.3 Verwendete Programme und Alternativen . . . . .	139
8.3.1 Betriebssysteme . . . . .	139
8.3.1.1 Cisco IOS . . . . .	139
8.3.1.2 VyOS . . . . .	139
8.3.2 Webserver und Proxys . . . . .	139
8.3.2.1 Apache . . . . .	140
8.3.2.2 NGINX . . . . .	140
8.3.2.3 HAProxy . . . . .	140
8.3.3 Netzwerktraffic generieren . . . . .	140
8.3.3.1 hping3 . . . . .	140
8.3.3.2 iperf . . . . .	140
8.3.3.3 TRex . . . . .	141
8.3.4 Kommandozeilen Werkzeug . . . . .	141
8.3.4.1 Terminal . . . . .	141
8.3.4.2 Netzwerk debugging . . . . .	141
8.3.4.3 Configuration Management . . . . .	142

<b>Literaturverzeichnis</b>	<b>143</b>
<b>Abbildungsverzeichnis</b>	<b>146</b>
<b>Tabellenverzeichnis</b>	<b>148</b>
<b>Abkürzungsverzeichnis</b>	<b>149</b>
<b>Begleitprotokolle</b>	

# 1 Einleitung

## 1.1 Ausgangslage

Aufgrund der Digitalisierung werden viele kritische Systeme von Computern gesteuert, man denke zum Beispiel an Behörden, Banken oder Energieversorger. Ausfälle in diesen Bereichen wären verheerend. Aus diesem Grund ist es wichtig, die permanente Verfügbarkeit und Funktionsfähigkeit zu gewährleisten. Um das zu erreichen, muss der Zustand der Komponenten laufend überwacht und überprüft werden. Auf diese Weise können Probleme und auf solche zusteuende Tendenzen sofort erkannt und Gegenmaßnahmen eingeleitet werden.

Um dies zu ermöglichen, verfügen alle Netzwerkgeräte über Schnittstellen, die es erlauben, Betriebsinformationen auszulesen. Hierbei tritt die Schwierigkeit auf, dass eine Vielzahl an Schnittstellen und Protokollen mit unterschiedlichen Versionen und Dialekten verwendet werden. Manche Hersteller nutzen leichte Abwandlungen von Standards oder ganz eigene Herangehensweisen. Dieser Variantenreichtum im Bereich des Netzwerkmonitorings erschwert den Einsatz eines einzelnen Tools.

## 1.2 Zielsetzung

Um Abhilfe für das geschilderte Problem zu schaffen, ist das folgende als gewünschtes Endergebnis des Projekts bei Besprechungen des Projektteams und der Partnerfirma festgelegt worden:

Es soll ein Softwarepaket entwickelt werden, welches

- die Unterstützung möglichst vieler Hersteller und Protokolle ermöglicht,
- leicht zu bedienen ist,
- die ausgelesenen Daten übersichtlich und ästhetisch ansprechend darstellt,
- den Anwender beim Erreichen kritischer Werte alarmiert.

Um das Monitoring System zu testen, soll ein Testnetzwerk aufgebaut werden, das das Überprüfen der genannten Punkte ermöglicht. Im Rahmen dieser Tests soll es auf repräsentative Weise belastet werden.

Als optionaler Zusatz sollte die Software sowohl generisch konzipiert als auch modular aufgebaut sein und daher nicht nur Rechenzentren, sondern auch Kraftwerke oder „smart homes“ überwachen können.

## 2 Organisation

### 2.1 Projektteam



Abbildung 2.1: Projektteam. Von links nach rechts: Marcus Edlinger, Anna Victoria Krupa, Prof. Dipl.-Ing. Robert Vogl, Rafael Haigermoser

Um die gesetzten Ziele in der vorgegebenen Zeit zu bewerkstelligen, ist ein gut kooperierendes Team zusammengestellt worden, dessen drei Mitglieder jeweils andere Kompetenzbereiche besitzen.

- Marcus Edlinger: Projektleiter und Spezialist für Softwareentwicklung
- Rafael Haigermoser: Spezialist für Netzwerktechnik und Systemmanagement
- Anna Victoria Krupa: Spezialistin für Design und graphische Benutzeroberflächen

## 2.2 Individuelle Aufgabenstellung und Zeitplan

Basierend auf den genannten Kompetenzprofilen sind die Ziele und die damit verbundenen Aufgaben in Aufgabenbereiche unterteilt und auf die Teammitglieder aufgeteilt worden. Um einen zeitlichen Überblick über das Projekt zu behalten und die Kooperation zwischen diesen Teilen durch Synchronisierung zu erleichtern, sind Zeitpläne erstellt worden.

### 2.2.1 Marcus Edlinger

Aufgabenbereiche:

- Organisation
  - Projektmanagement
  - Einrichtung des Versionmanagements von Software und Dokumentation
- Implementierung der Software
  - Programmkernel und Modulsystem
  - Abfragemodule
  - Datenbankschnittstelle
  - Entwurf der Konfigurationsstruktur

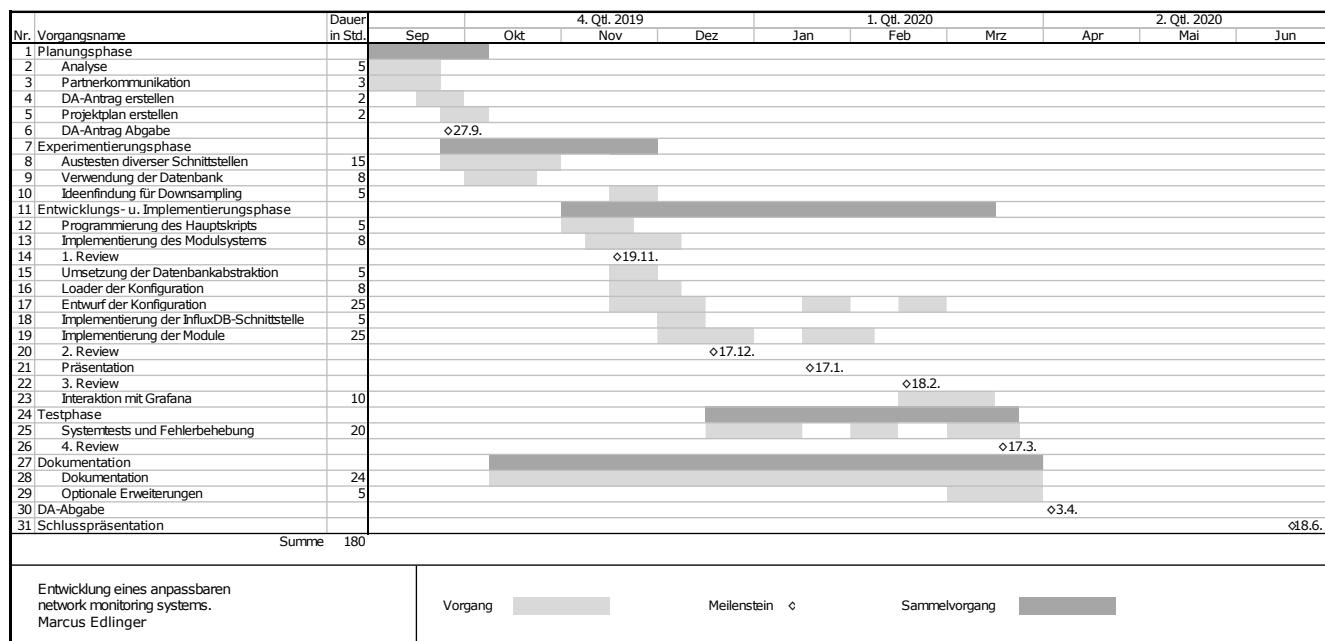


Abbildung 2.2: Zeitplan von Marcus Edlinger

## 2.2.2 Rafael Haigermoser

Aufgabenbereiche:

- Aufbau des Testnetzes
  - Entwurf der Topologie
  - Aufsetzen der Geräte und der Testumgebung
  - Wartung der Testinfrastruktur
  - Aufsetzen und Wartung des Grafana-Servers
- Testnetzwerkbelastung
  - Ermittlung von Testmethoden
  - Entwicklung und Durchführung von Tests

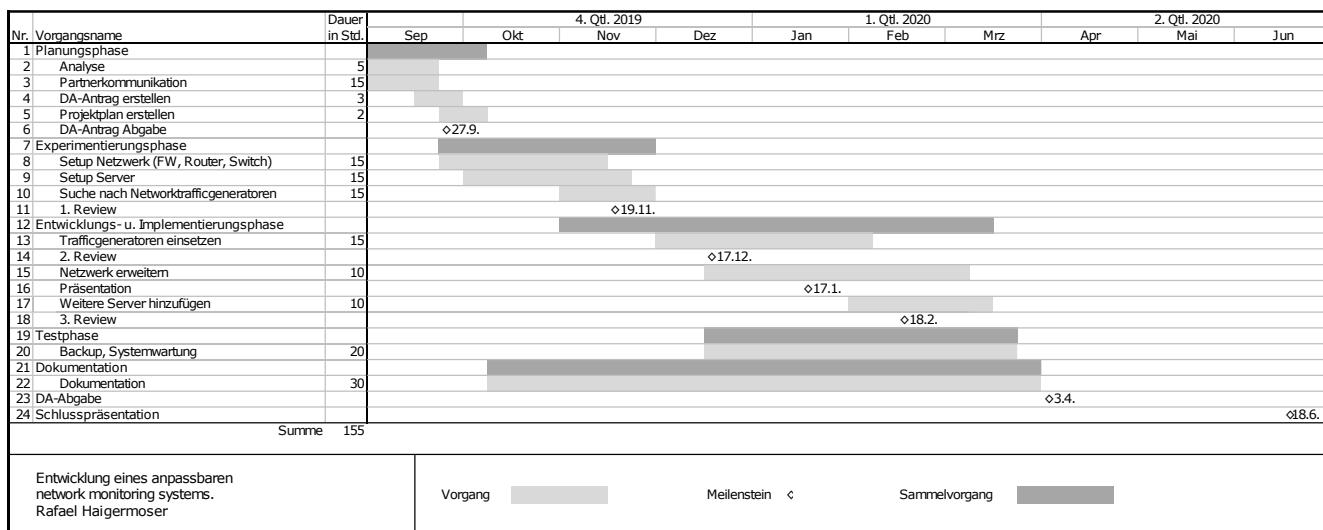


Abbildung 2.3: Zeitplan von Rafael Haigermoser

## 2.2.3 Anna Victoria Krupa

Aufgabenbereiche:

- Graphische Darstellung
  - Entwurf der Oberfläche
  - Erstellung von Standardelementen
  - Umsetzung für das Testnetzwerk
- Alarmierung

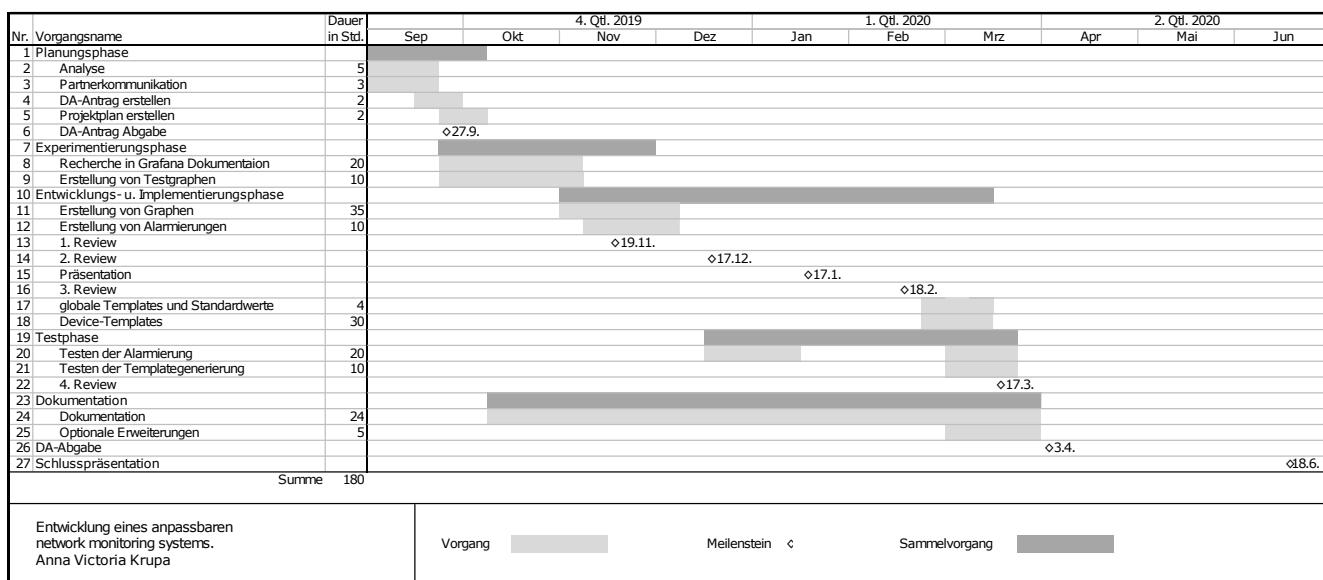


Abbildung 2.4: Zeitplan von Anna Victoria Krupa

## 3 Datenauslese

Um möglichst viele Informationen von beliebigen Geräten einzulesen, unterstützt NetBat eine Reihe von Standardmethoden, sich mit anderen Geräten und Programmen in Verbindung zu setzen.

Hierbei wirkt NetBat immer aktiv, also als Initiator der Verbindung, und fragt die verlangten Daten regelmäßig ab. Dieses Vorgehen bietet dem Monitoring-System mehr Kontrolle über die zeitliche Abstimmung und ist zudem universeller als ein passiver Zugang, da viele Geräte und Programme nur einlesbare Schnittstellen nach außen zur Verfügung stellen und selbst keine Kontrolldaten versenden. So würde etwa das Nutzen von Ping Nachrichten über ICMP zusätzliche Software benötigen und Datenverkehr für die Übermittlung des Ergebnisses erzeugen. Außerdem wäre es unmöglich, Informationen von Munin Nodes zu erhalten.

### 3.1 Verwendete Netzwerkprotokolle

Um Betriebsdaten von anderen Geräten in einem Internet Protocol (IP) Netzwerk zu erhalten, bedient sich NetBat mehrerer standardisierter Protokolle. Die hier ausgewählten 4 Protokolle eignen sich jeweils für bestimmte Einsatzgebiete oder Informationen. Auf diese Weise kann von möglichst vielen Geräten Information erhalten werden. Es werden sowohl Internet Protocol Version 4 (IPv4) als auch Internet Protocol Version 6 (IPv6) unterstützt.

#### 3.1.1 ICMP

Um die Erreichbarkeit der einzelnen Geräte sowie die Übertragungsdauer zwischen den Geräten zu ermitteln, werden die von der Internet Engineering Task Force (IETF) definierten Echo Messages des Internet Control Message Protocols (ICMP) [1] benutzt.

Diese als Ping [2] bezeichnete Abfrage ist keine Datenabfrage, sondern eine Ermittlung von Daten, wird allerdings im weiteren Verlauf von NetBat als Datenabfrage verwendet.

### 3.1.2 SNMP

Für die Kommunikation mit Netzwerkinfrastruktur, wie etwa Switches, Router, Wireless Access Points und Firewalls, eignet sich das verbreitete Simple Network Management Protocol (SNMP) [3] und dessen neuere Versionen. Es ist der Standard für den Zugriff auf die in Geräten der Netzwerkinfrastruktur gespeicherten Daten und somit bei allen Betriebssystemen von allen Herstellern anwendbar.

Wie noch später in Unterkapitel 5.5.2 genauer erläutert, wird von NetBat aufgrund der Kompatibilität der verwendeten Software nur das Community-Based Simple Network Management Protocol Version 2 (SNMPv2c), definiert in RFC 1901 [4], unterstützt.

SNMP ist in Object Identifier (OIDs) organisiert. Diese OIDs sind in einer Baumstruktur angeordnet, wobei die Identifikation innerhalb einer Ebene durch ein Byte erfolgt. [3] Eine OID ist folglich eine Kette aus Bytes, von denen jedes jeweils eine Ebene symbolisiert. Geschrieben werden OIDs, indem die Bytes im Dezimalsystem angegeben und durch Punkte getrennt werden. [3]

Als Sicherheitsmaßnahme werden bei SNMPv2c Zugriffsrechte für sogenannte „Communitys“ vergeben. Die verwendete Community muss bei Anfragen im SNMP-Header angegeben werden. Hierbei handelt es sich um einen unverschlüsselten Text. [3]

### 3.1.3 HTTP/HTTPS

Eine weitere Möglichkeit des Datentransfers bieten die Protokolle Hypertext Transfer Protocol (HTTP) und Hypertext Transfer Protocol Secure (HTTPS). Diese TCP Protokolle erlauben es Programmen, Information so zur Verfügung zu stellen, dass von außen auf diese zugegriffen werden kann. Solche Schnittstellen werden häufig verwendet, um die Entwicklung von Software zu erleichtern, da kein eigenes Protokoll entwickelt und implementiert werden muss. Außerdem wird diese Herangehensweise aufgrund der Verbreitung der Protokolle oft bei öffentlich zugänglichen oder möglichst universellen Application Programming Interfaces (APIs) verwendet. Gute Beispiele hierfür wären die Overpass API, welche den Zugriff auf die Daten von OpenStreetMaps ermöglicht [5] und hierfür ein HTTP Interface zur Verfügung stellt sowie das in Kapitel 6 beschriebene Grafana, welches das Einsehen und Editieren der Konfiguration über HTTP erlaubt. [6]

### 3.1.4 Munin

Ein weiterer wichtiger Aspekt der Datenauslese ist das Überwachen des Betriebsstatus von Hostcomputern mit Betriebssystemen, wie etwa Windows Server, OpenBSD oder einer Linux-Distribution.

Diese Überwachung lässt sich leicht mit dem Werkzeug Munin erreichen. Bei Munin handelt es sich um ein „networked resource monitoring tool“, welches in einer Master-Node Struktur arbeitet. Hierbei ruft der Master die Daten der Nodes ab, um sie daraufhin weiterzuverarbeiten. [7] Um dieses System für die Auslese von Informationen von Hostcomputern zu nutzen, kann NetBat als Munin-Master agieren und sich mit den Nodes über das von Munin verwendete Munin Protocol [7] verbinden.

Die Daten werden von Munin in „Plugins“ gruppiert. Diese enthalten mehrere „Fields“. Ein Field besitzt einen Wert und weitere Attribute [7], die für NetBat nicht von Bedeutung sind.

## 3.2 Besondere Datenquellen

Zusätzlich zu der Verbindung über Netzwerk-Protokolle können Daten auch auf anderem Wege gewonnen werden. Die folgenden Datenquellen haben keine direkte Verknüpfung zu einem anderen Gerät und werden daher für das eigene Gerät, indirekte Informationsabfrage oder spezielle Zwecke verwendet.

### 3.2.1 NetBat Logo

NetBat besitzt in seiner Standardausführung die Möglichkeit, bestimmte vordefinierte Datenreihen in einer Schleife zu erzeugen und als gemessene Daten weiterzuverarbeiten. Diese speziellen, in Tabelle 3.1 aufgelisteten Werte bilden grafisch übereinander dargestellt das NetBat Logo und dienen lediglich dem Zweck der Unterhaltung. Eine solche Darstellung ist in Abbildung 6.3 zu sehen.

Eine zusätzliche Überlegung wäre, diese Datenreihen zur Demonstration und für Testvorgänge der Implementierung der grafischen Anzeige zu verwenden.

### 3.2.2 Befehlsausgabe

Eine sehr allgemeine Art der Informationsbeschaffung ist das Auslesen der Ausgabe eines Kommandozeilenbefehls. Auf diese Weise können lokale Daten eingelesen, spezielle Programme ohne zusätzliche Implementierung genutzt und lokal installierte Programme leicht verbunden werden. Es ist somit von Vorteil, beliebige Befehle auszuführen und die erhaltene Ausgabe als Datenquelle zur Verfügung stellen zu können.

Datensatz 1	Datensatz 2	Datensatz 3
19	3	18
19	3	18
17	8	15
14	10	16
15	11	14
14	12	14
14	10	16
15	12	13
13	10	17
16	9	15
11	22	7
10	20	10
6	23	11
10	20	10
11	22	7
16	9	15
13	10	17
15	12	13
14	10	16
14	12	14
15	11	14
14	10	16
17	8	15
19	3	18
19	3	18

Tabelle 3.1: NetBat Logo Datensätze

## 4 Das Datenbanksystem

Die Daten eines Monitoring Systems abzuspeichern, ist eine große Herausforderung. Es werden laufend viele neue Daten eingelesen. Diese sollen dann in beliebigen Gruppen wieder ausgegeben werden. Weiters benötigen diese Daten ein hohes Speichervolumen. Für eine einzige 32 Bit Zahl, welche von einem Gerät einmal alle 5 Sekunden ausgelesen wird, werden nach 30 Tagen bereits 2,0736 Megabyte benötigt. Liest man nun 100 solcher Werte pro Gerät jede Sekunde aus einem Netzwerk mit 100 Geräten, benötigt man nach einem Monat für die Daten selbst schon über 100 Gigabyte.

Um diese Ausgabe zu bewältigen, ist die Verwendung eines Datenbanksystems ratsam. Hierbei sind Ablagestruktur und Geschwindigkeitsoptimierungen beim Hinzufügen, Verändern, Löschen und Abfragen der Daten bereits gelöst und verwendbar.

### 4.1 Wahl von InfluxDB

Eine entscheidende Frage im Zusammenhang mit der Verwendung eines Datenbanksystems ist es, welche Charakteristiken dieses haben soll. Da es bereits eine Menge Datenbanksysteme gibt, muss eine Auswahl getroffen werden.

Eine große Gruppe bilden jene relationalen Datenbanken, mit denen über die Structured Query Language (SQL), genormt durch den Standard ISO/IEC 9075 [8] der International Organization for Standardization (ISO), interagiert wird. Diese Datenbanksysteme bieten jedoch deutlich mehr Funktionen, als für dieses Projekt benötigt werden. So werden etwa Verweise zwischen Tabellen und spezielle Sicherheitsmechaniken nicht benötigt. Aus diesem Grund sind diese Datenbanken für den Anwendungsfall nicht optimal, da es ohne diese Funktionen möglich wäre, ein kleineres und somit schnelleres System zu nutzen.

Eine Alternative hierzu bietet das Datenbanksystem namens InfluxDB. Dieses folgt einem Konzept, welches für sogenannte „time-series data“ [9] entworfen ist. Dabei handelt es sich um Reihen gleichförmiger Daten, welche mit Zeitstempeln identifiziert werden. Dieses System bietet wenig Möglichkeiten, komplexe Strukturen aufzubauen, ist jedoch für eine hohe und laufend wachsende Anzahl ähnlicher Datensätze, wie sie bei einem Monitoring System erzeugt werden, optimiert. [9]

Aufgrund der expliziten Auslegung und Optimierung für den vorhandenen Anwendungsfall wurde InfluxDB als die Standarddatenbank für NetBat ausgewählt.

## 4.2 Datenbankschema

Das Schema, in dem die Daten in InfluxDB abgelegt werden, besteht im Wesentlichen aus Measurements, welche Einträge, bestehend aus Fields und Tags, enthalten. Measurements können als den Tabellen der SQL ähnlich gesehen werden. Sie enthalten eine Reihe an Datensätzen, welche einen bestimmten Aufbau aufweisen. Jeder Datensatz besitzt einen Zeitstempel als Primärschlüssel, der den Zeitpunkt angibt, an dem er in die Datenbank eingefügt wurde. Alle weiteren Informationen sind in Fields und Tags aufgeteilt. Sowohl Fields als auch Tags dienen der Ablage von Daten, besitzen allerdings unterschiedliche Eigenschaften und dienen einem anderen Zweck. [9]

Fields besitzen einen fixen Datentyp und werden nicht indexiert. Damit eignen sie sich für die Ablage der erhaltenen Daten, solange diese nur wieder ausgelesen und nicht für Filter genutzt werden. [9]

Tags hingegen können ohne Festlegung des Datentyps geschrieben werden und werden zusätzlich indexiert. [9] Sie können somit zur Gruppierung und Identifikation der einzelnen Datensätze verwendet werden, beispielsweise zur Angabe des Geräts, von dem die Daten stammen.

## 4.3 Downsampling

Eine wichtige Funktion, welche für die sinnvolle Verwendung eines Monitoring Systems benötigt wird, ist die Möglichkeit, alte Daten zusammenzufassen und so Speicherplatz zu sparen. Dies ist essentiell, da sich, wie bereits zu Beginn des Kapitels erläutert wird, mit der Zeit eine sehr große Menge an Daten ansammelt und man, um den steigenden Speicherbedarf zu decken, entweder laufend neue Datenträger hinzufügen oder alles ab einem gewissen Alter löschen müsste. Beide Alternativen sind nicht wünschenswert. Ersteres wäre mit hohen Kosten und erhöhtem Aufwand verbunden. Letzteres würde hingegen den Verlust von Informationen bedeuten, welche man, zum Beispiel für Vergleiche, zu einem späteren Zeitpunkt noch gebrauchen könnte.

Das sogenannte Downsampling behebt dieses Problem, indem alte Daten, welche nicht mehr in maximaler Präzision benötigt werden, zusammengefasst werden. Dies sorgt für erhebliche Speicherersparnisse, sodass grobe Informationen über den früheren Zustand über Jahrzehnte aufbewahrt werden können.

Diese Zusammenfassung kann durch verschiedene Operationen durchgeführt werden. Während CPU-Auslastung, genutzte Bandbreite oder Temperaturen eher arithmetisch gemittelt werden, können etwa boolesche Wahrheitswerte so kombiniert werden, dass ein Wert dominant ist und bei einmaligem Vorkommen innerhalb des Zeitintervalls das zusammengefasste Ergebnis bestimmt. Auch das Addieren von Zahlenwerten kann, etwa bei einer Zählung, eine gewollte Funktion sein.

Nach mehreren Versuchen stellt sich heraus, dass die Implementierung des Downsamplings eine komplexe und umständliche Aufgabe ist, bei der auf viele Dinge geachtet werden muss. Im Speziellen wurden im Rahmen des Projekts 3 Ansätze versucht, bis auf den dritten haben sich jedoch alle als unmöglich oder ungeeignet herausgestellt.

#### 4.3.1 Ansatz 1

Die erste Herangehensweise an das Downsampling besteht im Wesentlichen aus der Nutzung des in InfluxDB bereits integrierten Systems.

InfluxDB bietet zwei eigenständige Funktionen an, welche zusammen für Downampling genutzt werden können. Diese beiden heißen Continuous Query (CQ) und Retention Policy (RP) und werden in der Dokumentation von InfluxDB folgendermaßen beschrieben:

„A continuous query (CQ) is an InfluxQL query that runs automatically and periodically within a database. CQs require a function in the SELECT clause and must include a GROUP BY time() clause.

A retention policy (RP) is the part of InfluxDB data structure that describes for how long InfluxDB keeps data. InfluxDB compares your local server's timestamp to the timestamps on your data and deletes data that are older than the RP's DURATION. A single database can have several RPs and RPs are unique per database.“ [9]

Bei diesem bereits vorhandenen System werden beim Schreiben neuer Daten automatisch die Kombinationsfunktionen der Continuous Queries (CQs) angewandt und die Ergebnisse regelmäßig in die jeweilige Retention Policy (RP) gespeichert. Das heißt, dass die Daten stets mehrfach, in unterschiedlicher Auflösung, in den jeweiligen RPs liegen, die hochauflösenderen reichen hierbei nur eine bestimmte, kürzere Zeitspanne in die Vergangenheit. Alte Bereiche werden in großen Gruppen gelöscht, welche getrennt von einander auf der Festplatte gespeichert sind. [9]

Dieses System hat den Vorteil, sehr performant zu sein. Es werden keine Suchen in vergangenen Zeitabschnitten durchgeführt und keine Werte nachträglich abgeändert und der Löschvorgang entspricht lediglich dem Entfernen einer Datei, wodurch ebenfalls keine Suchen und Änderungen nötig sind.

Der entscheidende Nachteil dieser Funktion ist jedoch, dass die feinen und groben Daten in voneinander getrennten RPs liegen. Dadurch ist das Abfragen der Daten eines größeren, mehrere solche Genauigkeitsbereiche beinhaltenden Zeitabschnitts äußerst schwierig, da InfluxDB zum gegebenen Zeitpunkt keine Möglichkeit anbietet, die Informationen automatisch aus der hochauflösendsten RP zu verwenden. Eine solche Abfrage wird mit dem in Kapitel 6 beschriebenen System nicht unterstützt. Da es somit einen erheblichen Aufwand bedeuten würde, die für Kompatibilität nötigen Anpassungen vorzunehmen, ist Ansatz 1 nicht zum Einsatz in NetBat geeignet.

### 4.3.2 Ansatz 2

Die zweite Idee zur Implementierung des Downsamplings ist es, dem NetBat Server das regelmäßige Zusammenfügen der Datensätze sowie das Löschen der alten Information zu überlassen. Hierbei sendet der Server regelmäßig bestimmte Querys an die Datenbank.

Ein Downampling-Durchlauf kann in die folgenden Schritte unterteilt werden: Zuerst werden die feinen Daten mit einem `SELECT INTO` Befehl ausgewählt, in Gruppen zusammengefasst und mit der gewünschten Funktion akkumuliert. Diese neuen Datensätze werden in eine leere RP zwischengespeichert, um beim Löschvorgang nicht mit ausgewählt zu werden. Dann werden die alten Daten in dem ermittelten Zeitintervall auf der default RP mit `DELETE` gelöscht. Anschließend werden die zusammengefassten Datensätze mit `SELECT INTO` wieder in die default RP zurückgeschrieben. Abschließend wird die temporäre RP noch für den nächsten Durchlauf geleert.

Die Auswahl der richtigen Zeitintervalle ist hier von besonderer Bedeutung. Da sich der Gruppierungsraster zweier aufeinanderfolgender Durchläufe bei Unregelmäßigkeiten der Zeitabstände zwischen diesen sonst verschieben könnte, müssen sich der Beginn und das Ende des Kombinationsbefehls stets an diskreten, vom Ausführungszeitpunkt unabhängigen Punkten befinden. Dies gelingt, indem man die momentane Uhrzeit in Form eines Unix Timestamps auf das nächste Vielfache der Gruppenweite rundet. Dies sorgt im Weiteren dafür, dass die Anwendung der Berechnung auf bereits zusammengefasste Zeitintervalle keinerlei Auswirkung hat, da stets maximal ein Wert gefunden wird, erhalten bleibt und an der selben Stelle wieder in die Datenbank zurückgeschrieben wird. Um den Rechenaufwand des Servers zu verbessern, kann das bereits berechnete Intervall zwischengespeichert werden, sodass beim darauffolgenden Durchlauf nur mehr der neue Abschnitt kopiert, kombiniert und zurückgeschoben werden muss.

Um diese Durchläufe nicht für jedes Measurement einzeln durchzuführen, erlaubt es InfluxDB, alle Measurements mit der Form `./.*` auf einmal auszuwählen und die Ergebnisse durch die Angabe des Zielmeasurements `:MEASUREMENT` wieder in dieselben abzuspeichern. [9] Dies ist eine weitere Möglichkeit, die Geschwindigkeit des Prozesses zu verbessern.

Eine Hürde stellt hierbei das Zurückschreiben in die ursprüngliche RP dar, da die Akkumulation den Datentyp und den Fieldnamen ändern kann. Dies ist etwa bei `mean()` der Fall, hierbei ist das Ergebnis nämlich stets vom Typ Float, egal ob die ursprünglichen Daten vom Typ Float oder Integer waren. [9] Dies macht beim Zurückschreiben Probleme, da die Werte der Fields in einem Measurement, wie oben erwähnt, immer den gleichen Datentyp haben müssen. Dies kann gelöst werden, indem ein Typecast zum Kombinationsbefehl hinzugefügt wird. Eine Voraussetzung dafür ist allerdings, dass die Namen und Typen der Fields beim Absenden der Query bekannt sind. Um das Problem zu lösen, muss der Server diese Informationen also im Speicher behalten. Zusammen mit der im letzten Absatz erwähnten Kombination aller Measurements in eine Query ergibt das zusätzlich eine Einschränkung der genutzten Schemata. Werden die Fields mit Typkonvertierungen versehen, müssen Fields mit gleichem Namen in allen Measurements den gleichen Typ besitzen. Diese Einschränkung bei der Benutzung von InfluxDB als Datenspeicher wird in NetBat akzeptiert.

Das Problem, an dem diese als Ansatz 2 beschriebene Herangehensweise letztlich scheitert, ist die Tatsache, dass das Löschen von Datensätzen nur in allen RPs gleichzeitig, nicht allerdings in einer einzelnen erfolgen kann. Dadurch wird die oben genannte Löschung der alten Daten unmöglich, da die zusammengefassten Datensätze in der temporären RP ebenfalls gelöscht werden würden. Dieses Hindernis wurde erst im Laufe der Tests entdeckt, weil es in der Dokumentation von InfluxDB bislang keinen expliziten Hinweis darauf gibt.

#### 4.3.3 Ansatz 3

Der dritte und letztlich gewählte Ansatz basiert zum größten Teil auf dem zweiten. Die Aktionen werden vom NetBat Server über Querys durchgeführt und auf den oben genannten Wegen performanter gestaltet.

Der entscheidende Unterschied der beiden Optionen ist, dass das Problem, welches zuvor beim Löschen aus einzelnen RPs aufgetreten ist, durch das Verwenden unterschiedlicher Datenbanken umgangen wird. Statt die temporären Datensätze in eine andere RP zu kopieren, werden sie in eine leere Datenbank kopiert, welche den selben Aufbau besitzt wie die ursprüngliche.

Aus dieser Überlegung ergeben sich folgende Queries für die bei Ansatz 2 aufgezählten Schritte:

---

```
1 SELECT "field1"::integer, "field2"::boolean INTO {DBName}_tmp...:MEASUREMENT FROM
    (SELECT round(mean("field1")) AS "field1", mode("field2") AS "field2" FROM
     /* WHERE time < {lastRun}s AND time >= {upperBorder}s GROUP BY time({step}
     s),*) GROUP BY *;
2 DELETE FROM /* WHERE time < {lastRun}s AND time >= {upperBorder}s;
3 SELECT * INTO {DBName}...:MEASUREMENT FROM {DBName}_tmp.../* WHERE time < {
    lastRun}s AND time >= {upperBorder}s GROUP BY *;
4 DELETE FROM /* WHERE time < {lastRun}s AND time >= {upperBorder}s;
```

---

Diese Queries enthalten beispielhaft die Fields field1 und field2 mit den Datentypen Integer bzw. Boolean. Die mit geschwungenen Klammern markierten Bereiche stellen Platzhalter dar. {DBName} steht hier für den Namen der Datenbank, {lastRun} und {upperBorder} für die Grenzen des zusammenzufassenden Zeitintervalls in Sekunden und {step} für den neuen Abstand zwischen den Datensätzen in Sekunden.

## 5 NetBat-Server

Das zentrale Element des Netzwerk Monitoring Systems ist der NetBat-Server, welcher sich um die Steuerung, den Datenaustausch und die Datenverarbeitung kümmert. Er bildet das einzige aktive Element des Projekts, ist also das einzige, welches von sich aus und ohne äußere Einflüsse Aktionen durchführt. Alle anderen Teile werden stets vom ihm oder vom Anwender selbst aktiviert.

### 5.1 Programmiersprache und Runtime

Für die Implementierung des Servers ist die Programmiersprache JavaScript, standardisiert unter dem Namen ECMAScript®, mit dem Runtime Node.js® gewählt worden. Diese Wahl hat mehrere Gründe.

Node.js® baut auf Googles V8 JavaScript-Engine auf. Dadurch erübriggt sich der Vorgang des Kompilierens für den Entwickler, da der Vorgang hier zur Laufzeit durch einen Just-in-time-Compiler (JIT-Compiler) durchgeführt wird. [10] Der Aktualisierungsvorgang des Testservers während der Entwicklung wird erleichtert, da es genügt, den neuen Programmcode herunterzuladen und keine verwechselbaren Binärdateien erzeugt und ausgetauscht werden müssen. Außerdem erlaubt es, etwa zu Testzwecken, temporäre Änderungen am Testserver vorzunehmen und anschließend wieder zu löschen.

Weiters ermöglicht es Node.js®, sehr einfach Interaktionen mit anderen Netzwerkgeräten zu implementieren, da es für Netzwerkapplikationen entworfen ist und somit gut mit vielen gleichzeitigen Verbindungen umgehen kann. Durch seinen asynchronen und eventbasierten Aufbau erlaubt es, das Programm in einem Thread zu betreiben. [11] Dass auf diese Weise nicht auf das Blockieren von Variablen und anderen gespeicherten Informationen geachtet werden muss, erleichtert die Implementierung des Servers.

Ein weiterer Grund für die Wahl dieser Programmiersprache ist, dass es mit dem Tool namens „npm“ sehr leicht ist, bereits vorhandene JavaScript-Pakete anderer Programmierer zu nutzen. [12] Die von npm, Inc. verwaltete Datenbank, welche zum Zeitpunkt des Schreibens 1567649 Pakete enthalten hat, [13] bietet Implementierungen zu vielen Problemstellungen, von denen einige im Rahmen dieses Projekts genutzt werden.

Persönliche Präferenzen und bereits gesammelte Erfahrung sind ebenfalls ein Grund für die Wahl von JavaScript als Programmiersprache gewesen.

## 5.2 Aufgaben des Servers

Der NetBat Server ist für die folgenden Aufgaben innerhalb des NetBat Monitoring Systems zuständig:

- Auslese der Daten
- Verarbeitung der Daten
- Übergabe der Daten an die Datenbank
- Umsetzung des Downsamplings (vgl. Unterkapitel 4.3.3)
- Automatische Konfiguration der grafischen Oberfläche

„Verarbeitung der Daten“ meint hier die Anwendung mathematischer Funktionen, die Konvertierung in ein anderes Format oder eine andere Übersetzung, wie etwa das Beschränken auf einen Wertebereich.

Die automatische Konfiguration des für die grafische Darstellung verwendeten Grafana Servers (siehe Kapitel 6) ist hier als Zusatz zu etwaigen situationsspezifischen Einstellungen gedacht. Vom Anwender angelegte Dashboards werden von diesem System nicht beeinflusst.

## 5.3 Aufbau

Der Programmcode des Servers ist in das Hauptscript und 3 Ordner unterteilt. Ein vierter Ordner, `node_modules`, wird für die lokalen Kopien der npm-Pakete verwendet. Diese 5 Orte sind in Abbildung 5.1 durch Fettdruck markiert.

Das Hauptscript, die Datei `index.js`, kümmert sich um übergebene Kommandozeilenparameter, das Parsen der Konfigurationsdateien und den Ladevorgang. Dies geschieht in folgenden Schritten:

1. Erstellung globaler Variablen
2. Einlesen und Parsen des Environemnts aus der `env.json` und der `env.default.json` Datei
3. Verarbeitung der Kommandozeilenparameter
4. Einlesen und Parsen der Konfigurationsdatei
5. Aufbau der Datenbankverbindung
6. Laden von Templates und Devices
7. Starten der Deviceauslese
8. Starten des Downsamplings
9. Ausgabe der Abschlussnachricht „There will be dragons“



Abbildung 5.1: Die Ordnerstruktur des NetBat Servers. Fettgedruckte Ordner und Dateien enthalten Programmcode, kursiv gesetzte sind lokal.

Der Ordner `lib` enthält die Kernbibliotheken von NetBat. Die `Order database` und `interfaces` enthalten die Implementationen der Datenbankinterfaces sowie die Datenquellenmodule. Die hier enthaltenen Codedateien werden im Allgemeinen nicht für NetBat benötigt, sondern erweitern die Kompatibilität um zusätzliche Datenbanken, Protokolle und andere Datenquellen. Da die Konfiguration immer eine unterstützte Datenbank spezifizieren muss, ist hierzu zu sagen, dass trotzdem mindestens ein Datenbankinterface für das erfolgreiche Starten des Servers benötigt wird. Der `node_modules` Ordner, welcher die npm-Pakete enthält, wird nicht von den Entwicklern, sondern von npm verwaltet. Um ihn zu erstellen oder nach einer Änderung der benötigten npm-Pakete zu aktualisieren, muss der Befehl

```
$ npm ci
```

in der Kommandozeile ausgeführt werden. Hierdurch werden die npm-Pakete mit den vom Entwickler verwendeten Versionen, die in der Datei `package.lock.json` abgespeichert sind, heruntergeladen und installiert. [12]

## 5.4 Kernbibliotheken

Die in NetBat als „Kernbibliotheken“ bezeichneten Codedateien enthalten den Programmkerne, also die JavaScript-Dateien, welche nicht Teil eines optionalen Moduls sind und für den Start von NetBat in jedem Fall benötigt werden.

### 5.4.1 config

Die `config` Bibliothek enthält Funktionen für die Auffindung, Einlese und Verarbeitung von Templates und Devices. Die Implementierung der Vererbung von Templates und Devices befindet sich hier. `config` enthält die folgenden Funktionen:

**`addTemplate()`** fügt ein geladenes Template zum Cache hinzu. Hierbei wird mit der Funktion `assertTemplateKnown()` sichergestellt, dass alle angegebenen Basistemplates vorhanden sind.

**`addTemplates()`** ist eine Variante der `addTemplate()` Funktion, welche ein Array aus Templates erhält.

**`assertTemplateKnown()`** wirft einen `AssertionError` mit der Nachricht „Template not found“, wenn kein Template mit dem angegebenen Namen von `findTemplates()` gefunden wurde.

**assignInheritance()** nutzt die Funktion `deepAssignOneValue()` aus der `jstools` Bibliothek, um Templates und Devices mit ihren Basistemplates zu kombinieren. Eine besondere Vorgehensweise gibt es hier bei dem Key „grafana“. Hierbei wird die Funktion `assignInheritedGrafana()` verwendet. Diese Funktion wird nicht exportiert.

**assignInheritedGrafana()** kombiniert die Grafanakonfigurationen von Templates und Devices mit denen ihrer Basistemplates. Hierbei werden die Arrays „dashboards“ und „folders“ an die jeweiligen Arrays aus den Basistemplates angehängt und das Object „panels“ mit der Funktion `Object.assign()` so zusammengefügt, dass die Werte der Basistemplates - dies bezieht nicht vorhandene Werte mit ein - überschrieben werden, wenn solche in den erbenden Konfigurationen vorhanden sind.

**createDevice()** erhält eine vom Filesystem stammende Device-Konfiguration, vervollständigt diese mittels `evaluateInheritance()`, erzeugt eine neue Device-Instanz und gibt die resultierenden Datenbankschemata an das übergebene Datenbankinterface weiter. Die erzeugte Device-Instanz wird zurückgegeben.

**createDevices()** ist eine Variante der `createDevice()` Funktion, welche ein Array aus Device-Konfigurationen erhält und ein Array aus Device-Instanzen zurückgibt.

**evaluateInheritance()** ermittelt die Basistemplates eines Templates oder Devices, falls vorhanden. Diese werden im Anschluss mit `loadTemplate()` geladen. Die Pfade hierfür werden dem lokalen Object entnommen, welches von `findTemplates()` generiert wird. Als nächstes werden die Basistemplates mit `assignInheritance()` mit der übergebenen Konfiguration kombiniert. Die so erhaltene Konfiguration wird zurückgegeben. Diese Funktion wird nicht exportiert.

**findDevices()** sucht in einem angegebenen Ordner nach Devices und gibt deren Pfade als Array zurück.

**findTemplates()** sucht in einem angegebenen Ordner nach Templates und speichert die Information über deren Pfade in ein lokales Object.

**isTemplateKnown()** gibt `true` zurück, wenn kein Template mit dem angegebenen Namen von `findTemplates()` gefunden wurde, ansonsten `false`.

**loadTemplate()** lädt ein Template mit angegebenem Namen und Pfad vom Filesystem und fügt dieses anschließend mit `addTemplate()` zum Cache hinzu. Die Dateiformate JavaScript Object Notation (JSON) und YAML Ain't Markup Language (YAML) werden unterstützt.

**readDeviceFile()** liest eine Device-Konfigurationsdatei ein und gibt das resultierende Object oder Array zurück. Die Dateiformate JSON und YAML werden unterstützt.

### 5.4.2 db

Die db Bibliothek lädt die vorhandenen Implementierungen für Datenbankinterfaces aus dem Ordner database. Die hier enthaltene asynchrone Funktion **init()** wird benutzt, um im Hauptscript sicherzustellen, dass dieser asynchrone Filesystemzugriff angeschlossen ist, bevor die Datenbankverbindung aufgebaut wird. Die folgenden weiteren Funktionen und Variablen werden exportiert.

**FieldType** ist ein mit `Object.freeze()` eingefrorenes, also unveränderbares Object, welches die von der Datenbankabstraktion verwendeten Datentypen enthält. Hierbei werden den Datentypennamen, welche den Keys entsprechen, konstante Zahlen zugewiesen. Dies ähnelt der Funktionsweise von Enumerationen aus anderen Programmiersprachen.

**getFieldType()** wandelt Strings, welche einen der Datentypen bezeichnen, in die jeweilige Konstante um. Als Standardwert wird `null` zurückgegeben.

**createNewDB()** erzeugt eine neue Instanz des Datenbankinterfaces mit dem übergebenen Namen. Wird keine entsprechende Implementierung gefunden, wird ein Error mit der Nachricht „Unknown database system“ geworfen.

### 5.4.3 device

Die device Bibliothek lädt die vorhandenen Implementierungen für Datenbankinterfaces aus dem Ordner interfaces. Die hier enthaltene asynchrone Funktion **init()** wird benutzt, um im Hauptscript sicherzustellen, dass dieser asynchrone Filesystemzugriff angeschlossen ist, bevor die Datenbankverbindung aufgebaut wird. Die Bibliothek exportiert die folgenden Funktionen und Variablen.

**VarType** ist ein mit `Object.freeze()` eingefrorenes, also unveränderliches Object, welches die Variablentypen beinhaltet, die von der weiter unten beschriebenen Funktionsgenerierung verwendet werden. Hierbei werden den Variablentypennamen, welche den Keys entsprechen, konstante Zahlen zugewiesen. Dies ähnelt der Funktionsweise von Enumerationen aus anderen Programmiersprachen. Die hier festgelegten Variablentypen heißen `collected`, `stored`, `index`, `size`, `top`, `unwrapped` und `builtin`.

Die Bibliothek exportiert die Funktionen `varValid()`, `checkVarValid()` und `nameValid()`, welche die Einhaltung formaler Kriterien von Bezeichnern innerhalb der Device-Konfiguration prüfen.

**varValid()** und **checkVarValid()** werden zur Überprüfung von Variablenbezeichnern verwendet. Hierbei erlauben der zweite und dritte Funktionsparameter anzugeben, ob die referenzierende Form mit dem Präfix „\$“, die definierende Form ohne diesen, oder beide Formen als valide gesehen werden. Der Name der Variable, welcher der definierenden Form entspricht, wird hierbei an `nameValid()` übergeben.

`varValid()` und `checkVarValid()` selbst überprüfen nur das Dollarzeichen. Der Unterschied zwischen den beiden Funktionen besteht darin, dass `varValid()` einen booleschen Wert zurückgibt, welcher die Korrektheit des übergebenen Variablenbezeichners darstellt, während `checkVarValid()` einen Error mit der Nachricht „Invalid variable name found“ wirft, wenn den Kriterien nicht entsprochen wird.

`nameValid()` prüft die formalen Kriterien diverser Bezeichner, wie etwa Tabellen-, Field-, Tag- und Variablennamen. Um valide zu sein, darf ein Bezeichner nur aus alphanumerischen Zeichen und Unterstrichen bestehen und muss mit einem Buchstaben oder einem Unterstrich beginnen. Dies ist mit einer sogenannten „regular expression“, auch Regex oder RegExp genannt, realisiert, welche wie folgt aussieht:  
„`^ [A-Za-z_] \w* $`“.

Für den internen Gebrauch enthält die Bibliothek noch weitere Funktionen, welche nicht exportiert werden. Diese heißen `generateUnwrappers()`, `generateUnwrapper()`, `generateValue()` und `generateGrafanaValue()`. Diese vier werden im Konstruktor der Klasse `Device` verwendet und parsen Teile der Device-Konfiguration. Für nähere Information bezüglich des Aufbaus dieser Konfiguration und der hier verwendeten Bezeichnungen siehe Unterkapitel 5.8.

`generateUnwrappers()` verarbeitet die Unwrapper einer Datenbanktabelle. Die Schleifen werden basierend auf Beziehungen untereinander in die richtige Reihenfolge gebracht. Anschließend wird deren Programmcode mit `generateUnwrapper()` erzeugt und gesammelt zurückgegeben. Hierbei wird ein Error mit der Nachricht „Could not order array unwrappers. Looks like some indices are creating loops.“ geworfen, wenn die Beziehungen der Unwrapper Schleifen bilden. Ein Beispiel für eine solche Schleife wären zwei Unwrapper, welche jeweils zwei Indizes enthalten und mit dem ersten auf den zweiten des jeweils anderen referenzieren, also folgende Unwrapper-Konfiguration:

---

```
"unwrap": {
    "var1": ["$i", "i"],
    "var2": ["$j", "j"]
}
```

---

`generateUnwrapper()` wird von `generateUnwrappers()` aufgerufen und erzeugt den Programmcode für einen Unwrapper. In dieser Funktion werden alle weiteren Parameter der Unwrapper verarbeitet, das sind `size`, `top`, `exclude`, `include`, `min`, `max` und `step`. Im Laufe des Generierungsprozesses werden die neuen Variablen aus `index`, `top` und `size`, falls vorhanden, mit den gleichnamigen VariablenTypen in die Variablenliste hinzugefügt. Der Eintrag der ursprünglichen Variable wird mit einem neuen vom Typ `unwrapped` überschrieben, welcher den alten Eintrag und die Nummer des Unwrappers (der Index im Array der Unwrapper-Konfiguration) unter den optionalen Keys `base` und `n` enthält. Dies ist für die spätere Wiederherstellung des ursprünglichen Eintrags nötig.

**generateValue()** verarbeitet einen im Datenbankbereich der Device-Konfiguration angegebenen Wert und erzeugt den entsprechenden Programmcode. Der Wert kann eine Konstante, eine Variable oder ein ValueGenerator sein. Die ValueGenerators werden von `generateValue()` durch rekursive Aufrufe aufgelöst. Im Falle fehlerhafter Konfiguration kann ein Error oder ein AssertionError geworfen werden. Der Error „Unknown function“ deutet auf einen ValueGenerator mit einer unbekannten Funktion hin. Der Error „Unknown variable“ wird ausgegeben, wenn eine undefinierte Variable referenziert wird. Die Errors „Invalid value“ bzw. „Invalid value for <name>“ bedeuten, dass der angegebene Wert ungültig ist und nicht verarbeitet werden kann. Der AssertionError „Variable name <name> already in use“ wird ausgegeben, wenn eine neue Variable definiert wird, deren Name bereits an einer anderen Stelle verwendet wird. Der AssertionError „Expented Array at key ”values“, got“ heißt, dass einer Funktion, welche ein Array unter dem Key `values` verlangt, kein solches bekommen hat. Der AssertionError „Expented Array or Object at key ”values“, got“ heißt, dass eine Funktion, welche ein Array oder ein Object unter dem Key `values` verlangt, kein solches bekommen hat. Zusätzlich zu den genannten, in allen ValueGenerator-Funktionen gleichbedeutenden Fehlern können manche ValueGenerator-Funktionen spezielle Fehlernachrichten ausgeben (siehe Unterkapitel 5.8).

**generateGrafanaValue()** ist eine spezielle Variation von `generateValue()`, welche für allgemeine Werte innerhalb des Grafana-Teils der Device-Konfiguration verwendet wird. Hier werden auch Objects und Arrays als Konstanten behandelt. Objects, welche den Key „\$“ besitzen, werden hingegen verarbeitet und mit dem Wert, welcher sich unter dem Key „\$“ befindet, ersetzt. Ist dieser Wert eine Konstante vom Typ String, kann dieser außerdem Platzhalter enthalten, welche durch eine Begrenzung mit dem Zeichen „\$“ gekennzeichnet werden. Um ein einzelnes Dollarzeichen in den String einzufügen, kann „\$\$“ geschrieben werden. Hat der Wert hingegen einen anderen Typ, wird er an `generateValue()` übergeben und dort verarbeitet. Näheres hierzu findet sich in Unterkapitel 5.8.

Die Bibliothek exportiert die Klasse **Device**, deren Instanzen während des Betriebs die Informationen über die einzelnen Geräte beinhalten. Die Klasse hat die folgenden Eigenschaften und Methoden, bei denen zu beachten ist, dass Eigenschaften und Methoden, deren Namen mit „#“ beginnen, privat sind, also nur innerhalb der Klasse verwendet werden können.

**#nextId** ist eine statische private Eigenschaft, welche die nächste zu vergebene Device-ID enthält.

**#id** ist die einzigartige numerische ID des Devices.

**#timer** ist eine private Timeout Instanz, welche beim Starten der regelmäßigen Ausführung der `runCycle()` Methode von `setInterval()` zurückgegeben wird. Diese wird für das Beenden des Timers mit `clearInterval()` benötigt. [14]

**address** gibt die in der Konfiguration angegebenen Adresse des Geräts an. Hierbei kann es sich um eine IPv4 Adresse, eine IPv6 Adresse oder eine Domain handeln.

**hostname** gibt den in der Konfiguration optional angegebene Namen des Geräts an. Dieser kann, muss aber nicht der Hostname sein.

**devicegroup** gibt die in der Konfiguration optional angegebene Gerätgruppe des Geräts an.

**memory** ist ein Object und dient als Zwischenspeicher für die in der Konfiguration unter „memorize“ definierten Variablen, welche zwischen den Abfragezyklen gespeichert werden. Die Keys sind die Keys in der Konfiguration, die Values sind die Werte.

**funcMemory** ist ein Object und enthält die von diversen ValueGenerator-Funktionen zwischen den Abfragezyklen gespeicherten Daten. Die Keys werden vom Anwender festgelegt - sofern hier kein Modul Änderungen vornimmt, ist das immer die `generateValue()` Funktion -, die Values sind die Werte.

**samplePeriod** gibt die Periodendauer des Abfragezyklus in ms an.

**disablePing** gibt an, ob die automatische Ping-Abfrage für dieses Gerät deaktiviert ist.

**generatedGrafanaSuccessful** gibt an, ob die Grafana-Dashboard-Information dieses Geräts bereits erfolgreich erzeugt worden ist.

**dbSchema** ist ein Array, welches die für dieses Gerät benötigten Datenbankschemata enthält.

**id** ist ein Getter für die private Variable `#id`, welcher somit wie eine Eigenschaft aufgerufen werden kann. Das heißt, der Ausdruck `device.id` gibt, wenn `device` ein Device ist, die ID des Devices zurück. Somit wirkt `id` nach außen wie eine Eigenschaft, die nur gelesen werden kann, da kein Setter existiert.

**runCycle()** ist eine asynchrone Funktion, welche einen Abfragezyklus durchführt. Dies geschieht in folgenden Schritten: Zuerst werden die neuen Daten mit `retrieveData()` abgerufen. Hierfür werden die von den Modulen benötigten Objekte und Funktionen von den `getRetrievalInterface()` Funktionen der Module erhalten und gesammelt als Parameter übergeben. Dann wird `calcDBData()` genutzt, um die Daten zu verarbeiten und die zu schreibenden Datenbankeinträge zu erhalten. Anschließend wird die Grafana-Dashboard-Information mit `generateGrafanaOnDevice()` aus der Bibliothek `grafana` erzeugt, wenn dies noch nicht erfolgreich geschehen ist, also `generatedGrafanaSuccessful` noch nicht `true` ist, und `generatedGrafanaSuccessful` auf den zurückgegebenen Erfolgsstatus gesetzt. Dieser Schritt wird übersprungen, wenn die Erstellung der Visualisierung mittels Grafana in der Konfigurationsdatei deaktiviert ist. Zu guter Letzt werden die neuen Datensätze in die Datenbank geschrieben, indem die Methode `writeDataPoints()` der übergebenen Datenbank aufgerufen wird und die zuvor ermittelten Einträge übergeben werden.

**startTimer()** startet die regelmäßigen Ausführung der `runCycle()` Methode durch Aufruf der

`setInterval()` Funktion und setzt `#timer`.

`stopTimer()` stoppt die regelmäßigen Ausführung der `runCycle()` Methode durch Aufruf der `clearInterval()` Funktion und setzt `#timer` auf `undefined`.

`destruct()` ist eine noch ungenutzte Methode, welche bei der „Zerstörung“ des Devices, zum Beispiel durch das Beenden des Programms, aufgerufen werden soll. Sie stellt sicher, dass der Timer gestoppt ist und ruft die `destructDevice()` Methoden der Module auf.

`retrieveData()` ruft die Daten des Geräts durch Nutzung der Module ab und gibt diese als Object zurück. Der übergebene Parameter enthält von den Modulen benötigte Objekte und Funktionen, zum Beispiel Sockets, Abfragefunktionen und npm-Pakete. Diese asynchrone Methode ist in der Klasse nicht implementiert, sondern wird vom Konstruktor zur Laufzeit erzeugt.

`calcDBData()` übernimmt die erhaltenen Daten, erzeugt die daraus folgenden Datensätze für die Datenbank und gibt diese als Array zurück. Diese Methode ist in der Klasse nicht implementiert, sondern wird vom Konstruktor zur Laufzeit erzeugt.

`generateGrafana()` erhält die abgerufenen Daten, erzeugt damit Dashboards und fügt diese zu einem übergebenen Array hinzu. Wenn Ordner für diese Dashboards benötigt werden, werden deren Namen und IDs in ein übergebenes Object eingefügt. Hier entsprechen die Keys den IDs und die Values den Namen. Der Rückgabewert der Methode ist ein Boolean, der den Erfolg des Aufrufs signalisiert. Ein möglicher Grund für die Rückgabe von `false` ist, dass die Dashboards Variablen enthalten, die von abgerufenen Daten abhängen, diese bei der Abfrage allerdings nicht erhalten worden sind. Diese Methode ist in der Klasse nicht implementiert, sondern wird vom Konstruktor zur Laufzeit erzeugt.

Der Konstruktor der Klasse erhält die bereits vollständige, also mit allen Basistemplates erweiterte Device-Konfiguration und verwendet sie, um die Eigenschaften zu befüllen. Weiters werden hier die Methoden `retrieveData()`, `calcDBData()` und `generateGrafana()` erzeugt, indem Programmcode zusammengesetzt und mit `createFn()` aus der `jstools` Bibliothek in Funktionen umgewandelt wird. Der Konstruktorcode besteht aus den folgenden Schritten:

Zuerst erfolgt die Deklaration des Konstruktors mit einem Parameter namens „`obj`“. Dieser Parameter enthält die vollständige Device-Konfiguration.

---

<sup>163</sup> `constructor(obj) {`

---

Zu Beginn der Funktion werden in der Konfiguration übergebene Eigenschaften des Geräts den entsprechenden Objekteigenschaften zugewiesen und die ID des Geräts gesetzt. Diese Eigenschaften sind die Adresse, die Gerätgruppe, der Hostname, die Periodendauer des Abfragezyklus („`samplePeriod`“) sowie der Aktivierungszustand des automatischen Pings.

---

```

165     this.#id = Device.#nextId++;
166     this.address = obj.address;
167     this.devicegroup = obj.devicegroup || 'any';
168     this.hostname = obj.hostname || '';
169     this.samplePeriod = obj.samplePeriod < 0 ? -1 : obj.samplePeriod === 0 ? 1 :
170         obj.samplePeriod || 1000;
171     env.debug && obj.samplePeriod < 0 && console.log('Automatic sampling
172         disabled for device at ' + this.address);
173     this.disablePing = obj.disablePing || false;

```

---

Anzumerken sind hier die Standardwerte für die samplePeriod (1000), die Gerätegruppe ("any") und die Deaktivierung des Pings (false). Außerdem wird die samplePeriod bei einem negativen Wert auf -1 gesetzt und als deaktiviert angesehen. Befindet sich das Programm im Debug-Modus, wird auf diesen durch eine Konsolenausgabe aufmerksam gemacht.

Als nächstes werden temporäre Variablen erzeugt, die für die Erstellung der drei weiter oben genannten erzeugten Funktionen benötigt werden.

---

```

172     /** @type {Object.<string,VarDesc>} */
173     let vars = {};
174     let nextLocals = { inline: 0, funcMemory: 0 }; //use object to allow call-by
175         -reference

```

---

vars enthält die Beziehungen zwischen Variablennamen und deren Beschreibungen (im Code „VarDesc“), also deren Typen (type), Programmcodes (code) und die Werte base und n im Falle eines Unwrappers, wie bei generateUnwrapper() beschriebenen. Diese Beziehungen werden im weiteren Verlauf als „Variablendeskriptoren“ bezeichnet. nextLocals enthält Nummern, die den Namen von temporären lokalen Variablen im generierten Programmcode beigefügt werden, um keine gleichnamigen Variablen zu definieren. Diese Nummern müssen bei jeder Verwendung inkrementiert werden.

Im nächsten Schritt werden die Datenbankschemata des Pings zu dbSchema hinzugefügt. Ist der automatische Ping deaktiviert, wird dieser Schritt übersprungen.

---

```

176     //Add rtt schema if ping is not disabled
177     !this.disablePing && this.dbSchema.push({
178         measurement: 'rtt',
179         fields: {
180             value: db.FieldType.INTEGER,
181             alive: db.FieldType.INTEGER
182         },
183         tags: {
184             hostaddress: db.FieldType.STRING,
185             hostname: db.FieldType.STRING
186         }
187     });

```

---

Anschließend werden die vordefinierten Variablenbeschreiber mit dem Typ `builtin` zu `vars` hinzugefügt. Diese sind `time_ms`, `address`, `hostname` und `devicegroup` und enthalten die Systemzeit zum Zeitpunkt der Verwendung in ms seit der UNIX Epoch (1.1.1970 00:00:00 UTC) [15] sowie die drei gleichnamigen Eigenschaften des Devices. Außerdem werden die beiden Variablenbeschreiber des Pings, `rtt` und `alive`, vom Typ `collected` hinzugefügt, wenn dieser aktiviert ist.

```

176     //Add build-in variables (time_ms, hostname, address, devicegroup)
177     vars.time_ms = { type: VarType.builtin, code: 'Date.now()' };
178     vars.address = { type: VarType.builtin, code: 'this.address' };
179     vars.hostname = { type: VarType.builtin, code: 'this.hostname' };
180     vars.devicegroup = { type: VarType.builtin, code: 'this.devicegroup' };

182     //Add ping variables when active (rtt, alive)
183     if (!this.disablePing) {
184         vars.rtt = { type: VarType.collected, code: 'data._rtt' };
185         vars.alive = { type: VarType.collected, code: 'data._alive' };
186     }

```

Als nächstes werden die Variablenbeschreiber der zwischen den Zyklen gespeicherten Daten gesetzt. Sie erhalten den Typ `stored`. Die angegebenen Variablennamen werden auf ihre Korrektheit überprüft. Sollte hier ein Fehler auftreten, wird die Nachricht „Error thrown while reading memorized variables of device <address>“ in den Error-Stream der Konsole geschrieben.

```

188     //Add memorized variables
189     try {
190         Object.keys(obj.memorize || {}).forEach(newKey => {
191             checkVarValid(newKey, false, true);
192             if (vars[newKey])
193                 throw new Error(`Variable name ${newKey} already in use`);
194             vars[newKey] = { type: VarType.stored, code: `this.memory['${newKey
195                 }']` };
196         });
197     } catch (err) {
198         console.error('Error thrown while reading memorized variables of device
199             ' + this.address.toString() + ':');
200         throw err;
201     }

```

An diesem Punkt beginnt mit dem Code

```
201     let retrieveData = ['let out = {};'];
```

die Zusammensetzung der `retrieveData()` Methode. Ihr Code wird fürs Erste in einem gleichnamigen lokalen Array abgelegt, welches mit einem Element, nämlich der Zeile „`let out = {};`“, initialisiert wird.

In dieses Object namens out werden im weiteren Verlauf der Funktion alle erhaltenen Daten abgelegt. Am Schluss wird es zurückgegeben.

Im Anschluss wird der automatische Ping hinzugefügt, wenn er aktiviert ist.

---

```

202     if (!this.disablePing) {
203         retrieveData.push(`{
204 let ping_result = await retInt.ping.collectRTT('${this.address}');
205 out.${'_rtt'} = ping_result.time;
206 out.${'_alive'} = ping_result.alive ? 1 : 0;
207 env.debug && ping_result.err && console.log('Ping to ${this.address} returned: '
208     + ping_result.err.toString());
209 if (!ping_result.alive) return out;
210 });
211 }
```

---

Nach dem Ping werden alle anderen Abfragen angehängt. Der Programmcode wird hier von der Funktion genRetrieverCode() des jeweiligen Moduls erzeugt. Schlägt dies fehl, wird die Nachricht „Error thrown while reading <modul> input of device <address>“ im Error-Stream ausgegeben.

---

```

211     Object.keys(interfaces).forEach(interf => {
212         if (!obj[interf])
213             return;
214         env.debug && env.environment === 'development' && console.log(interf);
215         try {
216             interfaces[interf].genRetrieverCode(obj[interf], vars, this,
217                 retrieveData);
218         } catch (err) {
219             console.error(`Error thrown while reading ${interf} input of device
220                 ${this.address.toString()}:`);
221             throw err;
222         }
223     });
224 }
```

---

Zum Schluss wird die letzte Zeile an das Array angehängt, von der createFn() in eine Funktion umgewandelt und in die Device-Instanz geschrieben. Mit diesen Zeilen endet die Generierung der retrieveData() Methode.

---

```

222     //End retrieveData function
223     retrieveData.push('return out;');
224     //Evaluate and store finished retrieveData function
225     env.debug && console.log('retrieveData of device ' + this.address.toString()
226         + ':', retrieveData);
227     this.retrieveData = jstools.createFn('retrieveData', retrieveData, ['retInt']
228         , { async: true });
229 }
```

---

---

Als nächstes beginnt die Zusammensetzung der `retrieveData()` Methode mit dem folgenden Code.

```
228     //Add table calculations
229     let calcDBData = `function isExistingValue(x) { return x != undefined && !
    Number.isNaN(x) && Math.abs(x) !== Infinity; }`, 'let out = []';
```

---

Hierfür wird erneut ein lokales Array erzeugt, welches mit zwei Elementen initialisiert wird. Die erste Codezeile definiert eine lokale Funktion namens `isExistingValue()`, welche die „Existenz“ eines Wertes prüft. Als nicht „existent“ werden hier die Werte `undefined`, `null`, `NaN`, `Infinity` und `-Infinity` angesehen. Die zweite Zeile definiert und initialisiert eine Variable `out`, das Array an Datensätzen, welches am Ende der Funktion zurückgegeben wird.

Als nächstes wird der Code für die Daten des automatischen Pings angehängt, wenn dieser aktiviert ist. Die hier angegebenen Object-Literale legen zusammen mit dem Datenbankschema weiter oben den Aufbau der Tabelle bzw. des Measurements „`rtt`“ fest.

```
230     //Add rtt
231     if (!this.disablePing) {
232         calcDBData.push(`out.push({ measurement: 'rtt', tags: { hostaddress:
            this.address, hostname: this.hostname }, fields: { value: data._rtt,
            alive: data._alive } });`);
233         calcDBData.push(`if(!data._alive) return out;`);
234     }
```

---

Danach wird der Code für die Verarbeitung der restlichen Tabellen erzeugt. Hierbei werden unter anderem `generateUnwrappers()` und `generateValue()` verwendet. Anzumerken ist, dass hier jedem Eintrag die Tags „`hostaddress`“ und „`hostname`“ beigefügt werden, welche die Adresse und den Hostnamen des Devices beinhalten. In Verbindung mit `generateUnwrappers()` kann außerdem festgestellt werden, dass die von jener Funktion erzeugten Variablenbeschreiber am Ende jedes Durchgangs, also nach dem Abschluss einer Tabelle, gelöscht und jene mit dem Typ `unwrapped` mit ihren ursprünglichen Variablenbeschreiber ersetzt werden. In diesem Abschnitt des Konstruktors werden auch die benötigten Tabellen analysiert und die `dbSchema` Eigenschaft mit diesen beschrieben. Auf weitere Details des Codeabschnitts wird hier nicht eingegangen, da es den Rahmen der Arbeit sprengen würde.

```
235     //Add other tables/measurements
236     jstools.arrayMerge(calcDBData, ...Object.keys(obj.db || {}).map(tableName =>
        {
237         try {
238             if (!nameValid(tableName))
239                 throw new Error('Invalid table name found: ' + tableName);
240             let schema = { measurement: tableName, tags: {}, fields: {} }; //db
                schema object which will be added to this.dbSchema
241             let out = ['{', 'let writeOut = true;'];

```

---

```
242     let table = obj.db[tableName];
243     let [unwrapStart, unwrapEnd] = generateUnwrappers(table.unwrap, vars
244     );
245     jstools.arrayMerge(out, unwrapStart);
246
246     out.push(`let outTable = { measurement: '${tableName}', tags: {}, fields: {} };`);
247     let tableTags = Object.assign({
248       "hostaddress": {
249         "type": "string",
250         "value": "$address"
251       },
252       "hostname": {
253         "type": "string",
254         "value": "$hostname"
255       }
256     }, table.tags);
257     Object.keys(tableTags).forEach(tag => {
258       if (!tableTags[tag]) return;
259       if (!nameValid(tag))
260         throw new Error('Invalid tag name found: ' + tag);
261       let codeTag = generateValue(tableTags[tag].value, vars,
262         nextLocals, tag);
263       jstools.arrayMerge(out, codeTag[1]);
264       out.push(`outTable.tags.${tag} = ${codeTag[0]};`);
265       out.push(`if (!isExistingValue(outTable.tags.${tag})) ${
266         tableTags[tag].default === undefined ? 'writeOut = false' :
267         `outTable.tags.${tag} = ${JSON.stringify(tableTags[tag].default)}`
268       }`);
269       jstools.arrayMerge(out, codeTag[2]);
270       //Check and add tag with it's type to schema
271       let type = db.getFieldType(tableTags[tag].type);
272       if (!type)
273         throw new Error(`Invalid type at tag '${tag}': ${tableTags[
274           tag].type}`);
275       schema.tags[tag] = type;
276     });
277     table.fields && Object.keys(table.fields).forEach(field => {
278       if (!table.fields[field]) return;
279       if (!nameValid(field))
280         throw new Error('Invalid field name found: ' + field);
281       let codeField = generateValue(table.fields[field].value, vars,
282         nextLocals, field);
283       jstools.arrayMerge(out, codeField[1]);
284       out.push(`outTable.fields.${field} = ${codeField[0]};`);
285       out.push(`if (!isExistingValue(outTable.fields.${field})) ${
286         table.fields[field].default === undefined ? 'writeOut =
287         false' : `outTable.fields.${field} = ${JSON.stringify(table.
288           fields[field].default)}`
289       }`);
290       jstools.arrayMerge(out, codeField[2]);
291       //Check and add tag with it's type to schema
292       let type = db.getFieldType(table.fields[field].type);
293       if (!type)
```

```

284         throw new Error(`Invalid type at field '${field}': ${table.
285             fields[field].type}`);
286         schema.fields[field] = type;
287     });
288     out.push(`writeOut && out.push(outTable);`);

289     jstools.arrayMerge(out, unwrapEnd);
290     out.push('}');
291     Object.keys(vars).filter(k => vars[k].type === VarType.index).
292         forEach(k => delete vars[k]);
293     Object.keys(vars).forEach(vari => {
294         while (vars[vari].type === VarType.unwrapped)
295             vars[vari] = vars[vari].base;
296     });

297     //write schema into property
298     this.dbSchema.push(schema);
299     return out;
300 } catch (err) {
301     console.error('Error thrown while reading configuration for db table
302         ' + tableName.toString() + ' of device ' + this.address.
303         toString() + ':');
304     console.error(err);
305     return [];
306 }
307));

```

Als nächstes wird noch das Aktualisieren des zwischenzyklischen Speichers, also der Eigenschaft memory, zum Programmcode-Array hinzugefügt. Gibt es Probleme beim Einlesen des memorize Teils der Konfiguration, wird die Fehlermeldung „Error thrown while reading memorized variables of device <address>“ im Error-Stream ausgegeben.

```

307     //Add explicit memory of variables
308     try {
309         calcDBData = calcDBData.concat(Object.keys(obj.memorize || {}).map(
310             newKey => {
311                 if (!vars[obj.memorize[newKey]])
312                     throw new Error('Unknown variable: ' + obj.memorize[newKey]);
313                 return `this.memory['${newKey}'] = ${vars[obj.memorize[newKey]].code
314                     };`;
315             }));
316     } catch (err) {
317         console.error('Error thrown while reading memorized variables of device
318             ' + this.address.toString() + ':');
319         throw err;
320     }

```

An diesem Punkt wird die letzte Zeile an das Array angehängt, von der `createFn()` in eine Funktion umgewandelt und in die Device-Instanz geschrieben. Mit diesen Zeilen endet die Generierung der `calcDBData()` Methode.

---

```

319     //End calcDBData function
320     calcDBData.push('return out;');
321     calcDBData = calcDBData.flat();
322     //Evaluate and store finished calcDBData function
323     env.debug && env.environment === 'development' && console.log('calcDBData of
            device ' + this.address.toString() + ':', calcDBData);
324     this.calcDBData = jstools.createFn('calcDBData', calcDBData, ['data']);

```

---

Wenn die Erstellung einer Visualisierung mittels Grafana in der Konfigurationsdatei aktiviert ist (Standardfall), wird anschließend noch die `generateGrafana()` Methode erzeugt. Mit den folgenden Zeilen wird ein Array für den Programmcode deklariert und mit zwei Einträgen initialisiert.

---

```

326     //Add grafana calculations
327     if (grafana.enabled) {
328         try {
329             nextLocals.inline = 0; //reset inline locals as this is a new
                                function with it's own scope
330             let generateGrafana = [`function isExistingValue(x) { return x !=
                                undefined && !Number.isNaN(x) && Math.abs(x) !== Infinity; }`, '
                                let filledDashboards = [], filledPanels = {};', ];
331             let codePost = [];

```

---

Die erste Zeile im Array enthält die gleiche Funktion namens `isExistingValue()` wie in der Methode `calcDBData()`. Die zweite Zeile definiert die zwei lokalen Variablen namens `filledDashboards` und `filledPanels`. In `filledDashboards` werden alle erzeugten Dashboards, in `filledPanels` alle erzeugten Panels gesammelt. Das Array `codePost` wird für die Sammlung von Programmcode benutzt, welcher am Ende der Funktion positioniert sein soll.

Anschließend wird, wenn der automatische Ping aktiviert ist, das Ping-Template aus der globalen Grafana-Konfiguration (siehe Unterkapitel 5.7) zur Grafana-Konfiguration des Devices hinzugefügt.

---

```

332     //Add rtt
333     if (!this.disablePing)
334         obj.grafana = config.assignInheritedGrafana(obj.grafana || {}, 
                grafana.globalGrafanaConfig.pingTemplate || {});

```

---

Dann wird der Programmcode für die Bestimmung der benötigten Ordner und der Generierung von Panels und Dashboards erzeugt. Hierbei wird unter anderem die Funktion `generateGrafanaValue()` genutzt. Die Ordner werden in der `generateGrafana()` Methode in das übergebene Object geschrieben.

Die Panels werden nach der Generierung in der lokalen Variable `filledPanels` zwischengespeichert, um anschließend für die Dashboards verwendet zu werden. Diese werden wiederum in `filledDashboards` abgelegt. Muss dieser Generierungsprozess abgebrochen werden, wird mit dem Code „`return false;`“ `false` zurückgegeben. Auf weitere Details dieses Codeabschnitts wird hier nicht eingegangen, da es den Rahmen der Arbeit sprengen würde.

```

335         //Add folders
336         obj.grafana && obj.grafana.folders && obj.grafana.folders.forEach(
337             folder => {
338                 let toCheck = [];
339                 let codePartsUID = generateGrafanaValue(folder.uid, vars,
340                     nextLocals, 'uid', toCheck);
341                 let codePartsTitle = generateGrafanaValue(folder.title, vars,
342                     nextLocals, 'title', toCheck);
343
344                 jstools.arrayMerge(generateGrafana, codePartsUID[1],
345                     codePartsTitle[1]);
346                 jstools.arrayMerge(codePost, codePartsUID[2], codePartsTitle[2])
347                     ;
348
349                 toCheck.forEach(checkCode => generateGrafana.push(`if(!
350                     isExistingValue(${checkCode})) return false;`));
351                 generateGrafana.push(`folders[${codePartsUID[0]}] = ${
352                     codePartsTitle[0]};`);
353             });
354             //Add panels
355             obj.grafana && obj.grafana.panels && Object.keys(obj.grafana.panels)
356                 .forEach(k => {
357                     let panel = obj.grafana.panels[k];
358                     let toCheck = [];
359                     let codePartsPanel = generateGrafanaValue(panel, vars,
360                         nextLocals, k, toCheck);
361
362                     jstools.arrayMerge(generateGrafana, codePartsPanel[1]);
363                     jstools.arrayMerge(codePost, codePartsPanel[2]);
364
365                     toCheck.forEach(checkCode => generateGrafana.push(`if(!
366                         isExistingValue(${checkCode})) return false;`));
367                     generateGrafana.push(`filledPanels[${JSON.stringify(k)}] = ${
368                         codePartsPanel[0]};`);
369                 });
370             //Add dashboards
371             obj.grafana && obj.grafana_dashboards && obj.grafana_dashboards.
372                 forEach(dashboard => {
373                     let codePartsPanels = [[], [], []];
374                     let toCheck = [];
375                     const generatePanelCodeParts = (panel, outsideCodeParts) => {
376                         if (typeof panel === 'string') {
377                             outsideCodeParts[1].push(`if(!filledPanels[${JSON.
378                             stringify(panel)}]) {console.error("Panel ${JSON.
379                             stringify(panel)}.slice(1, -1)} not found in device $`)
```

```

366             {this.address.toString()}"); return false;`} );
367         return `filledPanels[${JSON.stringify(panel)}]`;
368     }
369     if (!panel || typeof panel !== 'object')
370         throw new Error('Invalid panel id: ' + panel);

371     let localCodePartsPanels = [[], [], []];
372     let codePartsUID = generateGrafanaValue(panel.uid, vars,
373         nextLocals, 'uid', toCheck);
374     let codePartsTitle = panel.rowTitle && generateGrafanaValue(
375         panel.rowTitle, vars, nextLocals, 'rowTitle', toCheck);
376     let codePartsCollapsed = panel.collapsed !== undefined &&
377         generateGrafanaValue(panel.collapsed, vars, nextLocals,
378             'collapsed', toCheck);
379     localCodePartsPanels[0] = panel.panels.map(p =>
380         generatePanelCodeParts(p, localCodePartsPanels)).join(', ');
381

382     jstools.arrayMerge(outsideCodeParts[1], codePartsUID[1]);
383     codePartsTitle && jstools.arrayMerge(outsideCodeParts[1],
384         codePartsTitle[1]);
385     codePartsCollapsed && jstools.arrayMerge(outsideCodeParts[1]
386         , codePartsCollapsed[1]);
387     jstools.arrayMerge(outsideCodeParts[1], codePartsPanels[1]);
388     jstools.arrayMerge(outsideCodeParts[2], codePartsUID[2]);
389     codePartsTitle && jstools.arrayMerge(outsideCodeParts[2],
390         codePartsTitle[2]);
391     codePartsCollapsed && jstools.arrayMerge(outsideCodeParts[2]
392         , codePartsCollapsed[2]);
393     jstools.arrayMerge(outsideCodeParts[2], codePartsPanels[2]);

394     toCheck.forEach(checkCode => generateGrafana.push(`if(!
395         isExistingValue(${checkCode})) return false;`));

396     return `{
397         collapsed: ${codePartsCollapsed ?
398             codePartsCollapsed[0] : 'true'}, panels:[${
399             localCodePartsPanels[0]}], ${codePartsTitle ? `title: ${
400             codePartsTitle[0]},` : ''}uid: ${codePartsUID[0]}, type:
401             "row"}`;
402     };

403     let codePartsUID = generateGrafanaValue(dashboard.uid, vars,
404         nextLocals, 'uid', toCheck);
405     let codePartsTitle = dashboard.title && generateGrafanaValue(
406         dashboard.title, vars, nextLocals, 'title', toCheck);
407     let codePartsFolder = dashboard.folder && generateGrafanaValue(
408         dashboard.folder, vars, nextLocals, 'folder', toCheck);
409     codePartsPanels[0] = '[' + dashboard.panels.map(panel =>
410         generatePanelCodeParts(panel, codePartsPanels)).join(',') +
411         ']'; //, vars, nextLocals, `panels[${i}]`, toCheck

412     jstools.arrayMerge(generateGrafana, codePartsUID[1]);
413     codePartsTitle && jstools.arrayMerge(generateGrafana,

```

```

    codePartsTitle[1]);
398   codePartsFolder && jstools.arrayMerge(generateGrafana,
      codePartsFolder[1]);
399   jstools.arrayMerge(generateGrafana, codePartsPanels[1]);
400   jstools.arrayMerge(codePost, codePartsUID[2]);
401   codePartsTitle && jstools.arrayMerge(codePost, codePartsTitle
      [2]);
402   codePartsFolder && jstools.arrayMerge(codePost, codePartsFolder
      [2]);
403   jstools.arrayMerge(codePost, codePartsPanels[2]);

405   toCheck.forEach(checkCode => generateGrafana.push(`if(!
      isExistingValue(${checkCode})) return false;`));
406   generateGrafana.push(`filledDashboards.push({uid: ${codePartsUID
      [0]}, ${codePartsTitle ? `title: ${codePartsTitle[0]}` : ''}
407 ${codePartsFolder ? `folder: ${codePartsFolder[0]}` : ''}panels: ${
      codePartsPanels[0]}});`);
408 );

```

Als nächstes wird der in der lokalen Variable codePost zwischengespeicherte Code und eine Zeile für das Hinzufügen der generierten Dashboards zum übergebenen Array an das Ende von generateGrafana angehängt.

```

410   jstools.arrayMerge(generateGrafana, codePost);

412   generateGrafana.push(`filledDashboards.forEach(d => dashboards.push(
      d));`);

```

An diesem Punkt wird die letzte Zeile an das Array angehängt, von der createFn() in eine Funktion umgewandelt und in die Device-Instanz geschrieben. Mit diesen Zeilen endet die Generierung der generateGrafana() Methode.

```

414 //End generateGrafana function
415 generateGrafana.push('return true;');
416 generateGrafana = generateGrafana.flat();
417 //Evaluate and store finished generateGrafana function
418 env.debug && env.environment === 'development' && console.log(
      generateGrafana + ' of device ' + this.address.toString() + ':',
      generateGrafana);
419 this.generateGrafana = jstools.createFn('generateGrafana',
      generateGrafana, ['dashboards', 'folders', 'data']);
420 } catch (err) {
421   console.error('Error thrown while reading configuration for grafana
      of device ' + this.address.toString() + ':');
422   console.error(err);
423 }
424 }

```

Am Ende des Konstruktors wird noch die Funktion `initDevice()` von jedem Modul ausgeführt, um das Erstellen benötigter Laufzeitobjekte (zum Beispiel Sockets oder Threads) zu ermöglichen. Der Konstruktor endet mit dem folgenden Code:

```
426     Object.values(interfaces).forEach(inf => typeof inf.initDevice === 'function'  
        && inf.initDevice(this, obj));  
427 }
```

---

#### 5.4.4 grafana

Die `grafana` Bibliothek enthält Funktionen für die Interaktion mit einem Grafana-Server. Hierfür wird Grafanas HTTP-API verwendet. Weiters sammelt die Bibliothek die von den Devices kommenden Dashboards und Ordnerinformationen und fügt sie zusammen. Objekte mit identer UID werden im weiteren Verlauf als Duplikate bezeichnet. Die folgenden Funktionen und Variablen werden exportiert.

`globalGrafanaConfig` enthält die globale Konfiguration der Darstellung mittels Grafana. Diese wird beim Laden der Bibliothek mit der internen Funktion `loadGlobalConfig()` generiert.

`enabled` gibt an, ob die automatisierte Interaktion mit Grafana aktiviert ist. Diese Variable gleicht dem Wert „enable“ aus der Konfiguration des NetBat-Servers und entspricht dem code „`env.config.grafana.enable`“.

`applyDashboard()` ist eine asynchrone Funktion. Sie sendet ein übergebenes Dashboard an den Grafana-Server. Hierfür wird eine POST-Anfrage an den Endpoint „/api/dashboards/db“ gesendet und die Antwort zurückgegeben. Mit einem zusätzlichen Parameter kann gesteuert werden, ob im Fall eines vorhandenen Duplikats dieses überschrieben oder die Anfrage verworfen werden soll. [6]

`setFolder()` ist ebenfalls eine asynchrone Funktion. Sie erzeugt oder aktualisiert Ordner auf dem Grafana-Server. Um einen neuen Ordner zu erstellen, wird eine POST-Anfrage an den Endpoint „/api/folders“ gesendet. Bei einer Aktualisierung muss die PUT-Methode verwendet und die UID an die Uniform Resource Locator (URL) angehängt werden, somit lautet der Endpoint „/api/folders/<uid>“. [6] Um entscheiden zu können, welche Anfrage verwendet wird, wird geprüft, ob die verlangte UID im Ordnercache (die interne Variable `foldersCache`) belegt ist. Wurde der Cache noch nicht geladen, geschieht dies mit der Funktion `getFolders()`. Die in der Antwort der jeweiligen Anfrage enthaltene ID [6] wird zurückgegeben und im Ordnercache unter der UID abgelegt. Die übergebene UID wird zu Beginn der Funktion mit der internen Funktion `replaceUID()` in eine mit Grafana kompatible Form gebracht. Wie bei `applyDashboard()` kann mit einem zusätzlichen Funktionsparameter gesteuert werden, ob in dem Fall, dass die UID belegt ist, der entsprechende Ordner aktualisiert oder die Anfrage verworfen werden soll.

Auch bei **getFolders()** handelt es sich um eine asynchrone Funktion. Sie ruft die Liste aller in Grafana vorhandenen Ordner mit einer GET-Anfrage an den Endpoint „/api/folders“ ab und gibt diese als Object zurück. Die Ordner werden hier jeweils unter ihren UIDs abgelegt. [6]

**mergeAndFinalizeConf()** ist eine asynchrone Funktion, welche ein Dashboard-Array und Object mit Ordner-UIDs und -namen erhält. Sie erzeugt bzw. aktualisiert die übergebenen Ordner am Grafana-Server mit der Funktion **setFolder**, kombiniert Duplikate im übergebenen Array und wandelt die resultierenden Dashboard-Konfigurationen in „vollständige“ Grafana-Dashboards um. Mit „vollständig“ ist gemeint, dass sämtliche Eigenschaften der Dashboards mit Ausnahme der UID, des Titels und der Panels mit Standardwerten befüllt werden. Für eine Auflistung dieser Eigenschaften siehe [6]. Das Erzeugen der Ordner zu Beginn der Funktion ist nötig, um die IDs der Ordner herauszufinden. Diese werden benötigt, da Grafana beim Erzeugen von Dashboards die ID des Ordners verlangt, in den das Dashboard eingeordnet werden soll. [6] Die Kombination von Dashboard-Duplikaten erfolgt so, dass jene mit einem höheren Index im übergebenen Array eine größere Priorität erhalten und somit Titel und Ordner der vorherigen überschreiben. Hat ein Dashboard nach Kombination aller Duplikate keinen Titel, wird „Untitled“ als Standardtitel verwendet. Die UID der Dashboards wird mit der internen Funktion **replaceUID()** in eine mit Grafana kompatible Form gebracht. Die Panel-Arrays werden aneinandergehängt. Duplikate der Panels vom Typ „row“ werden innerhalb eines Dashboards durch die Funktion **mergePanels()** zusammengefasst. Für Informationen zu „Rows“ siehe Unterkapitel 6.5. Die resultierenden Panels werden wie die Dashboards mit der Funktion **setAllAdditionalPanelData()** um zusätzliche Eigenschaften erweitert. Hierfür wird ein temporäres Object erzeugt, welches die weiteren Parameter der Funktion enthält und mit `{y: 0, id: 1}` initialisiert wird. Die nur vom NetBat-Server benötigte Eigenschaft **folder** wird aus den Dashboards gelöscht. Zurückgegeben wird ein Array aus Arrays, wobei die inneren jeweils ein fertiges Dashboard und die ID des Ordners, dem das Dashboard zugeordnet werden soll, enthalten.

**generateGrafanaOnDevice()** führt die **generateGrafana()** Methode eines übergebenen Devices aus und gibt deren Rückgabewert zurück. Ist die Generierung erfolgreich, wird das Device an die interne Funktion **onDeviceFinished()** übergeben. Auf den Abschluss von **onDeviceFinished()** wird nicht gewartet. Diese Funktion wird verwendet, damit die internen Variablen **tmpDashboards** und **tmpFolders** zur Befüllung an **generateGrafana()** übergeben werden können.

**waitForDevicesAndApply()** setzt die interne Variable **neededDevices** auf das übergebene Array mit Device-Instanzen. Jene, die sich bereits im internen Array **finishedDevices** befinden, werden herausgefiltert. Ist das resultierende Array leer, muss also auf kein Device mehr gewartet werden, wird **applyTmpData()** ausgeführt.

Für den internen Gebrauch enthält die Bibliothek die folgenden Funktionen und Variablen, welche nicht exportiert werden.

**foldersCache** ist ein Cache für die am Grafana-Server vorhandenen Ordner. Es handelt sich um ein Object, bei dem die Keys die Ordner-UIDs und die Values die von Grafana erhaltenen Ordnerobjekte sind (siehe Funktion `getFolders()`). Diese Variable wird mit null initialisiert.

**finishedDevices** ist ein Array, welches die Device-Instanzen enthält, die ihre Dashboards bereits erfolgreich mit ihrer Methode `generateGrafana()` generiert haben.

**neededDevices** ist ein Array, welches die Device-Instanzen enthält, die ihre Dashboards noch nicht erfolgreich mit ihrer Methode `generateGrafana()` generiert haben. Bis zur Verwendung von `waitForDevicesAndApply()` ist es leer.

**tmpDashboards** ist ein Array, welches alle bereits generierten Dashboards enthält. Es wird von `generateGrafanaOnDevice()` an die Device-Methode `generateGrafana()` übergeben und dort beschrieben.

**tmpFolders** ist ein Object, das den gesetzten Ordner-UIDs Titel zuordnet. Es wird von der Funktion `generateGrafanaOnDevice()` an die Device-Methode `generateGrafana()` übergeben und dort beschrieben.

**loadGlobalConfig()** lädt die globale Konfiguration der automatischen Visualisierung mit Grafana. Die Werte der allgemeinen `grafana_global.json` werden hier mit denen der lokalen Datei (siehe Unterkapitel 5.7) zusammengeführt. Für die lokale Datei werden die Dateiformate JSON und YAML unterstützt. Anschließend wird `generateGlobalGrafana()` verwendet, um die dort definierten Dashboards und Ordner in die internen Variablen `tmpDashboards` und `tmpFolders` zu übernehmen. Da diese Funktion beim Laden der Bibliothek ausgeführt wird, befinden sich die globalen Dashboards immer am Anfang von `tmpDashboards` und können so von den Devices überschrieben werden.

**promiseRequest()** wird für asynchrone HTTP Anfragen verwendet. Das für HTTP genutzte npm-Paket „simple-get“ unterstützt nur die Verwendung mit Callbacks, also als Parameter übergebene Funktionen, welche nach Abschluss des Prozesses aufgerufen werden. [16] `promiseRequest()` umgibt den simple-get-Aufruf mit einem Promise, der bei Abschluss der Anfrage mit den erhaltenen Daten aufgelöst wird und bei einem Fehler verworfen wird. Wird ein HTTP-Statuscode zurückgegeben, der sich nicht im Bereich 200-399 befindet, wird erfolgreich aufgelöst und die Nachricht „Request to <url> returned with status code <status code> and body: <body>“ in der Konsole ausgegeben. „<body>“ ist hier die Antwort der Anfrage.

---

**replaceUID()** bringt UIDs in eine mit Grafana kompatible Form. Dies ist nötig, da sich in Tests gezeigt hat, dass Grafana in den UIDs von Ordnern und Dashboards nur alphanumerische Zeichen, Unterstriche und Bindestriche erlaubt. Mit der folgenden Codezeile werden alle anderen Zeichen mit Ausnahme von Punkten, Doppelpunkten und Whitespaces entfernt und Punkte, Doppelpunkte und Whitespaces durch Unterstriche ersetzt.

---

```
114     return uid.replace(/[^w\s_.:-]/g, '').replace(/[:\s]/g, '_');
```

---

Die spezielle Behandlung von Punkten, Doppelpunkten und Whitespaces wird durchgeführt, um IP-Adressen und Strings mit mehreren Wörtern lesbar zu halten.

**mergePanels()** erhält ein Array aus Panels, fügt Duplikate der enthaltenen „Rows“, also Panels vom Typ „row“, zusammen und gibt das resultierende Array zurück. Die Kombination von Row-Duplikaten erfolgt so, dass jene mit einem höheren Index im übergebenen Array eine größere Priorität erhalten und somit Titel und Öffnungsstatus (`collapsed`) der vorherigen überschreiben. Hat eine Row nach Kombination aller Duplikate keinen Titel, wird „Untitled“ als Standardtitel verwendet. Die Panel-Arrays werden aneinandergehängt und an `mergePanels()` selbst übergeben, womit eine Rekursion entsteht. Die von NetBat verwendete UID der Rows wird entfernt. (nähere Informationen zu Rows siehe Unterkapitel 6.5).

**setAllAdditionalPanelData()** erweitert das übergebene Panel um zusätzliche Eigenschaften, die von Grafana benötigt werden. Hierfür wird ein zusätzliches Object übergeben, welches die Eigenschaften „id“ und „y“ enthält. Diese geben die Panel-ID und die y-Koordinate des nächsten Panels an. Die ID des Panels („id“) wird auf die Eigenschaften „id“ des zusätzlich übergebenen Objects gesetzt, welche anschließend um 1 inkrementiert wird. Die Position des Panels („gridPos“) ist ein Object mit den Eigenschaften „w“, „h“, „x“ und „y“, welche für die Breite, Höhe, X- und Y-Koordinate des Panels stehen. Wenn in der übergebenen Konfiguration nichts anderes angegeben wird, werden die Breite, Höhe und X-Koordinate mit den Standardwerten 24, 8 und 0 beschrieben. Die Y-Koordinate wird immer auf die Eigenschaft „y“ des Object-Parameters gesetzt, welcher anschließend um 8 inkrementiert wird. Ist im Panel die Eigenschaft „alert“ vorhanden, wird „alert.notifications“ auf den „defaultAlertNotifications“ Wert der globalen Grafana-Konfiguration gesetzt. Wenn es sich bei dem Panel um eine „Row“ handelt, so ist die Standardhöhe 1 und die enthaltenen Panels werden ebenfalls durch einen Aufruf der `setAllAdditionalPanelData()` Funktion erweitert.

**applyTmpData()** ist eine asynchrone Funktion. Sie übergibt die gesammelten Dashboards und Ordner aus den internen Variablen `tmpDashboards` und `tmpFolders` an die oben genannte Funktion `mergeAndFinalizeConf()` und sendet die erhaltenen vollständigen Dashboards mit `applyDashboard()` auf den Grafana-Server. Bei Erfolg wird `true` zurückgegeben. Bei einem Fehler wird `false` zurückgegeben und „Error thrown during finalization of Grafana configuration“ in den Error-Stream geschrieben.

**onDeviceFinished()** ist eine asynchrone Funktion. Sie fügt die übergebene Device-Instanz zu finishedDevices hinzu, entfernt sie aus neededDevices und führt applyTmpData() aus, wenn kein Device mehr in neededDevices verblieben ist.

**generateGlobalGrafana()** erzeugt Dashboards und Ordner aus der übergebenen Konfiguration und speichert sie in die übergebenen Parameter für Dashboards und Ordner. Diese Funktion wird ausschließlich in loadGlobalConfig() mit der globalen Grafana-Konfiguration und den internen Variablen tmpDashboards und tmpFolders aufgerufen.

## 5.4.5 jstools

Die jstools Bibliothek enthält diverse Funktionen, welche die Standardfunktionalität von JavaScript und Node.js® erweitern. Sie beinhalten keinen Bezug zu NetBat und können in beliebigen Node.js® Projekten verwendet werden. Hierzu gehören beispielsweise spezielle Array Operationen, asynchrone Schleifen und rekursive Object Operationen. Die folgenden Funktionen werden exportiert: arrayMerge, createFn, deepAssign, deepAssignOneValue, deepEquals, deepEqualsStrict, deepFreeze, deepMapObject, filterObject, forEachAsync, mapAsync, mapObject, reduceAsync, round, swapObjEntries.

Zur Funktion arrayMerge ist zu erwähnen, dass hier aufgrund der häufigen Verwendung im Konstruktor der Device Klasse nach dem performantesten Weg gesucht wurde, ein Array mit hunderttausenden Elementen an ein anderes anzuhängen. Für die Situation

---

```
1 let arr1 = Array.from({ length: 100000 }, (x, i) => i);
2 let arr2 = Array.from({ length: 100000 }, (x, i) => i);
```

---

wurden die folgenden Varianten getestet.

Variante 1: „concat“

---

```
1 let result = arr1.concat(arr2);
```

---

Variante 2: „new with spread“

---

```
1 let result = [...arr1, ...arr2];
```

---

Variante 3: „spread push“

---

```
1 arr1.push(...arr2);
2 let result = arr1;
```

---

---

#### Variante 4: „push loop“

---

```

1 for(let i = 0; i < arr2.length; i++)
2   arr1.push(arr2[i]);
3 let result = arr1;

```

---

#### Variante 5: „set loop“

---

```

1 let len1 = arr1.length;
2 for (let i = 0; i < arr2.length; i++)
3   arr1[len1 + i] = arr2[i];
4 let result = arr1;

```

---

#### Variante 6: „set loop with prealloc“

---

```

1 let len1 = arr1.length, len2 = arr2.length;
2 arr1.length = len1 + len2;
3 for (let i = 0; i < len2; i++)
4   arr1[len1 + i] = arr2[i];
5 let result = arr1;

```

---

Aufgrund der Testergebnisse, welche in Tabelle 5.1 aufgelistet sind, wird Variante 6, „set loop with prealloc“, für arrayMerge() verwendet.

Variante	Durchläufe pro Sekunde
concat	886,32
new with spread	222,04
spread push	82,15
push loop	461,31
set loop	449,20
set loop with prealloc	1004,77

Tabelle 5.1: Ergebnisse der Geschwindigkeitsmessung für das Zusammenhängen von Arrays

## 5.5 Modulare Datenquellen

Bei den Datenquellenmodulen, auch „Modulen“ oder „Interfaces“, handelt es sich um JavaScript-Dateien, die im Ordner `interfaces` abgelegt sind. Um als Module geladen werden zu können, müssen sie die Funktionen `genRetrieverCode()` und `getRetrievalInterface()` exportieren. Optional können die Funktionen `initDevice()` und `destructDevice()` exportiert werden.

**genRetrieverCode()** generiert den Programmcode, mit dem die Daten der jeweiligen Quelle in der `retrieveData()` Methode eines Devices erhalten werden. Es werden vier Parameter übergeben: die Konfiguration der Quelle aus der Device-Konfiguration (`pingConfig`), das Object der Variablennamen-Variablenbeschreibungen-Zuordnungen (`vars`), die Device-Instanz (`device`) und das String-Array, an das die Codezeile angefügt werden sollen (`code`).

**getRetrievalInterface()** wird verwendet, um vom Modul benötigte Daten, Objekte und Funktionen an die `retrieveData()` Methode der Devices zu übergeben. Der Rückgabewert dieser Funktion wird in den Object-Parameter `retInt` der `retrieveData()` Methode unter dem Namen des Moduls abgelegt.

**initDevice()** wird am Ende des Device-Konstruktors aufgerufen und erhält die Device-Instanz und ihre Konfiguration als Parameter.

**destructDevice()** wird bei Aufruf der Methode `destruct()` eines Devices aufgerufen und erhält die Device-Instanz als Parameter.

Es werden sieben Module mit NetBat mitgeliefert: `ping`, `snmp`, `http`, `munin`, `netbat` und `cmd`. Weitere Module können zusätzlich implementiert werden. Die Namen „`address`“, „`hostname`“, „`samplePeriod`“, „`disablePing`“, „`db`“, „`memorize`“, „`grafana`“ und „`extends`“ sind aufgrund der Struktur der Device-Konfiguration reserviert und können nicht verwendet werden.

### 5.5.1 ICMP

Das Modul `ping` nutzt ICMP um Pings zu versenden, damit die Verbindung auf Funktionalität zu testen und die Round Trip Time (RTT) zu messen. Pro Ping-Anfrage des Moduls werden zwei Pings gesendet. Wird statt einer IP-Adresse eine Domain angegeben, wird diese vor dem Ping aufgelöst.

Das Modul exportiert die asynchrone Funktion `collectRTT()` zusätzlich zu den benötigten Funktionen. Diese sendet durch Nutzung des npm-Pakets „`ping`“ einen Ping an die übergebene Adresse bzw. Domain und gibt ein Object zurück, das die Eigenschaften `alive` und `time` besitzt. Der Boolean `alive` gibt den Erfolg des Pings an, `time` die mittele gemessene RTT.

Die `getRetrievalInterface()` Funktion gibt ein Object zurück, welches eine Referenz auf die zuvor genannte exportierte Funktion mit dem Namen `collectRTT()` enthält.

Die Konfiguration des Moduls ist wie folgt aufgebaut.

---

```
{
  "values": {
    <variable name>: <ip|domain>
  }
}
```

---

values ist ein Object, das Variablennamen entweder IP-Adressen oder Domains zuweist. Vom Modul werden pro Eintrag in values zwei Variablen erzeugt. Die Namen dieser Variablen bestehen aus dem angegebenen Namen und den Suffixen „\_rtt“ und „\_alive“. Die Variable mit dem Suffix „\_alive“ enthält einen Boolean, der angibt, ob der Ping erfolgreich gewesen ist. Die Variable mit dem Suffix „\_rtt“ enthält die mittlere gemessene RTT oder 0, wenn der Ping nicht erfolgreich beantwortet wurde.

Beispielsweise würde die Konfiguration

---

```
{  
    "values": {  
        "haus": "10.0.0.1"  
    }  
}
```

---

einen Ping an die IP-Adresse „10.0.0.1“ senden, den Erfolgsstatus in die Variable „haus\_alive“ und die RTT in die Variable „haus\_rtt“ speichern.

### 5.5.2 SNMP

Das Modul `snmp` nutzt SNMP, um Betriebsinformation aus Netzwerkgeräten zu lesen. Der Lesevorgang erfolgt immer für das Gerät, in dem sich die Konfiguration befindet. Da das verwendete npm-Paket „snmp-native“ nur SNMPv2c unterstützt, ist das auch bei NetBat der Fall.

Für die Verwendung des Moduls wird die zusätzliche Einstellung „defaultCommunity“ in der globalen Konfiguration in der Sektion „snmp“ benötigt. Diese gibt die Community an, die für Abfragen verwendet wird, wenn keine andere angegeben wird.

Das Modul exportiert, zusätzlich zu den benötigten Funktionen, die Funktionen `OIDToArr()` und `isValidOID()` sowie die Klasse `Session`.

`OIDToArr()` wandelt eine OID in Form eines Strings in ein Array aus Zahlen um, indem der String bei allen Punkten getrennt wird. Wird ein Array übergeben, wird es unverändert zurückgegeben.

`isValidOID()` prüft, ob der übergebene String eine korrekte OID ist, also ob er aus Zahlen zwischen 0 und 255 besteht, die mit Punkten voneinander getrennt sind. Ein optionaler zweiter Parameter kann mit `true` übergeben werden, um zu erlauben, dass statt einer Zahl ein einzelnes Dollarzeichen verwendet werden kann.

Die Session Klasse enthält die Abfragemethoden `get()`, `getNext()`, `getAll()` und `getSubtree()`, mit denen Daten von einem Gerät gelesen werden können. Die Funktion `close()` schließt eine Session und gibt die von ihr benötigten Ressourcen frei.

Intern greifen Session-Instanzen auf Instanzen der gleichnamigen Klasse des npm-Pakets „snmp-native“ zu. Diese sind als Eigenschaft namens `session` zu finden.

Die Session Klasse enthält zusätzlich die private statische Eigenschaft `mainSession` und den statischen Getter `mainSession`, der die gleichnamige private Variable zurückgibt und beim ersten Aufruf zuerst mit einer neuen Session-Instanz befüllt.

Die `getRetrievalInterface()` Funktion gibt die vom Getter `mainSession` erhaltene Session-Instanz zurück.

Die Konfiguration des Moduls ist wie folgt aufgebaut.

---

```
{
    "community"?: <string>,
    "values": {
        <variable name>: <oid string>
    }
}
```

---

`community` gibt optional die verwendete Community an. Wird keine Community angegeben, wird die in der globalen Konfiguration unter „snmp“ angegebene Einstellung „`defaultCommunity`“ verwendet. `values` weist Variablennamen OIDs zu. Die so definierten Variablen enthalten die von den OIDs gelesenen Daten.

Es können mehrere OIDs entlang einer Ebene gesammelt gelesen werden, indem das jeweilige Byte der OID mit einem Dollarzeichen ersetzt wird. Das Ergebnis ist ein Object, das den Werten des gewählten Bytes die erhaltenen Werte zuordnet. Werden mehrere Bytes durch Dollarzeichen ersetzt, bildet sich eine Baumstruktur, wobei das erste, am weitesten links stehende Byte der äußersten Ebene entspricht.

Diese Mechanik zeigt sich im folgenden Beispiel. Mit der Community „steve“ liefert ein Gerät die Daten:

OID	Wert
1.0.1.1	„hi“
1.0.1.3	3
1.0.2.1	21
1.1.6.2	90

---

Das Modul wird nun in der Device-Konfiguration folgendermaßen konfiguriert:

---

```
{
    "community": "steve",
    "values": {
        "objX": "1.0.$.$",
        "num": "1.1.6.2"
    }
}
```

---

Die Variablen werden jetzt mit folgenden Werten befüllt:

Variable	Wert
objX	{ "1": { "1": "hi", "3": 3 }, "2": { "1": 21 } }
num	90

### 5.5.3 HTTP/HTTPS

Das Modul `http` ermöglicht, HTTP-Anfragen zu senden und die erhaltenen Daten zu verwenden. Intern wird das npm-Paket `simple-get` für die Netzwerkkommunikation verwendet.

Die Verwendung des HTTP-Moduls erfolgt in „Requests“. Ein Device kann mehrere voneinander unabhängige Requests definieren. Sollen Daten zwischen Requests ausgetauscht werden, können „Request-Variablen“ verwendet werden. Dies erfolgt, indem der Regex `\$w+$` in den URLs, den Headernamen, den Headerinhalten und den zusätzlichen Daten durch den Wert der zwischen den Paragraphenzeichen angegebenen Request-Variable ersetzt wird. Die Namen von Request-Variablen dürfen nur alphanumerische Zeichen und Unterstriche beinhalten. Um ein einzelnes Paragraphenzeichen in den String zu schreiben, muss „§§“ geschrieben werden. Die Request-Variablen „time\_ms“, „hostname“, „address“ und „devicegroup“ sind automatisch in allen Devices verfügbar und enthalten die Systemzeit zum Zeitpunkt der Abfrage in ms sowie den Hostname, die Adresse und die Gerätegruppe des Devices.

In einem internen Object namens `sessions` wird jedem Device mit HTTP-Konfiguration das Object zugeordnet, welches von `getRetrievalInterface()` zurückgegeben wird. In diesem Object befinden sich eine Referenz auf die interne Funktion `promiseRequest()` unter dem Namen `request` und ein Object unter dem Namen `reqVars`, in dem die Werte der Request-Variablen gespeichert werden. Diese Device-bezogenen Objects werden in der Funktion `initDevice()` erzeugt und in der Funktion `destructDevice()` wieder gelöscht.

---

Die Konfiguration des Moduls ist wie folgt aufgebaut.

---

```
{
    <request name>: {
        "url": <url string>,
        "method"??: <string>,
        "headers"??: {
            <header name>: <string>
        }
        "data"??: <any>
        "decode"??: <string>
        "store": <variable name|object>
    }
}
```

---

Es handelt sich um ein Object, welches mehrere Objects enthält. Diese Objects entsprechen den Requests. Die Keys, unter denen sie abgelegt sind, werden als ihre Namen bezeichnet. Der Aufbau als Object mit einer Benennung für jede Request ermöglicht es Devices und Templates, die Requests der Basistemplates zu modifizieren. Eine Request besitzt die in der Folge beschriebenen Eigenschaften:

url ist die URL, welche von der Request aufgerufen werden soll. Wird in der URL das Protokoll weggelassen, wird HTTP verwendet. Die Adresse bzw. die Domain kann ebenfalls weggelassen werden. In diesem Fall wird die Adresse des Devices verwendet. Um beispielsweise den Endpoint „/test“ des Devices mit der IPv4-Adresse „10.0.0.1“ und der Domain „testdevice.at“ über HTTP abzufragen, können für url folgende Werte verwendet werden: „http://testdevice.at/test“, „testdevice.at/test“, „http://10.0.0.1/test“, „10.0.0.1/test“, „http://\$address\$/test“, „\$address\$/test“, „http:////test“ und „/test“.

Die optionale Eigenschaft method gibt die HTTP-Methode an. Die Methoden GET, POST, PUT, DELETE, PATCH und HEAD werden unterstützt.

Über das optionale Object headers können HTTP-Header übergeben werden.

Mit der optionalen Eigenschaft data können zusätzlich Daten im Körper der Anfrage mitgesendet werden. Objects und Arrays werden hier in einen JSON-String umgewandelt.

Die Antwort der Nachricht kann optional dekodiert werden, indem das Format in decode angegeben wird. Die einzige verfügbare Kodierung ist JSON, wofür decode auf "json" gesetzt wird. Die Werte null, 0 und "" werden ignoriert, sorgen also für die Verwendung von keinem Dekodierverfahren.

Die Eigenschaft store gibt den Namen der Variable an, in der die Antwort gespeichert werden soll. Um eine Antwort in eine Request-Variable zu speichern, wird deren Name umgeben von Paragraphenzeichen angegeben. Wenn die Antwort als JSON-Object dekodiert wird, können Teile des erhaltenen Objects in unterschiedliche Variablen und Request-Variablen gespeichert werden. Hierfür wird für store

ein Object benutzt, dessen Struktur jener der Antwort gleicht. In diesem werden die Variablennamen an der Stelle platziert, an der sich die zuzuweisenden Daten befinden.

Beispielsweise würden die Variablen mit der dekodierten Antwort

---

```
{
    "key1": {
        "key2": 1
    },
    "key3": 2
}
```

---

und dem store-Object

---

```
{
    "key1": {
        "key2": "$var1$"
    },
    "key3": "var2"
}
```

---

folgendermaßen befüllt werden:

Request-Variable var1 = 1

Variable var2 = 2

## 5.5.4 Munin

Das Modul `munin` ermöglicht Daten von Munin-Nodes auszulesen. Für die Netzwerkkommunikation intern wird das npm-Paket `node-munin-client` verwendet. Der Lesevorgang erfolgt immer für das Gerät, in dem sich die Konfiguration befindet.

Das Modul enthält die interne Klasse `MuninRetriever`. Hierbei handelt es sich um eine Erweiterung der Klasse `MuninClient` aus dem verwendeten npm-Paket, die sich um den automatischen Aufbau, Wiederaufbau und Abbau der Verbindung kümmert. Hierbei wird eine Eigenschaft `session` vom Typ `MuninClient` verwendet, die bei einem Abbruch der Verbindung auf `undefined` gesetzt wird. Wenn kein `MuninClient` in `session` vorhanden ist, ruft die Methode `getMetrics()`, die eine Datenabfrage an die `MuninClient`-Instanz weiterleitet, die Methode `connect()` auf und stellt somit die Verbindung nach einem Fehler automatisch wieder her. Die Methode `connect()` erzeugt eine neue `MuninClient`-Instanz, initiiert deren Verbindung und schreibt sie in die Eigenschaft `session`.

Um Debug-Meldungen der `MuninClients` im Log besser erkennen zu können, wird die `log()` Methode der Klasse `MuninClient` überschrieben und das Wort „Munin“ an den Beginn der Nachrichten angehängt.

Für jedes Device, bei dem das Modul konfiguriert ist, wird von der Funktion `initDevice()` eine MuninRetriever-Instanz erstellt und im internen Object `sessions` unter dem Device abgelegt. Die `getRetrievalInterface()` Funktion gibt den MuninRetriever des übergebenen Devices zurück. Nicht mehr benötigte MuninRetriever werden von der Funktion `destructDevice()` wieder gelöscht.

Die Konfiguration des Moduls ist wie folgt aufgebaut:

---

```
{
    "port"? : <number>,
    "values": {
        <plugin name>: <variable name> | {
            <variable name>: <field name>
        }
    }
}
```

---

Der für Munin verwendete Port kann mit der Eigenschaft `port` angegeben werden. Der Standardport ist 4949. Die Eigenschaft `Values` enthält einen Eintrag für jedes Plugin, auf das zugegriffen werden soll, wobei die Keys die Namen der Plugins sind. Die als `Values` zu findenden Objects enthalten einen Eintrag pro gelesenem Field, wobei die Keys die Variablennamen und die Values die Namen der Fields angeben. Wird statt dem Object ein String als Wert für das Plugin angegeben, gibt dieser den Namen der Variable an, in der ein Object mit allen Daten des Plugins abgelegt wird. Hierbei werden die Namen der Fields als Keys und ihre Werte als Values verwendet.

Das folgende Beispiel dient der Veranschaulichung dieser Funktion: Ein Gerät enthält die folgenden Daten.

Plugin	Field	Wert
cpu	idle	50
ram	system	100
	user	2000

Das Modul ist in der Device-Konfiguration nun folgendermaßen konfiguriert:

---

```
{
    "values": {
        "cpu": {
            "var1": "idle"
        },
        "ram": "var2"
    }
}
```

---

Die Variablen werden daher mit folgenden Werten befüllt:

Variable	Wert
var1	50
var2	{ "system": 100, "user": 2000 }

### 5.5.5 NetBat Logo

Das Modul netabt liefert periodisch die in Unterkapitel 3.2.1 beschriebenen Daten. Der Index wird aus der Systemzeit ermittelt und erhöht sich ein Mal pro Sekunde.

Die Funktion getRetrievalInterface() gibt in diesem Modul immer undefined zurück.

Die Konfiguration des Moduls ist wie folgt aufgebaut.

---

```
[<variable1>,<variable2>,<variable3>]
```

---

Sie ist ein Array aus drei Strings, welche die Namen der Variablen angeben, in die die Daten geschrieben werden sollen. Die erste Variable erhält Datensatz 1, die zweite Datensatz 2 und die dritte Datensatz 3.

### 5.5.6 Befehlsausgabe

Das Modul cmd ermöglicht das Ausführen von Kommandozeilenbefehlen und das Verarbeiten ihrer Ausgaben.

Die Funktion getRetrievalInterface() gibt ein Object mit den zwei Methoden exec() und createRegex() zurück. exec() führt einen Befehl aus und gibt dessen Ausgaben zurück. Es handelt sich um die mit der Node.js Bibliotheksfunktion util.promisify() in eine Promise-Variante umgewandelte Form (vgl. [14]) der Node.js Bibliotheksfunktion child\_process.exec(). Die Methode createRegex() erzeugt ein RegExp-Objekt aus dem übergebenen String.

Die Konfiguration des Moduls ist wie folgt aufgebaut.

---

```
{<command name>: {"command": <string>,"output"?: {<variable name>: <regex>}},}
```

---

```
"error"?: {
    <variable name>: <regex>
}
}
```

---

Jeder Befehl wird als eigenes Object angegeben. Die dafür verwendeten Keys, die Namen der Befehle, sind nicht die Namen der auszuführenden Kommandozeilenbefehle, sondern dienen ausschließlich der Identifikation der Konfigurationsobjekte bei der Vererbung!

Die optionalen Eigenschaften output und error enthalten die Information über die Verarbeitung der Ausgabe. output verwendet hierfür den Standardausgabestream und error den Standarderrorstream. In den jeweiligen Streams wird nach den als Values angegebenen Regexen gesucht. Die die Regexe erfüllenden Textabschnitte werden in den Variablen gespeichert, deren Namen den jeweiligen Keys entsprechen. Ist im Regex das Flag „global“ gesetzt, wird ein Array aller gefundener Textbereiche, ansonsten die erste gefundene Übereinstimmung abgelegt.

Die zu suchenden Regexe können ohne und mit Begrenzungszeichen („/“), jeweils eins vor und nach dem Regex, angegeben werden. Sind Begrenzungszeichen vorhanden, muss das erste am Anfang des Strings stehen. Die Zeichen nach dem zweiten Begrenzungszeichen geben die Flags des RegExp-Objekts an. So wird beispielsweise das genannte Flag „global“ durch das Anhängen des Zeichens „g“ gesetzt. [15].

Im Befehl und in den Regexen können die vordefinierten Variablen (time\_ms, address, hostname und devicegroup) mit dem Platzhalter „\${varname}“ eingefügt werden, wobei „varname“ den Namen der Variable symbolisiert. „\${\$}“ wird mit „\$“ ersetzt.

### 5.5.7 Implementierung weiterer Datenquellen

Um zusätzliche Datenquellen zu implementieren, muss eine JavaScript-Datei im Ordner interfaces erstellt werden. Der Name dieser Datei gleicht dem Namen des Moduls.

Um als Modul geladen werden zu können, müssen die Funktionen genRetrieverCode() und getRetrievalInterface() implementiert und exportiert werden. Der Typ des Rückgabewerts von getRetrievalInterface() kann frei gewählt werden und darf auch undefined sein. Das Verwenden der Bibliotheken im Ordner lib oder schon inkludierter npm-Pakete ist ohne Einschränkungen möglich. Zusätzliche npm-Pakete müssen zuvor mit npm installiert werden und können dann ebenfalls verwendet werden.

Beim Schreiben des generierten Codes in der Funktion `genRetrieverCode()` ist besonders darauf zu achten, dass keine Variablen verwendet werden, die in der Umgebung der generierten Methode `retrieveData()` nicht vorhanden sind. Die einzigen immer definierten Variablen innerhalb der Funktion sind

- der Parameter `retInt`, das Object, welches alle Laufzeitkomponenten der Module enthält, die von den `getRetrievalInterface()` Funktionen erhalten werden,
- die lokale Variable `out`, in dem die gelesenen Werte gespeichert werden,
- die Device-Instanz `this`,
- die globale Umgebungsvariable `env`,
- die globale Variable `__rootdir`, die den absoluten Pfad des Verzeichnisses enthält, in dem der NetBat-Server liegt,
- sowie sämtliche globalen Variablen von node.js.

Des Weiteren dürfen im erstellten Code keine lokalen Variablen deklariert und genutzt werden, die bereits von anderen Modulen oder vom Device-Konstruktor verwendet werden. Um dieses Problem zu umgehen wird es empfohlen, den gesamten generierten Programmcode eines Moduls mit einem Block, also mit geschwungenen Klammern, zu umgeben und für Variablen-deklarationen das Schlüsselwort `let` zu verwenden. Auf diese Weise werden bereits definierte Variablen innerhalb des Codes des Moduls lediglich überdeckt und im restlichen Bereich der `retrieveData()` Methode nicht beeinflusst. Außerdem können im Code des Moduls deklarierte Variablen nur dort gesehen werden und blockieren somit keine Variablennamen für die anderen Module.

Werden zusätzlich zu den zuvor genannten allgemeinen Variablen spezielle Variablen, Funktionen oder Objekte benötigt, können diese von `getRetrievalInterface()` zurückgegeben und somit an das Device übergeben werden. Der von `getRetrievalInterface()` zurückgegebene Wert landet im Parameter `retInt` der `retrieveData()` Methode unter dem Namen des Moduls.

Um bei der Erzeugung oder der Entfernung von Devices zusätzliche Aktionen durchzuführen, können die Funktionen `initDevice()` und `destructDevice()` exportiert werden. Diese werden im entsprechenden Fall mit dem Device als Parameter aufgerufen. Bei `initDevice()` wird die Konfiguration des Devices zusätzlich als zweiter Parameter übergeben.

## 5.6 Datenbankabstraktion

NetBat verwendet eine Datenbankabstraktion, damit nur ein kleiner Teil des Programmcodes, die Implementierung des Interfaces, datenbanksystemspezifische Elemente beinhaltet. Somit kann der Rest des Programms mit beliebigen Speichersystemen zusammenarbeiten, solange für diese eine Interfaceimplementierung vorhanden ist.

Diese Implementierungen werden im Ordner `database` in Form von JavaScript-Dateien abgelegt. Die Namen dieser Dateien sind gleichzeitig die intern verwendeten Namen der Datenbankinterfaces. Um die zu verwendende Implementierung auszuwählen, wird der jeweilige Name bei „`type`“ unter „`db`“ in der globalen Konfiguration angegeben. Wird keine Implementierung mit dem angegebenen Namen gefunden, wird ein Error mit der Fehlermeldung „Unknown database“ geworfen.

Eine Interfaceimplementierung muss eine Konstruktor-Funktion als Default-Export exportieren, die ein Object erzeugt, welches die weiter unten beschriebenen, benötigten Methoden enthält. Die Konfiguration des Datenbankinterfaces, also die in der globalen Konfiguration unter „`db`“ angegebenen Eigenschaften, werden in einem Object als Parameter übergeben. Für die Implementierung der Funktion wird das Schreiben einer Klasse mit entsprechendem Konstruktor empfohlen. Die erhaltene Instanz muss die folgenden Methoden besitzen:

**destruct()** ist eine noch ungenutzte Methode, welche bei der Beendigung der Datenbankverbindung mit diesem Interface, zum Beispiel durch das Beenden des Programms, aufgerufen werden soll. Hier werden beispielsweise offene Netzwerkverbindungen, Dateizugriffe und Hintergrundprozesse beendet und von dem Interface benötigte Ressourcen freigegeben.

**writeDataPoints()** ist eine asynchrone Methode, welche die übergebenen Datensätze in die Datenbank schreibt. Der übergebene Parameter ist ein Array aller Datensätze, welche jeweils die Eigenschaften `measurement`, `tags` und `fields` besitzen. `measurement` gibt den Namen der Tabelle bzw. des Measurements an, in die bzw. in das der jeweilige Datensatz geschrieben wird. `fields` und `tags` sind die Namen und Werte der Fields und Tags des Datensatzes. Wenn für Field- und Tag-Werte eine Typkonvertierung erfolgen muss, wird diese hier durchgeführt.

**addSchema()** fügt das übergebene Schema zur Datenbank hinzu. Hier werden beispielsweise Tabellen und Schemata in der Datenbank angelegt.

**addSchemas()** fügt die in einem Array übergebenen Schemata zur Datenbank hinzu. Die Funktion ist hier gleich wie in der Methode `addSchema()`.

**setSchemas()** löscht alle zuvor definierten Schemata und fügt die in einem Array übergebenen hinzu. Das Hinzufügen erfolgt gleich wie in der Methode **addSchema()**.

**initDownsampling()** ist eine asynchrone Methode. Sie startet das Downsampling der Datenbank. Der übergebene Parameter ist ein Array aller Downsampling-Stufen und der Alter der Daten, ab denen diese angewandt werden.

Es werden zwei Datenbankinterfaces mitgeliefert: „influxdb“ und „stdout“.

### 5.6.1 InfluxDB

Das Datenbanksystem InfluxDB ist, wie in Kapitel 4 beschrieben, die Standardspeichermethode von NetBat. Das hierfür implementierte Datenbankinterface heißt „influxdb“. Für die Kommunikation mit InfluxDB wird intern das npm-Paket „influx“ verwendet.

In der Konfiguration des Datenbankinterfaces müssen die Eigenschaften „host“ und „database“ angegeben werden. Die Eigenschaft „port“ ist optional. „host“ gibt die IP-Adresse oder die Domain der Datenbank an. „port“ gibt den Port an, auf dem die InfluxDB-Instanz erreichbar ist. Wird „port“ nicht angegeben, wird 8086 verwendet. „database“ gibt den Namen der Datenbank an, die verwendet werden soll. Hierbei ist zu beachten, dass für das Downsampling (vgl. Unterkapitel 4.3) eine weitere Datenbank benötigt wird, deren Name der angegebene Name mit dem Suffix „\_tmp“ ist.

Aufgrund von Vereinfachungen und der in Unterkapitel 4.3 beschriebenen Implementierung des Downsamplings sind bei der Verwendung dieses Interfaces folgende Einschränkungen zu beachten:

- Der Datentyp von Fields mit gleichem Namen muss in allen Measurements derselbe sein.
- Die Zusammenfassung des Downsamplings erfolgt für einen bestimmten Datentyp immer mit der gleichen Funktion. Fließkommazahlen werden arithmetisch gemittelt. Ganzzahlen werden arithmetisch gemittelt und auf die nächste ganze Zahl gerundet. Bei booleschen Werten und Strings wird der am häufigsten vorkommende Wert genommen.
- Die minimale Periodendauer des Datenabfragezyklus aller Devices ist eine Sekunde.

### 5.6.2 Konsolenausgabe

Das Datenbankinterface namens „stdout“ ist nicht in der Lage, Daten zu speichern, sonder für die Fehleranalyse gedacht. Aufgrund des fehlenden Speicherns ist kein Downsampling implementiert.

Alle an das Interface gesendeten Datensätze werden im Format „DBOut “<measurement name>”: [<tag1>: <value1>, <tag2>: <value2>, ...] <field1>: <value1>, <field2>: <value2>, ...“ in die Konsole geschrieben.

## 5.7 Globale Konfiguration

Die Funktion von NetBat kann über die globale Konfiguration gesteuert werden. Hier werden jene Einstellungen vorgenommen, die von den überwachten Geräten unabhängig sind und einmal für den gesamten Server definiert werden.

Die globale Konfiguration ist in mehrere Teile getrennt: das Environment, die allgemeine Konfiguration und die globale Grafana-Konfiguration. Diese besitzen jeweils eine lokale und eine mitgelieferte Variante. Die mitgelieferte gehört zum Softwarepaket von NetBat und enthält die Standardwerte der Konfiguration. Da hier alle Einstellungen gesetzt werden, können diese Dateien als Vorlage für die Lokalen verwendet werden, welche vom Anwender erstellt werden.

Zusätzlich zu den Konfigurationsdateien können Kommandozeilenparameter übergeben werden.

### 5.7.1 Kommandozeilenparameter

Der NetBat-Server akzeptiert die folgenden Kommandozeilenparameter.

`-h`

`--help`: Zeigt die Hilfe zu den Kommandozeilenparametern an.

`-v`

`--verbose`: Aktiviert die Ausgabe von zusätzlicher Information, den Debug-Modus.

`-c file`

`--config-file file`: Überschreibt den Pfad der allgemeinen Konfiguration mit *file*.

### 5.7.2 Environment-Konfiguration

Die Environment-Konfiguration (`env.json`) mit der mitgelieferten Variante `env.default.json` enthält Einstellungen, welche die Umgebung beschreiben, in der sich der NetBat-Server befindet, sodass die Funktionsweise entsprechend angepasst werden kann. Diese Einstellungen sind nicht für die Verwendung durch den Anwender gedacht, sondern dienen der Erleichterung des Entwicklungsprozesses.

Die Werte der mitgelieferten Konfiguration werden geladen und von den lokalen überschrieben. Darauf muss die lokale Datei weder vorhanden sein, noch alle Einstellungen angeben.

Die vorhandenen Einstellungen sind „environment“, „debug“ und „config“. Die Einstellung „environment“ gibt an, ob sich der Server in einer Entwicklungs- („development“), oder einer Produktionsumgebung („production“) befindet. Die Einstellung „debug“ gibt an, ob der Server immer im Debug-Modus starten soll. Die Einstellung „config“ gibt den Standardpfad der allgemeinen Konfiguration an.

Die geladene Environment-Konfiguration ist im Code als globale Variable namens `env` verfügbar.

### 5.7.3 Allgemeine Konfiguration

Die allgemeine Konfiguration enthält die vom Anwender gesetzten globalen Einstellungen, die die Funktionsweise des Servers steuern. Der Pfad dieser Datei kann in der Environment-Konfiguration und über den Kommandozeilenparameter „configfile“ abgeändert werden. Der Standardwert liegt hier bei config.ini.

Die mitgelieferte Konfiguration (config\_template.ini) wird nicht geladen, sondern dient als Vorlage für die lokale Datei, welche daher alle Einstellungen angeben muss. Module und Datenbankinterfaces können zusätzliche Einträge benötigen, die in der mitgelieferten Konfiguration nicht vorhanden sind. Ein Beispiel hierfür ist die „defaultCommunity“ für das SNMP-Modul.

Das Laden der Datei erfolgt mit dem npm-Paket „ini“. Das resultierende Object wird in die globale Variable env unter dem Key config abgelegt.

Die allgemeine Konfiguration ist aufgeteilt in die Sektionen „db“, „downsampling“ und „grafana“ sowie den keiner Sektion zugehörigen Einstellungen. Hinzu kommen etwaige von den Modulen benötigte Sektionen.

Außerhalb der Sektionen liegt der Key „localDirectory“, dessen Wert den Pfad des Verzeichnisses für die lokalen Templates, Devices und die lokale Grafana-Konfiguration angibt.

Die Sektion „db“ enthält die Einstellungen, die sich auf die Datenbank beziehen. Der Key „type“ gibt das zu verwendende Datenbankinterface an. Alle in dieser Sektion vorkommenden Einstellungen, inklusive „type“, werden als Parameter an den Konstruktor des Datenbankinterfaces übergeben. Das Datenbankinterface kann zusätzliche Einstellungen in der Sektion „db“ verlangen (siehe Unterkapitel 5.6).

Die Sektion „downsampling“ steuert die Stufen des Downsamplings. Die sich hier befindenden Keys „durations“ und „steps“ enthalten Listen an Zeitspannen, welche die gleiche Länge haben müssen. Jede Stufe setzt sich aus den n-ten Elementen der beiden Keys zusammen, wobei die Zeitspanne aus „durations“ das Mindestalter der Daten angibt, ab dem die jeweilige Stufe angewandt wird, und die Zeitspanne aus „steps“ die Schrittweite der komprimierten Daten. Die Stufen werden in steigender Reihenfolge der beiden Werte angegeben. Eine Zeitspanne besteht aus einer positiven Ganz- oder Fließkommazahl und einem optionalen Einheitssuffix. Zur Verfügung stehen die Einheiten Sekunde („s“), Minute („m“), Stunde („h“) und Tag („d“), wobei eine Zahl ohne Einheitssuffix als Anzahl von Sekunden gedeutet wird. Jede Zeitspanne wird auf Sekunden gerundet. Die kleinstmögliche Zeitspanne ist eine Sekunde.

Die Sektion „grafana“ enthält die zur Verbindung mit dem Grafana-Server notwendigen Informationen. Der Key „url“ gibt die URL an, unter der die HTTP-API zu erreichen ist. Der Key „apiKey“ enthält den API-Token (siehe [6]), der für den Zugriff auf den Grafana-Server verwendet werden soll. Der Key „enable“ gibt an, ob die automatische Erzeugung einer grafischen Visualisierung in Grafana aktiviert ist. Wird dieser auf `false` geändert, sind sämtliche in Verbindung mit Grafana stehenden Features von NetBat deaktiviert. Dies kann verwendet werden, um NetBat ohne einen Grafana-Server zu betreiben und die Daten in der Datenbank anders oder nicht zu visualisieren.

### 5.7.4 Globale Grafana-Konfiguration

Die globale Grafana-Konfiguration enthält einstellbare und für alle Devices gleiche Grafana-Konfigurations-Blöcke sowie „Device-lose“ allgemeine Ordner, Panels und Dashboards.

Ihre mitgelieferte Variante findet sich in der Datei `grafana_global.json`. Die lokale Datei befindet sich mit gleichem Namen im lokalen Ordner, der in der allgemeinen Konfiguration angegeben ist. Die lokale Datei kann anstelle von einer JSON-Datei im Format YAML mit dem Namen `grafana_global.yml` abgelegt werden.

Die Eigenschaft `defaultAlertNotifications` ist ein Array, welches in allen Panels, bei denen die Eigenschaft „alert“ vorhanden ist, unter „`alert.notifications`“ eingefügt wird. Es gibt die „Notification Channel“ an, über welche die Alert-Nachricht versendet wird. Der Standardwert, ein leeres Array, wird von der lokalen Datei überschrieben. Jeder „Notification Channel“ wird durch ein Object angegeben, dessen einzige Eigenschaft `uid` die UID des „Notification Channels“ enthält.

Die Eigenschaften `folders`, `panels` und `dashboards` bilden eine Pseudo-Device-Grafana-Konfiguration. Im Gegensatz zu einer Grafana-Konfiguration in einem Device enthält diese statische, globale Grafana-Elemente, die genau einmal erzeugt werden. Sie wird von der `grafana` Bibliothek vor allen Device-Grafana-Konfigurationen geladen. Diese statische und globale Konfiguration kann beispielsweise genutzt werden, um Ordner und Dashboards zu erstellen, die von den Devices erweiternd befüllt werden. Da diese Konfigurationen statisch sind und nicht generiert werden, können hier keine Variablen, ValueGenerator oder andere zu verarbeitende Objekte eingesetzt werden. Der Standardwert und der lokale werden hier wie die Device-Grafana-Konfiguration eines Templates und eines erbenden Geräts zusammengefügt (siehe Unterkapitel 5.8.3).

Die Eigenschaft `pingTemplate` enthält eine Device-Grafana-Konfiguration, die in jedem Device, in dem der automatische Ping aktiviert ist, hinzugefügt wird. Dieses Template wird an die Position des ersten Basistemplates gestellt und kann somit von allen Templates in der Vererbungskette und dem Device verändert und erweitert werden. Auch hier werden der Standardwert und der lokale wie die Device-Grafana-Konfiguration eines Templates und eines erbenden Geräts zusammengefügt (siehe Unterkapitel 5.8.3).

## 5.8 Device-Konfiguration

Die Konfiguration der Datenabfrage, -verarbeitung, -speicherung und -anzeige im Bezug auf einzelne Geräte ist mit Device-Konfigurationen aufgebaut. Diese enthalten jeweils die gesamte Konfiguration für ein Device. Somit müssen für das Hinzufügen und Entfernen eines Geräts nur an einer Stelle Änderungen durchgeführt werden.

Die Device-Konfigurationen werden mit einer rekursiven Suche im Unterordner `devices` des in der allgemeinen Konfiguration angegebenen lokalen Ordners geladen. Hierbei werden die Dateiformate JSON mit der Endung „.json“ und YAML mit der Endung „.yml“ unterstützt.

Eine solche Device-Konfigurationsdatei kann ein oder mehrere Device-Konfigurationen beinhalten. Ist die äußerste Struktur der Datei ein Object, so wird es als einzelne Device-Konfiguration geladen. Ist sie ein Array, wird jedes Element als eigene Device-Konfiguration geladen. Dies ermöglicht die Gruppierung von Geräten.

Innerhalb der Device-Konfiguration werden die Verbindungen zwischen Datenquellen und Datensentken mit Variablen realisiert. Die Namen dieser Variablen sind, wenn nicht anders angegeben, innerhalb des Devices einzigartig. Eine Variable kann also nur an einer Stelle definiert werden. Durch diese Definition erhält die Variable ihren Wert. Dieser Wert kann alle in JavaScript verfügbaren Werte und Datentypen annehmen. Der Name einer Variable kann nur alphanumerische Zeichen und Unterstriche beinhalten und muss mit einem Buchstaben oder einem Unterstrich beginnen. Eine Referenz auf eine Variable wird durch ein dem Namen vorangestelltes Dollarzeichen angegeben.

Die Variablen `time_ms`, `address`, `hostname` und `devicegroup` sind vordefinierte Variablen, welche in jedem Device automatisch definiert sind. `time_ms` gibt die Zeit zum Zeitpunkt der Verwendung in ms seit der UNIX Epoch (1.1.1970 00:00:00 UTC) [15], die anderen drei die gleichnamigen Eigenschaften des Devices wieder.

Ist der automatische Ping für das Device aktiviert, werden für diesen die beiden Variablen `rtt` und `alive` definiert. Für die Beschreibung von deren Inhalt siehe Unterkapitel 5.5.1.

*Alle Eigenschaften einer Device-Konfiguration bis auf eine sind optional.*

Eine Device-Konfiguration ist folgendermaßen aufgebaut.

---

```
{
    "extends"?: <string|string []>,
    "address": <ipv4 address|ipv6 address|domain>,
    "hostname"?: <string>,
    "samplePeriod"?: <number>,
    "disablePing"?: <boolean>,
    "db"?: <device database config>,
    "memorize"?: {
        <memorized name>: <variable name>
    },
    "grafana"?: <device grafana config>
    <module name>?: <module config>
}
```

---

Die Eigenschaft `extends` gibt ein Template oder ein Array aus Templates an, von denen geerbt werden soll. Dies ist im folgenden Unterkapitel 5.8.1 näher beschrieben.

Die einzige benötigte Eigenschaft `address` gibt die Adresse des Geräts an. Hier kann eine IPv4-Adresse, eine IPv6-Adresse oder eine Domain verwendet werden.

Die Eigenschaft `hostname` enthält einen Namen, der dem Device vergeben wird. Dieser kann, muss aber nicht dem Hostnamen des Geräts entsprechen. Der Name wird von manchen Debug-Ausgaben und Fehlermeldungen angegeben (siehe Unterkapitel 5.9) und kann von Modulen und der grafischen Darstellung verwendet werden. Der Defaultwert von `hostname` ist ein leerer String.

`samplePeriod` gibt den Zeitabstand zwischen den Datenabfragen in ms an. Dieser Wert muss eine Ganzzahl sein und ist defaultmäßig 1000.

`disablePing` deaktiviert den automatischen Ping für das Device. Anzumerken ist, dass Devices in einem Abfragezyklus, in dem ihr automatischer Ping fehlschlägt, keine weiteren Anfragen mehr tätigen.

`db` enthält die Device-Datenbank-Konfiguration. Hier wird festgelegt, wie die erhaltenen Daten verarbeitet und abgespeichert werden (näheres hierzu im übernächsten Unterkapitel 5.8.2).

`memorize` definiert Variablen, die die Werte anderer Variablen aus dem vorherigen Zyklus speichern und somit im folgenden Zyklus zur Verfügung stellen. Die Konfiguration ist hier ein Object, dessen Keys die Namen der neuen Variablen und dessen Values die Namen der zu erinnernden Variablen sind.

`grafana` enthält die Device-Grafana-Konfiguration. Hier wird festgelegt, wie die erhaltenen Daten verarbeitet und abgespeichert werden (näheres hierzu im Unterkapitel 5.8.3).

Für jedes geladene Modul gibt es eine weitere optionale Eigenschaft in der Device-Konfiguration, die den gleichen Namen trägt wie das Modul. In diesen Konfigurationen werden die Variablen definiert, welche die ausgelesenen Daten enthalten. Die Struktur dieser Konfigurationen wird vom Modul bestimmt. Die Konfigurationen der mitgelieferten Module werden in Unterkapitel 5.5 beschrieben.

### 5.8.1 Templates

Es ist möglich, Vorlagen, sogenannte Templates, für Devices zu erstellen. Diese Funktion ermöglicht es, Konfigurationen wiederzuverwenden, ohne sie mehrfach abzuschreiben. Auf diese Weise müssen Fehler nur an einer Stelle ausgebessert werden. Außerdem verkürzt diese Auslagerung die Device-Konfigurationsdateien, womit Übersicht gewonnen wird.

Templates werden aus zwei Verzeichnissen geladen, die mitgelieferten aus dem Ordner `templates` und die lokalen aus dem Unterordner `templates` des in der allgemeinen Konfiguration angegebenen lokalen Ordners. Es werden die Dateiformate JSON mit der Endung „`.json`“ und YAML mit der Endung „`.yml`“ unterstützt. Anders als bei den Device-Konfigurationsdateien kann nur ein Template pro Datei definiert werden. Die äußerste Struktur der Dateien muss ein Object sein.

Jedes Template besitzt einen Bezeichner. Dieser gibt den Pfad der Template-Datei relativ zu dem Ordner, in dem die rekursive Suche gestartet wurde, an und endet mit dem Namen der Datei, abzüglich ihrer Dateierweiterung. Folglich gibt es für jeden Bezeichner vier mögliche Dateien, eine mit der Endung „`.json`“ und eine mit der Endung „`.yml`“ in jedem der beiden zuvor genannten Ordner. Wird ein Template in beiden Ordner definiert, so ersetzt das Template im lokalen Order das aus dem mitgelieferten. Im selben Ordner dürfen nie zwei gleichnamige Templates mit unterschiedlichen Dateiendungen existieren.

Templates besitzen den gleichen Aufbau wie Device-Konfigurationen. Dies inkludiert identifizierende Eigenschaften, wie `address` oder `hostname`.

Ein Template oder ein Device kann von einem anderen Template erben, indem es dessen Bezeichner unter `extends` angibt. Ist `extends` ein einzelner String, wird von dem mit diesem String bezeichneten Template geerbt. Wird ein Array mehrerer Templates angegeben, so wird von allen geerbt, wobei jene mit niedrigerem Index im Fall von Überschneidungen als grundlegender angesehen werden. Das Ergebnis gleicht also dem, welches entstehen würde, wenn die Templates mit höherem Index jeweils von den vorherigen und das Device schließlich vom letzten Template erben würde.

Das Erben wird durch das rekursive Kopieren aller Eigenschaften der erbenden Konfiguration realisiert. Das heißt, dass elementare Werte (Zahlen, Strings, Booleans, null) jene des Basistemplates überschreiben und Arrays und Objects alle ihre Elemente bzw. Eigenschaften einzeln betrachten. Hiervon ausgeschlossen ist die Device-Grafana-Konfiguration. Diese besitzt eine eigene Vererbungsvorschrift, die im Unterkapitel 5.8.3 beschrieben wird.

## 5.8.2 Datenbank

Die Device-Datenbank-Konfiguration definiert, welche Werte wie in der Datenbank abgespeichert werden. Hier werden die Datenbankschemata festgelegt.

Eine Device-Datenbank-Konfiguration besteht aus einem Object mit der folgenden Struktur.

```
{
  <measurement name>: {
    "unwrap"??: {
      <variable name>: (<variable name>|<variable reference>|{
        "index": <variable name>|<variable reference>,
        "size"??: <variable name>,
        "top"??: <variable name>,
        "exclude"??: <any []>,
        "include"??: <any []>,
        "min"??: <number|string>,
        "max"??: <number|string>,
        "step"??: <number>
      }) []
    },
    "tags": {
      <tag name>: {
        "type": <db type>,
        "value": <any>,
        "default"??: <any>
      }
    },
    "fields": {
      <field name>: {
        "type": <db type>,
        "value": <any>,
        "default"??: <any>
      }
    }
  }
}
```

Die Device-Datenbank-Konfiguration setzt sich aus Tabellen bzw. Measurements zusammen, die im Object ihren Namen als Key verwenden.

Die Measurement-Eigenschaft `unwrap` gibt die Unwrapper der Tabelle an. Hierbei wird ein Unwrapper einer Variable zugeordnet, deren Name als Key verwendet wird. Ein Unwrapper ist die Beschreibung einer Schleife, mit der der von dem Measurement beschriebene Datensatz für mehrere Elemente eines Arrays oder eines Objects generiert wird. Es werden dabei nur jene Indizes des Arrays bzw. Keys des Objects verwendet, welche die angegebenen Bedingungen erfüllen. Wird die als Quelle des Arrays bzw. des Object angegebene Variable innerhalb des Measurement verwendet, zeigt sie dort nicht auf ihren ursprünglichen Wert, sondern auf das vom Unwrapper selektierte Element. Alle vom Unwrapper definierten Variablen können nur innerhalb des Measurement verwendet werden.

Der Wert des Unwrappers kann ein Variablename, eine Variablenreferenz und ein Object sein. Ist der Wert ein Variablename, definiert er die Variable, mit der der selektierte Index referenziert werden kann. Ist der Wert eine Variablenreferenz, wird keine eigene Schleife erstellt, sondern der Wert der angegebenen Variable als Index benutzt. Diese Variable kann auch ein Index eines anderen Unwrappers sein. Ist der Wert ein Object, können weitere Eigenschaften der Schleife spezifiziert werden. Die benötigte Eigenschaft `index` erfüllt die selbe Funktion, wie die Angabe eines Strings als Wert des Unwrappers. Die Eigenschaft `size` definiert eine zusätzliche Variable, die die Anzahl an zu durchlaufenden Indizes bzw. Keys enthält. Die Eigenschaft `top` definiert eine zusätzliche Variable, die den größten Index bzw. Key enthält. Die Eigenschaften `exclude`, `min`, `max` und `step` geben optionale Bedingung an, die ein Index bzw. ein Key erfüllen muss, um verwendet zu werden. Indizes bzw. Keys, die kleiner als `min`, größer als `max` oder in `exclude` enthalten sind, werden übersprungen. `step` kann nur für Zahlen verwendet werden und setzt die Bedingung, dass der Abstand zu `min` - ist `min` nicht gesetzt wird 0 verwendet - ein Vielfaches von `step` sein muss. Die Eigenschaft `include` gibt ein Array an Indizes bzw. Keys an, die immer durchlaufen werden. Diese müssen weder die Bedingungen erfüllen noch im Ausgangsobjekt definiert sein.

Die Eigenschaften `tags` und `fields` geben die Tags und Fields der Datensätze an und sind identisch aufgebaut. Die Einträge besitzen den Namen des Tags bzw. Fields als Key und die Beschreibung von dessen Wert als Value. Ein solcher Wert wird durch die Eigenschaften `type`, `value` und `optional default` beschrieben. Die Eigenschaft `type` gibt den Datentyp des Werts an. Die Datentypen „`integer`“, „`boolean`“, „`string`“ und „`float`“ stehen zur Verfügung. `default` gibt einen Standardwert an, der verwendet wird, wenn der ermittelte Wert `undefined`, `null`, `NaN`, `Infinite` oder `-Infinite` ist. Tritt dieser Fall in einem Zyklus ein und ist `default` nicht festgelegt, wird der gesamte betroffene Datensatz in diesem Zyklus verworfen und nicht an das Datenbankinterface übergeben. Dies hat keine Auswirkungen auf die Datensätze anderer Measurements und anderer Indizes der Unwrapper.

Die Eigenschaft `value` gibt den Wert an, den der Tag bzw. das Field erhält. Hier können Konstanten, Variablenreferenzen und sogenannte „ValueGenerator“ verwendet werden. Konstanten können Arrays, Zahlen, Strings, `true`, `false` und `null` sein. Strings, die mit zwei Dollarzeichen beginnen, werden als normale Strings behandelt, wobei das erste Dollarzeichen entfernt wird. Im Fall eines Arrays können alle Elemente des Arrays wiederum Konstanten, Variablenreferenzen und ValueGenerator sein. Ein ValueGenerator kann einen vordefinierten Satz an Funktionen verwenden, um Werte aus Variablen und Konstanten zu berechnen. Diese werden mit einem Object definiert, das die Eigenschaft `function` besitzt. Diese gibt die zu verwendende Funktion an. Die Funktionsparameter werden als weitere Eigenschaften übergeben. Welche Parameter benötigt werden oder optional angegeben werden können sowie deren erlaubte Datentypen hängt von der verwendeten Funktion ab. Nicht konstante Parameter können Variablenreferenzen und ValueGenerator enthalten. Die folgenden Funktionen sind verfügbar:

**add:** Addiert die beiden Parameter `value` und `value2`. Die Funktionsweise gleicht dem Operator „`+`“ in JavaScript (siehe [15]).

**subtract:** Subtrahiert `value2` von `value`. Die Funktionsweise gleicht dem Operator „`-`“ in JavaScript (siehe [15]).

**multiply:** Multipliziert die beiden Parameter `value` und `value2`. Die Funktionsweise gleicht dem Operator „`*`“ in JavaScript (siehe [15]).

**divide:** Dividiert `value` durch `value2`. Die Funktionsweise gleicht dem Operator „`/`“ in JavaScript (siehe [15]).

**sqrt:** Errechnet die Quadratwurzel des Parameters `value`. Die Funktionsweise entspricht der JavaScript Funktion `Math.sqrt()` (siehe [15]).

**power:** Potenziert den Basis `value` mit dem Exponent `value2`. Die Funktionsweise entspricht der JavaScript Funktion `Math.pow()` (siehe [15]).

**integrate:** Integriert den übergebenen Parameter `value` im Zeitverlauf. Wird ein String übergeben, der einen Zahlenwert darstellt, wird dieser in eine Zahl konvertiert. Der Wert `false` wird als 0, `true` als 1 interpretiert. Der optionale konstante Parameter `clock` gibt an, welche Quelle für die mit den Werten multiplizierten Zeitdifferenzen verwendet wird. Mögliche Werte hierfür sind „`none`“ und „`internal`“, wobei Letzteres verwendet wird, wenn `clock` nicht angegeben wird. Bei Angabe des Werts „`internal`“ wird die Zeit seit dem letzten Zyklus genutzt, bei „`none`“ wird die Zeitabhängigkeit deaktiviert, sodass die einkommenden Werte unverändert aufaddiert werden. Die optionalen Parameter `min` und `max` geben die untere und obere Grenze des Bereichs an, in dem sich das Ergebnis bewegen soll. Ist `min` nicht angegeben, wird 0 als Defaultwert genommen. `max` muss immer größer als `min` sein. Ist `max` angegeben,

wird der Ausgangswert beim Überschreiten der Grenzen durch Addition oder Subtraktion von Vielfachen der Differenz zwischen `min` und `max` innerhalb des Bereichs gehalten. Als Startwert wird der Wert mit dem kleinsten Betrag verwendet, der sich innerhalb der Grenzen befindet.

**difference:** Differenziert den übergebenen Parameter `value` im Zeitverlauf. Wird ein String übergeben, der einen Zahlenwert darstellt, wird dieser in eine Zahl konvertiert. Der Wert `false` wird als 0, `true` als 1 interpretiert. Der optionale konstante Parameter `clock` gibt an, welche Quelle für die die Werte dividierenden Zeitdifferenzen verwendet wird. Mögliche Werte hierfür sind „`none`“ und „`internal`“, wobei Letzteres verwendet wird, wenn `clock` nicht angegeben wird. Bei Angabe des Werts „`internal`“ wird die Zeit seit dem letzten Zyklus genutzt, bei „`none`“ wird die Zeitabhängigkeit deaktiviert, sodass die Differenzen der einkommenden Werte unverändert ausgegeben werden. Die optionalen Parameter `min` und `max` geben die untere und obere Grenze der Änderung des Eingangswertes zwischen zwei Zyklen an. Dies wird benötigt, um Überläufe bei einem beschränkten Eingangswert zu erkennen. Die Differenz von `min` und `max` muss die Intervalllänge des Eingangswertebereichs sein. Ist `min` nicht angegeben, wird 0 als Defaultwert genommen. `max` muss immer größer als `min` sein. Ist `max` angegeben, wird die Änderung des Eingangswerts beim Überschreiten der Grenzen durch Addition oder Subtraktion von Vielfachen der Differenz zwischen `min` und `max` innerhalb des Bereichs gehalten. Im ersten Zyklus wird `NaN` zurückgegeben.

**log:** Errechnet den Logarithmus zur Basis `base` des Wertes `value`. Der natürliche Logarithmus wird durch die Basis „`e`“ angegeben. Wird keine Basis definiert, wird der natürliche Logarithmus verwendet. Die Funktionsweise bezüglich der Datentypkonvertierung beider Parameter entspricht der JavaScript Funktion `Math.log()` (siehe [15]). Wird eine negative Zahl in einem der beiden Parameter, 1 in beiden, `Infinity` in beiden, oder `Infinity` in `value` und 0 in `base` übergeben, ist das Ergebnis `NaN`.

**max:** Erhält ein Array oder ein Object als Parameter `values` und gibt den größten enthaltenen Wert zurück. Bei einem Object werden die Values betrachtet. Die Funktionsweise bezüglich der Datentypkonvertierung der Werte entspricht der JavaScript Funktion `Math.max()` (siehe [15]).

**min:** Erhält ein Array oder ein Object als Parameter `values` und gibt den kleinsten enthaltenen Wert zurück. Bei einem Object werden die Values betrachtet. Die Funktionsweise bezüglich der Datentypkonvertierung der Werte entspricht der JavaScript Funktion `Math.min()` (siehe [15]).

**toString:** Wandelt den im Parameter `value` übergebenen Wert in einen String um. Die Funktionsweise entspricht der JavaScript Funktion `String()` (siehe [15]).

**tonumber:** Wandelt den im Parameter `value` übergebenen Wert in einen String um. Die Funktionsweise entspricht der JavaScript Funktion `Number()` (siehe [15]).

**floor:** Gibt die größte Ganzzahl zurück, die kleiner oder gleich dem Parameter value ist. Die Funktionsweise entspricht der JavaScript Funktion `Math.floor()` (siehe [15]).

**floor:** Gibt die kleinste Ganzzahl zurück, die größer oder gleich dem Parameter value ist. Die Funktionsweise entspricht der JavaScript Funktion `Math.ceil()` (siehe [15]).

**count:** Zählt die Anzahl an übergebenen Elementen im Parameter value. Ist value ein Array, wird die Länge des Arrays zurückgegeben. Ist value ein Object, wird die Anzahl an Keys zurückgegeben. Ist value undefined, null, NaN, Infinite oder -Infinite, wird 0 zurückgegeben. Tritt keiner dieser Fälle ein, wird 1 zurückgegeben.

**element:** Gibt ein Element des im Parameter value übergebenen Arrays oder Objects zurück. Der Parameter index gibt den Index bzw. den Key an.

**replace:** Ersetzt alle mit dem im Parameter pattern angegebenen Regex übereinstimmende Teile in input durch replacement. Alle Parameter, die keine Strings sind, werden wie mit „tostring“ in Strings konvertiert. In replacement können die Platzhalter des Ersatzstrings der JavaScript Funktion `String.prototype.replace()` verwendet werden.

**concat:** Hängt die im Parameter values in einem Array übergebenen Arrays in der übergebenen Reihenfolge mit der JavaScript Funktion `Array.prototype.concat()` aneinander und gibt das resultierende Array zurück.

**assign:** Kopiert die Werte aller aufzählbaren, eigenen Eigenschaften von einem oder mehreren Objects in ein gemeinsames Object. Diese Objects werden in einem Array im Parameter values übergeben. Ist das erste Element ein Array, ist das gemeinsame Object auch ein Array. Die Funktionsweise entspricht der JavaScript Funktion `Object.assign()` (siehe [15]) mit der Ausnahme, dass der Wert des Ziel-Objects nicht verändert, sondern das Ergebnis in einem neuen Object zurückgegeben wird.

**arraymap:** Evaluiert den Wert des Parameters value für jedes Element des Eingangsobjekts im Parameter input und gibt die Ergebnisse in einem Array zurück. Es werden neue Variablen definiert, deren Namen in den konstanten Parametern keyname und valuename als String übergeben werden. Diese sind innerhalb des Ausdrucks in value verfügbar und enthalten den Key des Elements in der Variable in keyname und den Wert des Elements in der Variable in valuename. Das Eingangsobjekt kann ein Array und ein Object sein. Handelt es sich um ein Array, entspricht die Reihenfolge der Ergebnisse der Reihenfolge, der für diese verwendeten Elemente im Array. Bei einem Object ist keine Reihenfolge definiert.

**objectmap:** Bildet Key-Value-Paare durch die Evaluierung der Parameter key für die Keys und value für die Values für jedes Element des Eingangsobjekts im Parameter input. Diese Key-Value-Paare werden in einem Object abgelegt. Es werden neue Variablen definiert, deren Namen in den konstanten Parametern keyname und valuename als String übergeben werden. Diese sind innerhalb der Ausdrücke in key und value verfügbar und enthalten den Key des Elements in der Variable in keyname und den Wert des Elements in der Variable in valuename. Das Eingangsobjekt kann ein Array und ein Object sein.

**if:** Prüft, ob der Wert des Parameters test „truthy“ ist, und gibt, wenn das der Fall ist, den Wert des Parameters then, ansonsten den des optionalen Parameters else zurück. Wird kein Parameters else angegeben, wird er mit dem Wert undefined belegt.

**equals:** Prüft, ob die Werte der Parameter value und value2 gleich sind, und gibt das Ergebnis als booleschen Wert zurück. Führt die gleichen Typkonvertierungen durch wie der JavaScript Operator „==“, es sei denn der optionale konstante Parameter strict ist true, dann wird auf strikte Gleichheit (entspricht dem Operator „==“) geprüft (siehe [15] für eine Beschreibung der Typkonvertierungen des Gleichheitsoperators). Wird strict nicht angegeben, wird er als false angenommen.

**greater:** Prüft, ob der Wert des Parameters value größer ist als der des Parameters value2, und gibt das Ergebnis als booleschen Wert zurück. Der Vergleich verhält sich wie der Größer-als-Operator „>“ in JavaScript (siehe [15]).

**smaller:** Prüft, ob der Wert des Parameters value kleiner ist als der des Parameters value2, und gibt das Ergebnis als booleschen Wert zurück. Der Vergleich verhält sich wie der Kleiner-als-Operator „<“ in JavaScript (siehe [15]).

**truthy:** Prüft, ob der Wert des Parameters value „truthy“ ist, und gibt das Ergebnis als booleschen Wert zurück. Die Werte false, 0, 0n, "", null, undefined und NaN sind „falsy“, alle anderen „truthy“. [15]

**exists:** Prüft, ob der Wert der Parameters value von NetBat als existent gewertet wird. Die nicht existenten Werte sind undefined, null, NaN, Infinity und -Infinity. Diese Werte sorgen, wie zuvor beschrieben, beim Auftreten als Wert eines Tags oder Fields, für das Verwerfen des Datensatzes, wenn kein Default-Wert gesetzt ist.

**not:** Prüft, ob der Wert des Parameters value „truthy“ ist, und gibt das Gegenteil des Ergebnisses als booleschen Wert zurück. Die Werte false, 0, 0n, "", null, undefined und NaN sind „falsy“, alle anderen „truthy“. [15] Diese Funktion ist das Gegenteil der Funktion „truthy“.

**and:** Prüft, ob alle in einem Array im Parameter values angegebenen Werte „truthy“ sind, und gibt das Ergebnis als booleschen Wert zurück. Es kann auch ein Object als Parameter übergeben werden. In diesem Fall werden die Values des Objects geprüft.

**or:** Prüft, ob mindestens einer der in einem Array im Parameter values angegebenen Werte „truthy“ ist, und gibt das Ergebnis als booleschen Wert zurück. Es kann auch ein Object als Parameter übergeben werden. In diesem Fall werden die Values des Objects geprüft.

### 5.8.3 Grafana

Die Device-Grafana-Konfiguration gibt an, welche Visualisierungselemente ein Device automatisch auf dem Grafana-Server für die grafische Darstellung seiner Daten erstellt, wo sich diese befinden und welche Eigenschaften sie besitzen.

Eine Device-Grafana-Konfiguration ist folgendermaßen aufgebaut.

---

```
{
  "folders"??: {
    "uid": <folder uid>
    "title"??: <string>
  }[],
  "panels"??: {
    <panel uid>: <panel>
  },
  "dashboards"??: {
    "uid": <dashboard uid>,
    "title"??: <string>,
    "folder"??: <folder uid>,
    "panels": (<panel uid>|{
      "uid": <row uid>,
      "rowTitle"??: <string>,
      "panels": <panel uid>[],
      "collapsed"??: <boolean>
    })[]
  }[]
}
```

---

Die optionale Eigenschaft `folders` ist ein Array, welches Ordnerbeschreibungen enthält. Eine solche Beschreibung setzt sich aus einer UID `uid` und einem Titel `titel` zusammen. Die Ordner-UID wird für Grafana auf die in Unterkapitel 5.4.4 beschriebenen Weise angepasst.

Die optionale Eigenschaft `panels` ist ein Object, welches Panels enthält. Der Key, unter dem ein Panel im Object zu finden ist, ist dessen UID. Bei einem Panel handelt es sich um ein fast vollständiges JSON-Modell eines Panels aus Grafana. Die Panel-Eigenschaft `id` wird nicht gesetzt. Die Panel-Eigenschaft `alert.notifications` wird von NetBat im Standardfall auf den in der globalen Grafana-Konfiguration

angegebenen Wert gesetzt, kann aber in der Device-Grafana-Konfiguration spezifisch festgelegt werden. Die Panel-Eigenschaft `gridPos` enthält die Untereigenschaften `w`, `h`, `x` und `y` für die Breite, Höhe, X- und Y-Koordinate des Panels. Die Y-Koordinate wird immer während des Ladevorgangs bestimmt. Die anderen drei Werte werden mit Standardwerten befüllt, können aber in der Device-Grafana-Konfiguration spezifisch festgelegt werden.

Die Eigenschaft `dashboards` ist ein Object, welches die vom Device beschriebenen Dashboards enthält. Jedes Dashboard hat eine UID, die mit der Dashboard-Eigenschaft `uid` festgelegt wird. Die Dashboard-UID wird für Grafana auf die in Unterkapitel 5.4.4 beschriebenen Weise angepasst. Die optionalen Eigenschaften `title` und `folder` geben den Titel und die UID des Ordners an, in den das Dashboard eingeordnet werden soll. Die benötigte Eigenschaft `panels` enthält ein Array aller Panels, die in das Dashboard hinzugefügt werden. Sie werden mit ihrer UID angegeben. Die Panels werden hier in der selben Reihenfolge angegeben, in der sie in Grafana von oben beginnend angezeigt werden.

Zusätzlich zu Panels können im Array `panels` eines Dashboards Rows definiert werden. Diese stellen einklappbare Gruppen von Panels da. Diese Rows besitzen die Eigenschaften `uid`, `rowTitle`, `panels` und `collapsed`. Die Eigenschaft `uid` gibt die UID einer Row an, die optionale Eigenschaft `rowTitle` ihren Namen. `panels` enthält ein Array aller Panels, die sich in einer Row befinden. Sie werden mit ihrer UID in der selben Reihenfolge angegeben, in der sie in Grafana von oben beginnend angezeigt werden. Die optionale Eigenschaft `collapsed` gibt an, ob die Row sich im eingeklappten Zustand befindet und wird, wenn sie nicht angegeben wird, mit dem Standardwert `true` angenommen.

Alle Values innerhalb der Ordner, Panels, Rows und Dashboards mit Ausnahme der Panel-Listen (Eigenschaft `panels`) und ihrer Elemente können durch Generatoren ersetzt werden, die das Anpassen einer Konfiguration über die abgefragten Variablen erlaubt. Ein Generator ist ein Object, welches den Key „\$“ besitzt und bei der Verarbeitung mit dem Wert, welcher sich unter dem Key „\$“ befindet, ersetzt wird. Dieser Wert kann wie die Tag- und Field-Werte in der Device-Datenbank-Konfiguration eine Konstante, eine Variablenreferenz oder ein ValueGenerator sein. Eine Besonderheit der Generatoren in der Device-Grafana-Konfiguration ist, dass eine Konstante vom Typ String Platzhalter beinhalten kann, welche durch eine Begrenzung mit dem Zeichen „\$“ gekennzeichnet werden. Diese Platzhalter werden mit dem Wert der zwischen den Dollarzeichen genannten Variable ersetzt. Dieser wird hierfür in einen String konvertiert. Um ein einzelnes Dollarzeichen in den String einzufügen, kann „\$\$“ geschrieben werden.

Wenn Device-Grafana-Konfigurationen vererbt werden, wird die Vereinigung der Konfigurationen anders durchgeführt als beim Rest der Device-Konfiguration. Die Ordner- und Dashboard-Arrays werden aneinandergehängt, so dass die Elemente der spezielleren Konfiguration nach denen des Basistemplates kommen. Bei den Panels werden alle vorhandenen Panels mit ihren UIDs in ein Object zusammengefasst, wobei im Fall der Mehrfachdefinition einer UID das Panel aus der speziellsten Konfiguration genommen wird.

Während der Generierung der Device-Grafana-Konfigurationen durch die `generateGrafana()` Methode des Devices werden die Panel-UIDs in den Dashboards und Rows durch die Panels ersetzt. Hierbei ist es möglich, dass mehrere Kopien eines Panels erzeugt werden. Ab diesem Schritt gibt es keine separaten Panel-Definitionen mehr, da sie in den Dashboards integriert sind.

Sobald alle Device-Grafana-Konfigurationen generiert sind, werden sie von der Bibliothek `grafana` zu einer gesammelten Konfiguration kombiniert und an den Grafana-Server gesendet. Bei dieser Kombination werden Mehrfachdefinitionen folgendermaßen zusammengefasst:

Jeder im Ordner-Array angegebenen UID wird der angegebene Titel zugewiesen. Wird der Titel eines Orders mehrfach definiert, wird die im Array letzte Angabe verwendet. Gibt es keine Definition des Titels für eine Ordner-UID, wird „Untitled“ verwendet. Diese Ordner werden dann zusammen mit den Titeln an den Grafana-Server übermittelt.

Dashboards mit gleicher UID werden zusammengefasst, indem der Titel und die Ordner-UID jeweils auf den im Array letzten angegebenen Wert gesetzt werden. Gibt es keine Definition des Titels für eine Dashboards-UID, wird „Untitled“ verwendet. Gibt es keine Definition der Ordner-UID, wird das Dashboard in keinen Ordner eingeordnet. Die Panels werden aneinander gehängt, wobei hier die Reihenfolge des Dashboard-Arrays eingehalten wird.

In den kombinierten Dashboards werden Rows mit gleichen UIDs verbunden, indem der Titel und der Zusammenklappstatus jeweils auf den im Array letzten angegebenen Wert gesetzt werden. Gibt es keine Definition des Titels für eine Row-UID, wird „Untitled“ verwendet. Gibt es keine Definition des Zusammenklappstatus, wird `true` verwendet. Die Panels werden aneinander gehängt, wobei hier die Reihenfolge des Arrays, in dem die Rows liegen, eingehalten wird.

## 5.9 Fehlermeldungen

Die Konfiguration des NetBat-Servers wird beim Start zum Teil auf ihre Korrektheit geprüft. Diese Prüfungen auf alle Eigenschaften der Konfiguration auszuweiten, ist eine Möglichkeit, das Projekt in der Zukunft zu verbessern. Das selbe gilt für die Berechnungen und Abfragen während des Betriebs, also nach Abschluss des Ladevorgangs.

Eine mögliche zukünftige Verbesserung ist es, eigene Klassen für die unterschiedlichen Fehler zu erstellen.

Im folgenden werden die in NetBat eingebauten Fehlermeldungen, deren Ausgabestreams sowie die Bedeutungen der Fehler aufgelistet.

„No config file provided.“

Weder die Environment-Konfiguration noch der Kommandozeilenparameter „--config-file“ gibt eine allgemeine Konfigurationsdatei an.

„Time period has to be a string or number“

Einer der Werte in den Downsampling-Parametern „durations“ und „steps“ der allgemeinen Konfiguration ist weder ein String noch eine Zahl.

„Time period invalid: <Wert>“

Einer der Werte in den Downsampling-Parametern „durations“ und „steps“ der allgemeinen Konfiguration hat ein ungültiges Format.

„Durations and steps of downsampling must have same length“

Die Wertelisten „durations“ und „steps“ der allgemeinen Konfiguration sind nicht gleich lang.

„Invalid HTTP output storage: <Wert>“

Die Ausgabekonfiguration (store) einer HTTP-Request hat entweder einen falschen Typ oder ein falsches Format.

„HTTP Headers must be an object“

Die Eigenschaft headers einer HTTP-Request ist angegeben, ist aber kein Object.

„Unknown database interface: <Interface>“

Das in der Konfiguration angegebene Datenbankinterface kann nicht gefunden werden.

„Variable name <Name> already in use“

Die Variable „Name“ wird in einem Device mehrfach definiert.

„Invalid table name found: <Name>“

Der Name einer Tabelle in einer Device-Konfiguration ist ungültig.

„Invalid tag name found: <Name>“

Der Name eines Tags in einer Device-Konfiguration ist ungültig.

„Invalid type at tag '<Name>': <Typ>“

Der Typ (type) eines Tags in einer Device-Konfiguration ist ungültig. Mögliche Werte sind „integer“, „boolean“, „float“ und „string“.

„Invalid field name found: <Name>“

Der Name eines Fields in einer Device-Konfiguration ist ungültig.

„Invalid type at field '<Name>': <Typ>“

Der Typ (type) eines Fields in einer Device-Konfiguration ist ungültig. Mögliche Werte sind „integer“, „boolean“, „float“ und „string“.

„Unknown variable: <Name>“

Die Variable „Name“ wird in einem Device verwendet, aber nicht definiert.

„Invalid panel id: <ID>“

Die ID eines Grafana-Panels ist ungültig.

„Could not order array unwrappers. Looks like some indices are creating loops.“

Die Beziehungen der Unwrapper einer Tabelle eines Devices bilden eine Schleife.

„Could not find variable name for description: <Unwrapper-Element>“

Für die Indexvariable eines Unwrappers wird kein Name angegeben.

„Can't apply looping configuration to reference <Name>“

Einem Unwrapper, dessen Index eine Referenz ist, werden weitere Einstellungen übergeben.

„Invalid variable name found: <Name>“

Eine Variable oder eine Variablenreferenz enthält unerlaubte Zeichen oder beginnt mit einer Zahl.

„Invalid value for <Name>: <Wert>“

In einer Device-Konfiguration wird ein ungültiger Wert namens „Name“ angegeben.

„Invalid value: <Wert>“

In einer Device-Konfiguration wird ein ungültiger Wert angegeben.

„Unknown clock: <Zeitquelle>“

Der Zeitmesser (clock) eines ValueGenerators mit der Funktion „difference“ oder „integrate“ ist ungültig. Mögliche Werte sind „internal“ und „none“.

„Unknown function: <Funktion>“

Die Funktion eines ValueGenerators ist ungültig. Die möglichen Werte finden sich in Unterkapitel 5.8.2.

„Could not get id of Grafana folder "<UID>"“

Für den Ordner mit der UID „UID“ ist von Grafana keine ID zurückgegeben worden.

„Could not find Grafana folder "<UID>"“

Der in einem Dashboard angegebene Ordner mit der UID „UID“ ist unbekannt.

„Panel <UID> not found in global Grafana configuration“

Das in der globalen Grafana-Konfiguration verwendete Panel mit der UID „UID“ wird nicht definiert.

„Panel <UID> not found in global Grafana configuration“

Das in der globalen Grafana-Konfiguration verwendete Panel mit der UID „UID“ wird nicht definiert.

„Field <Name> was given type <new Type> while already being used as <old Type>. When using InfluxDB, equal field names must have the same type.“

Fields mit gleichem Namen werden in mehreren Tabellen mit unterschiedlichen Typen angegeben. Dies ist nicht erlaubt, wenn InfluxDB benutzt wird.

„Invalid OID found at key <Key>: <OID>“

Eine im SNMP-Teil einer Device-Konfiguration angegebene OID ist unzulässig.

„SNMP community must be a string!“

Die SNMP-Community einer Device-Konfiguration ist kein String.

„Template not found: <Bezeichner>“

Das Template mit dem angegebenen Bezeichner ist nicht gefunden worden.

„File not found“

Ein bereits gefundenes Template hat beim Ladevorgang nicht eingelesen werden können.

„Folder not found: <Ordner>“

Der Ordner „Ordner“ ist nicht gefunden worden.

„Expented Array or Object at key "values", got: <Wert>“

Ein ValueGenerator, dessen Funktion ein Array, ein Object oder eine Variablenreferenz im Parameter values benötigt, hat in diesem Parameter einen anderen, unzulässigen Wert erhalten.

„Expented Array at key "values", got: <Wert>“

Ein ValueGenerator, dessen Funktion ein Array oder eine Variablenreferenz im Parameter values benötigt, hat in diesem Parameter einen anderen, unzulässigen Wert erhalten.

„Could not parse downsampling duration list“

Die in der allgemeinen Konfiguration angegebene Liste „durations“ ist falsch formatiert.

„Could not parse downsampling step list“

Die in der allgemeinen Konfiguration angegebene Liste „steps“ ist falsch formatiert.

„Error thrown while loading config“

Es hat einen Fehler beim Laden der allgemeinen Konfiguration gegeben.

„Error in downsampling schedule <Stufe> : <Error>“

Während des Downsamplings ist ein Fehler in der Stufe „Stufe“ aufgetreten.

„Exception thrown in HTTP request block of device <Adresse>:“

Während der Datenabfrage mit dem HTTP-Modul ist ein Fehler aufgetreten.

„Error loading template <Bezeichner> at <Pfad>“

Beim Laden des Templates „Bezeichner“, dessen Datei sich im Ordner „Pfad“ befindet, ist ein Fehler aufgetreten.

„Error adding template <Bezeichner>“

Beim Hinzufügen des Templates „Bezeichner“ in den Cache der Template-Suche ist ein Fehler aufgetreten.

„Error loading device <Adresse>“

Während des Ladens des Devices mit der Adresse „Adresse“ ist ein Fehler aufgetreten.

„There was an error while searching for templates at: <Pfad>“

Es ist ein Fehler aufgetreten, während im Ordner „Pfad“ nach Templates gesucht worden ist.

„There was an error while searching for devices at: <Pfad>“

Es ist ein Fehler aufgetreten, während im Ordner „Pfad“ nach Devices gesucht worden ist.

„Error thrown while reading memorized variables of device <Adresse>:“

Während die Eigenschaft memorize der Device-Konfiguration des Devices mit der Adresse „Adresse“ geladen worden ist, ist ein Fehler aufgetreten.

„Error thrown while reading <Modul> input of device <Adresse>:“

Während die Konfiguration des Moduls „Modul“ in der Device-Konfiguration des Devices mit der Adresse „Adresse“ geladen worden ist, ist ein Fehler aufgetreten.

„Error thrown while reading configuration for db table <Tabelle> of device <Adresse>:“

Beim Laden der Tabelle „Tabelle“ in der Device-Konfiguration des Devices mit der Adresse „Adresse“ ist ein Fehler aufgetreten.

„Panel <Panel> not found in device <Adresse>“

Das Grafana-Panel „Panel“ ist im Device mit der Adresse „Adresse“ nicht gefunden worden.

„Error thrown while reading configuration for grafana of device <Adresse>:“

Beim Laden der Grafana-Konfiguration des Devices mit der Adresse „Adresse“ ist ein Fehler aufgetreten.

„Error thrown while unwrapping <Variable> : <Index>“

Der Unwrapper an der Stelle „Index“ der Variable „Variable“ einer Tabelle eines Devices hat nicht geladen werden können.

„Error thrown while unwrapping“

Beim Laden der Unwrapper einer Tabelle eines Devices ist ein Fehler aufgetreten.

„Error thrown during finalization of Grafana configuration: <Error>“

Beim Finalisieren der Grafana-Konfiguration durch die Funktion applyTmpData() der Bibliothek grafana ist ein Fehler aufgetreten.

## 6 Grafische Darstellung

Die grafische Darstellung wird mithilfe von Grafana, einer plattformübergreifenden Open-Source-Analyse- und interaktiven Visualisierungssoftware realisiert, welche Diagramme, Grafiken und Warnungen für den Webbrower in Verbindung mit unterstützten Datenquellen bereitstellt. [6] Dem Anwender wird daher ermöglicht, von jedem Betriebssystem aus auf Grafana zuzugreifen und somit betriebssystemunabhängig zu sein.

### 6.1 Allgemeines zu Grafana

Grafana ist in einer Hierarchie aufgebaut, bei der Ordner erstellt werden können, die mit Dashboards (Unterkapitel 6.3) gefüllt werden. Die Dashboards können auch unabhängig von einem Ordner stehen und mehrere Panels (Unterkapitel 6.4) beinhalten. Diese Panels sind in einer oder mehreren Rows (Unterkapitel 6.5) angeordnet.

### 6.2 Globale Einstellungen

Jeder Benutzer kann individuell die User-Einstellungen konfigurieren. Die Einstellungen werden dabei unter „Preferences“ geändert. Hier können die Profileinstellungen wie „Name“, „Email“ und „Username“, eingestellt werden. Außerdem können im „Preferences“ Menü die „UI Theme“ mit „Default“, „Light“ und „Dark“, das „Home Dashboard“ und die „Timezone“ mit „Default“, „Local browser time“ und „UTC“ angepasst werden. Diese Einstellungen können jederzeit, jeweils mit „Save“ gespeichert werden. Das Passwort kann unter „Change Password“ jederzeit verändert werden. Hierfür muss zu Beginn das alte Passwort eingegeben. Danach wird ein neues Passwort gewählt und dieses zum Bestätigen ein weiteres Mal eingegeben werden. Durch die Schaltfläche „Change Password“ kann das neue Passwort gesetzt werden. [6]

## 6.3 Dashboards

In dem Menü „Create“ können Dashboards mit der Schaltfläche „Dashboard“ erstellt werden. Deren Einstellungen können dann unter „Dashboard settings“ verändert werden. Diese Einstellungen sind in die drei Abschnitte „General“, „Time Options“ und „Panel Options“ aufgeteilt. Mit der Schaltfläche „Save“ können die Einstellungen gespeichert werden. Mit „Delete Dashboard“ wird das gesamte Dashboard gelöscht. [6]

Der erste Abschnitt, „General“, beinhaltet die Optionen „Name“ und „Description“, sowie „Tag“, den zugeordneten „Folder“, in dem sich das Dashboard befindet, und, ob dieses unter „Editable“ bearbeitet werden darf. [6]

Im zweiten Abschnitt „Time Options“ können die globalen Einstellung der Zeitzone „Timezone“ nochmals individuell konfiguriert werden. Für die Zeitintervallangabe „Auto-refresh“ sind bereits „5s“, „10s“, „30s“, „1m“, „5m“, „15m“, „30m“, „1h“, „2h“ und „1d“ voreingestellt. Zu den vorhandenen Zeitwerten können noch beliebige weitere Werte hinzugefügt werden. Das Entfernen der voreingestellten Werte ist auch möglich. Hierfür wird beispielsweise, im Zuge des Projektes, aus Gründen der Genauigkeit eine zusätzliche „Auto-refresh-Zeit“ von 1s hinzugefügt. Zudem stehen die Einstellungen „Now delay now“ und „Hide time picker“ zur Verfügung. [6]

Der dritte Abschnitt „Panel Options“ enthält die „Graph Tooltip“ Einstellung, welche die Auswahlmöglichkeiten „Default“, „Shared crosshair“ und „Shared Tooltip“ besitzt. [6]

## 6.4 Panels

Panels sind die Grundbausteine von Grafana und werden über die Konfigurationsabschnitte Abfrage („Query“) und Visualisierung eingestellt. Sie können innerhalb eines Dashboard beliebig verschoben werden und sind in ihrer Größe anpassbar. [6]

Im Rahmen der Erstellung von Diagrammen, Grafiken und Warnungen stellt Grafana verschiedene Panels zur Verfügung. Neben den schon verfügbaren Panels, die bereits in Grafana inkludiert sind, können zusätzliche Panels in Form von Plugins installiert werden. Alle installierten Panels sind dann unter „Add Panel“ verfügbar und damit bereit zur Verwendung. [6]

In der folgenden Auflistung wird eine kurze Übersicht über einige Panels gegeben, die im Zuge des Projekts verwendet werden.

- **Alert List** ist in Grafana inkludiert und zeigt eine Auflistung der Alarmierungen sowie deren derzeitigen Status an. Diese Liste kann so konfiguriert werden, dass entweder der derzeitige Status oder der letzte Alarmzustand angezeigt wird. [6]
- **Clock** stellt die derzeitige Uhrzeit mit Datum oder einen Countdown dar. Die Uhrzeit kann in der 12- oder der 24-Stunden Ansicht angezeigt werden. Das „Clock“ Panel kann in Größe und Formatierung angepasst werden. [17]
- **Discrete** zeigt diskrete Werte in einem horizontalen Graphen an, welche die Zustandsübergänge klar darstellt. Das „Discrete“ Panel ist zur Veranschaulichung von String und Boolean Daten geeignet. [18]
- **FlowCharting** zielt darauf ab, Diagrammzeichnungen, Topologien, Blockschaltbilder, Gebäudegrundrisse und Workflows mithilfe der Website „draw.io“ zu zeichnen und im folgenden visuell darzustellen. [19]
- **Gauge** ist in Grafana inkludiert und kann ein einzelnes Wertefeld als Messgerät für jede Serie, Spalte oder Zeile anzeigen. Hierbei können entweder alle Werte einzeln oder ein berechneter Wert angezeigt werden. [6]
- **Graph** ist das Hauptpanel, welches in Grafana inkludiert ist. Es bietet eine große Vielfalt an Darstellungsoptionen. Das „Graph“ Panel unterstützt als einziges Panel Alarmierungen. [6]
- **Multistat** wurde von dem schon inkludierten „Singlestat“ Panel inspiriert. Ein „Singlestat“ Panel zeigt einzelne Metriken aus Zeitreihendatensätzen mit optionaler schwellenwertbezogener Farbgebung an. „Multistat“ Panel orientiert sich an dieser Basis, enthält jedoch mehrspaltige Tabellendatensätze. Diese Abfragedaten können in Form von Balkendiagrammen mit optionalen oberen und unteren Grenzen angezeigt werden. [20]

Die Panels werden regelmäßig im „Auto-refresh-Zeitintervall“ des Dashboards aktualisiert. Eine Ausnahme sind dabei Panels wie zum Beispiel das „FlowCharting“ Panel, da bei diesen keine Zeitspanne veranschaulicht wird. [19]

Das aufklappbare Menü der Panels beinhaltet die Optionen „View“, „Edit“, „Share“, „Explore“, „More“ und „Remove“.

Wird „View“ gewählt, so wird das Panel in Vollbild dargestellt.

Bei „Edit“ öffnet sich das Panel-Menü, welches sich in die Optionen „Queries“, „Visualization“, „General“ und im Fall des „Graph“ Panel „Alert“ gliedert. Im Menüpunkt „Queries“ werden die einzelnen Daten in den Grafen konfiguriert. „Visualization“ ermöglicht die Einstellung von „Draw Modes“. Hierbei kann gewählt werden, ob die Daten mit Balken, Linien oder Punkten dargestellt werden. Außerdem wird unter „Mode Option“ bestimmt, ob der Graf ausgefüllt sein soll. Ist er gefüllt, wird hier bestimmt, mit welchem Füllgrad er dargestellt wird. Auf dieselbe Weise kann die Linienbreite zugewiesen werden. Unter „Hover tooltip“ wird gewählt, ob „All series“ oder „Single“ und ob eine Sortierung vorliegt. Eine Sortierung kann auf- oder absteigend sein. „Stacked value“ besitzt die Auswahlmöglichkeiten „individual“ oder „cumulative“. „Stacking and Null value“ ermöglicht die Stapelung der einzelnen Daten im Panel. Die Prozent-Anzeige kann entweder ein- oder abgeschaltet werden. „Null value“ bietet die Auswahl zwischen „connected“, „null“ oder „null as zero“. Die Darstellung sowie Beschriftung der einzelnen Achsen des Grafen wird im Abschnitt „Axes“ eingestellt. Der Abschnitt „Legend“ gliedert sich in drei Spalten, die sich mit „Options“, „Values“ und „Hide series“ betiteln. Unter „Options“ wird gewählt, ob eine Legende vorhanden ist, wie diese angeordnet sein soll (entweder unter dem Grafen oder rechts davon) und ob sie als Tabelle ausgeführt wird. „Values“ bestimmt, welche der folgenden Werte in der Legende zu sehen sein sollen. Hierbei kann zwischen dem Minimalwert, dem Maximalwert, dem arithmetischen Mittel, dem derzeitigen Stand und der Summe aller Werte gewählt werden. Hier wird auch bestimmt, mit wie vielen Dezimalstellen diese Werte angezeigt werden. „Hide series“ bietet die Auswahl zwischen „With only nulls“ und „With only zeros“. Von diesen Werten muss aber keiner angewählt werden. Die Abschnitte „Threshold and Time Regions“ und „Data links“ sind bei dem Projekt nicht verwendet worden. Im Menüpunkt „General“ wird der Titel und die Transparenz des Panels gewählt. „Description“ ermöglicht das Hinzufügen einer Beschreibung für den jeweiligen Grafen. Im Kapitel 7 wird der „Alert“ Abschnitt im Detail behandelt.

Zum Teilen des Grafen dient der Menüpunkt „Share“.

„Explore“ ist bei Experimenten hilfreich und kommt dann zum Einsatz, wenn an dem einzelnen Panel nichts verändert werden soll. Der Grund dafür ist, dass im „Explore“ Bereich ein Panel lokal, im Browser konfiguriert werden kann, ohne dass es abgespeichert wird.

Unter „More“ sind Funktionen wie „Duplicate“ und „Copy“ zu finden. Die Funktion „Panel JSON“ ermöglicht es den Grafen als JSON anzusehen und dieses zu verändern.

„Remove“ ermöglicht das Löschen des Grafen.

## 6.5 Rows

Um Panels zu gruppieren wird die Zeilenfunktion „Row“ verwendet. „Row“ ist dabei der logische Teiler innerhalb des Dashboards. Die „Row“ wird je nach horizontaler Auflösung des Webbrowsers automatisch skaliert. Die Breite der Panels kann innerhalb einer Zeile gesteuert werden, in dem die spezifische Breite festlegt wird. Die Zeilen können durch Anklicken des Zeilentitels reduziert beziehungsweise erweitert werden. [6]

## 6.6 Playlist

Mit der Wiedergabeliste wird eine Sequenz von Dashboards angezeigt. Dies wird genutzt, um die Sequenz zu präsentieren, zum Beispiel dem Team oder Besuchern. Die Skalierung des Dashboards wird je nach Auslösung automatisch durchgeführt und passt sich daher an die jeweilige Bildschirmgröße an. [6]

Eine neue Wiedergabeliste kann unter „New Playlist“ im Menü „Playlist“ erstellt werden. Hierbei ist der Name der Liste und das Umschaltintervall zu wählen. Die Liste der verfügbaren Dashboards ist unter „Add dashboards“ zu sehen und kann mit „Add to playlist“ hinzugefügt werden. Mit den Pfeilen wird die Reihenfolge, in der die Dashboards abgespielt werden, definiert. Die Konfiguration einer Wiedergabeliste wird mit „Create“ abgeschlossen. Um Wiedergabelisten oder Dashboards zu entfernen, ist das „X“ zu betätigen. [6]

Soll eine bereits erstellte Wiedergabelist bearbeitet werden, kann dies mit einem Klick auf die Liste erfolgen. Etwaige Änderungen werden mit „Save“ gespeichert. [6]

## 6.7 Topologieansicht

Das „FlowCharting“ Panel ermöglicht das Darstellen von Topologien, Blockschaltbildern, etc., welche mit draw.io erstellt werden. Diese können dann auf zwei verschiedene Arten konfiguriert werden. [19] Zum einen kann die Konfiguration über das „Query“ oder per SQL über den „text edit mode“ erfolgen. Bei keinem oder nur einem Tag ist die Konfiguration über SQL möglich aber nicht zwingend, bei mehreren Tags ist es notwendig und kann nicht umgangen werden.

Die in „draw.io“ erstellten Darstellungen werden im Dateiformat von „.drawio“ gespeichert. Die Darstellung kann auch im Vektorgrafikformat SVG abgespeichert werden, um das Ändern der Elementfarben zu erleichtern.

## 6.8 Umsetzung im Testnetzwerk

Hier wird die grafische Implementierung des Testnetzwerk, welches in Unterkapitel 8.2 ausgeführt ist, beschrieben. Diesbezüglich wurden mehrere Ordner erstellt.

### 6.8.1 Ordner

Die erstellten Ordner „Devices“ und „General“ sind mit Dashboards befüllt worden, welche wiederum verschiedene Panels in unterschiedlichen Anordnungen beinhalten.

### 6.8.2 Dashboards

In dem Ordner „General“ finden sich die Dashboards „Overview“ und „Topology“. Alle überwachten Geräte des Netzwerks werden, jeweils in einem eigenen Dashboard, in dem Ordner „Devices“ angeordnet. Das Dashboard „Netbat Logo“ beinhaltet das Netbat-Logo.

#### 6.8.2.1 *Overview*

In dem „Overview“ Dashboard werden die Uhrzeit mit Datum sowie die wichtigsten Daten in einer kurzen Übersicht präsentiert. Welche Daten angezeigt werden, wird in den verschiedenen Templates definiert (siehe Kapitel 6.9).

### 6.8.2.2 Topologie

Das Erstellen von Topologien wird in Unterkapitel 6.7 beschrieben. Die Konfiguration erfolgt in diesem Fall händisch und nicht über ein Template. Die Daten RTT und CPU werden anhand einer Topologie visualisiert. Der Status der einzelnen Geräte wird durch die Farben grün, orange und rot ausgedrückt.

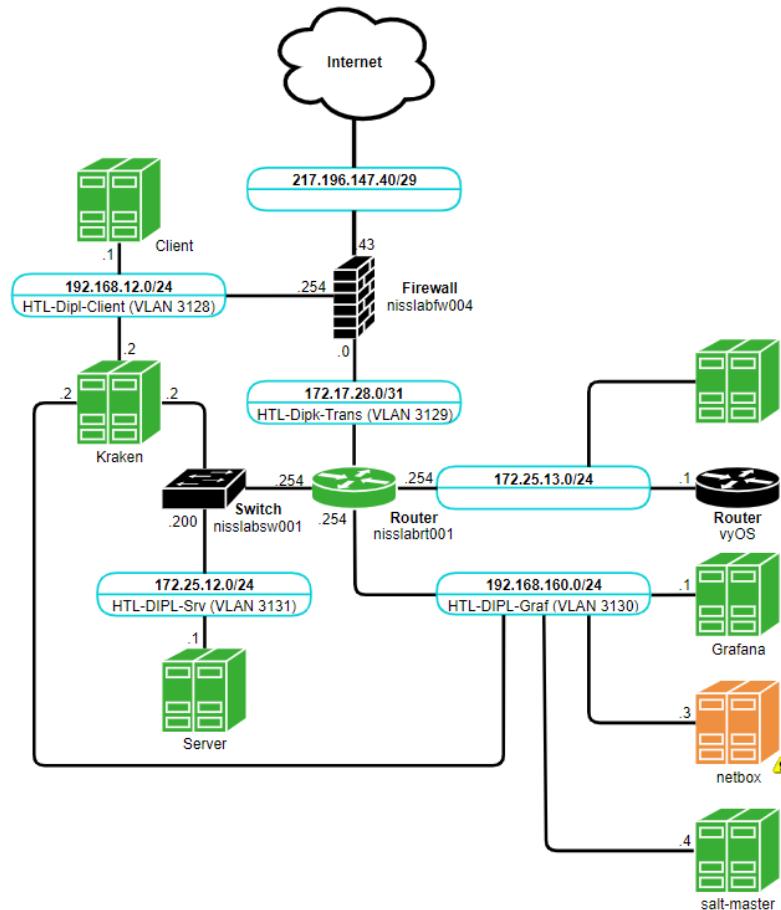


Abbildung 6.1: Topologie in Grafana GUI

Standardmäßig wird die Abfrage wie folgt konfiguriert. Dies funktioniert ausschließlich, wenn die Abfrage nicht über mehrere Tags verfügt.

FROM	default	cpu_usage	WHERE	hostaddress	=	192.168.160.254	+
SELECT	field (value)	last ()	+				
GROUP BY	time (\$__interval)	fill (none)	+				
FORMAT AS	Time series	-					
ALIAS BY	Router CPU						

Abbildung 6.2: Query Darstellung in Grafana GUI

Da die CPU über mehrere Tags verfügt, muss sie speziell für diesen Zweck konfiguriert werden. Es wird folgendermaßen vorgegangen:

---

```
SELECT (last("system") + last("user") + last("guest") + last("nice")) * 100 / (
    last("system") + last("user") + last("guest") + last("nice") + last("idle")
    + last("steal") + last("iowait") + last("irq") + last("softIRQ")) AS value
FROM "munin_cpu" WHERE ("hostaddress" = '192.168.12.1') AND $timeFilter
GROUP BY time($__interval) fill(none)
```

---

### 6.8.2.3 Devices

#### Router

Der Router wird in dem Dashboard „Devices“ geführt. Hierbei wird der Ping mithilfe des Templates `grafana_global.json` erstellt und angezeigt. Die Daten, welche die Auslastung von CPU und Random-Access Memory (RAM) betreffen, werden mithilfe des `router/cisco/ASR-920-4SZ-A` Templates generiert. Die Interfaces 1, 2, 10, 11, 12, 13, 14 und 15, die den Status, den Throughput und die Fail-Rate anzeigen, werden händisch erstellt.

#### Server

Die Server sind nach der im Unterkapitel 8.2.2.1 dargestellten Serverarchitektur aufgebaut. Für die Erstellung der Visualisierung der einzelnen Geräte wird das Template `other/munin_generic` verwendet. Diese Geräte befinden sich im Ordner „Devices“. Es werden folgende Geräte veranschaulicht:

- **Client** (192.168.12.1)
- **Server** (172.25.12.1)
- **Proxy** (172.25.13.10)
- **Grafana** (192.168.160.1)
- **Kraken** (192.168.160.2)
- **Netbox** (192.168.160.3)
- **Salt-Master** (192.168.160.4)

### 6.8.2.4 NetBat Icon

Das Netbat-Logo wird mit der Adresse Localhost eingetragen und mit dem Template other/netbat-icon generiert. Es befindet sich nicht im Dashboard „Overview“, sondern ist nur im Dashboard „Netbat Logo“ zu sehen. Das NetBat-Logo ist in Abbildung 6.3 veranschaulicht.

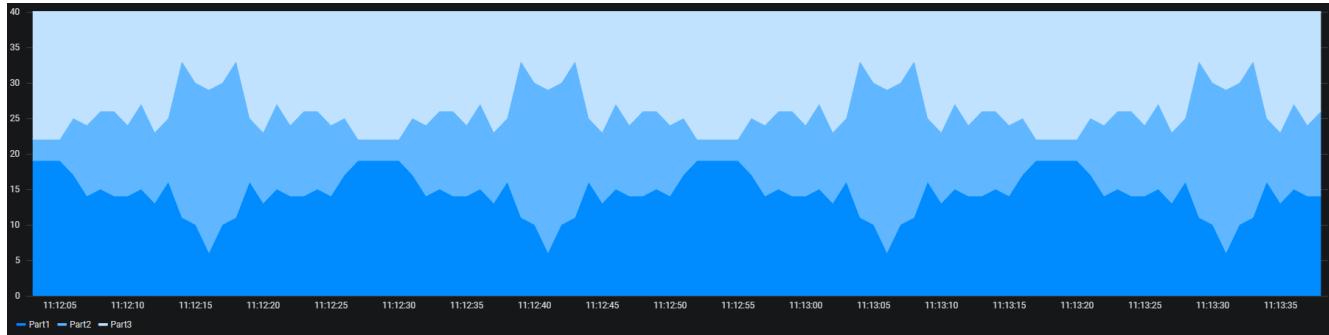


Abbildung 6.3: NetBat-Logo in Grafana GUI

## 6.9 Templates

Die Templates werden in den Dateien grafana\_global.json, ASR-920-4SZ-A.json, munin-generic.json und netbat-icon.json, welche eigenständig konfiguriert werden, erstellt.

Die Konfigurationsdatei grafana\_global.json hat die in Unterkapitel 5.7.4 beschriebene Struktur. Die Templates werden wie in Unterkapitel 5.8 angegeben konfiguriert, wobei im Template router/cisco/ASR-920-4SZ-A das SNMP-Modul, im Template other/munin\_generic das Munin-Modul und im Template other/netbat-icon das Modul „netbat“ verwendet wird.

Die Device-Grafana-Konfiguration wird in allen Templates auf die folgende Art und Weise eingestellt. Hierbei wird über die „dashboards“ Einstellung gesteuert, ob die Panels in „Overview“, in einem eigenen oder in beiden Dashboards vorhanden sind.

```
"grafana": {
    "panels": {
        <panel uid>: <panel>
    },
    "dashboards": [
        {
            "uid": "netbat-overview",
            "panels": [
                {
                    "uid": {
                        "$": "device-$address$"
                    },
                    "rowTitle": {

```

```
        "$": "Device $address$ (\\"$hostname$\")"
    },
    "panels": [
        <panel uid>
    ]
}
],
{
    "uid": {
        "$": "device-$address$"
    },
    "title": {
        "$": "Device $address$ (\\"$hostname$\")"
    },
    "panels": [
        <panel uid>
    ],
    "folder": "netbat-devices"
}
]
}
```

---

### 6.9.1 grafana\_global

Die Datei `grafana_global.json` ist für die Erstellung des Ordners „Devices“ verantwortlich. Zudem dient sie zur Konfiguration des „Clock“ Panel im Dashboard „Overview“.

Mithilfe von "pingTemplate" werden ein Panel „\_rtt“ mit und ein Panel „\_rtt2“ ohne Warnung konfiguriert, da bei den Dashboards im Ordner „Devices“ Alarmierungen vorhanden sein sollen und in „Overview“ nicht. Auf diese Weise werden doppelte Alarmierungen verhindert.

## 6.9.2 router/cisco/ASR-920-4SZ-A

Das `router/cisco/ASR-920-4SZ-A` Template stellt die allgemeinen Daten, wie die RTT und die Auslastung, dar.

### RTT

Die RTT wird in dem „Graph“ Panel veranschaulicht und mit einer Alarmierung (siehe Kapitel 7) versehen, durch welche man bei Störungen verständigt wird. In `grafana_global.json` (siehe Unterkapitel 6.9.1) wird bestimmt, in welchen Dashboards und Panels diese Warnung beigelegt ist. Das aufklappbare Menü verfügt über die Funktion „Panel JSON“ und ermöglicht es ein Template zu konfigurieren. Dafür müssen folgende Anpassungen getroffen werden.

Die bereits vorhandenen "datasource" und "id" werden für die Template-Konfiguration entfernt. Ebenso wird die Positionsangabe, die in "gridPos" steht, nicht benötigt, außer es ist eine Herabsetzung der Breite „w“, von den üblichen 24 auf beispielsweise 12, vonnöten. Zudem müssen auch im Bereich "tags", der von Grafana wie folgt übergeben wird, Anpassungen getroffen werden.

---

```
"tags": [
  {
    "key": "hostaddress",
    "operator": "=",
    "value": "192.168.12.1"
  }
]
```

---

Hierfür wird der Wert "value", wie im Folgenden zu sehen ist, geändert.

---

```
"tags": [
  {
    "key": "hostaddress",
    "operator": "=",
    "value": {
      "$": "$address"
    }
  }
]
```

---

### 6.9.3 other/munin-generic

Bei den Servern werden die Daten RTT, „CPU Usage“, „RAM Usage“, „Disk Usage“, „Processes“, „Process Priority“, „Swap“, „vmstat“, „netstat“, „Threads“, „Load“, „Entropy“, „Open Files“ und „Uptime“ visualisiert. Die Änderungen gestalten sich gleich, wie schon in dem Unterkapitel 6.9.2 beschrieben. Eine Ausnahme ist hierbei die „Disk Usage“.

Da eine Reihe an Templates erstellt worden sind, die in Grafana visualisiert worden sind, resultieren daraus einige Daten, welche in der folgenden Auflistung benannt und teilweise genauer beschrieben werden.

- **RTT** - Round Trip Time
- **CPU-Usage**
  - **system** - CPU-Zeit, die der Kernel mit Systemaktivitäten verbringt.
  - **user** - CPU-Zeit, die von normalen Programmen und Hintergrundprozessen benötigt wird.
  - **guest** - Die Zeit, die für die Ausführung einer virtuellen CPU für Guest-Betriebssysteme unter der Kontrolle des Linux-Kernels aufgewendet wird.
  - **idle** - Leerlauf-CPU-Zeit.
  - **nice** - CPU-Zeit, die von nice(1)d Programmen verbraucht wird.
  - **steal** - Die Zeit, in der eine virtuelle CPU ausführbare Aufgaben hat, die virtuelle CPU selbst jedoch nicht ausgeführt wird.
  - **io wait** - CPU-Zeit, die damit verbracht wurde, auf den Abschluss der I/O-Operationen zu warten, wenn nichts anderes zu tun ist.
  - **irq** - CPU-Zeit für die Bearbeitung von Interrupts.
  - **soft irq** - CPU-Zeit für die Verarbeitung von „gestapelten“ Interrupts.
- **RAM-Usage**
  - **slab** - Vom Kernel verwendeter Speicher (Hauptbenutzer sind Caches wie Inode, Dentry, etc.).
  - **apps** - Speicher, der von Anwenderprogrammen verwendet wird.
  - **buffers** - Block-Geräte-Cache (zum Beispiel Festplatte). Auch dort, wo Blöcke bis zum Schreiben gespeichert werden.
  - **swap cache** - Ein Teil des Speichers, der Seiten verfolgt, die aus dem Swap abgerufen, aber noch nicht verändert worden sind.

- **cached** - Chache mit „geparkte“ Dateidaten (Dateiinhalten).
- **free** - Speicher, der für nichts verwendet wird.
- **page tables** - Speicher, der zum Zuordnen zwischen virtuellen und physischen Speicheradressen verwendet wird.
- **swap** - Verwendeter virtueller Arbeitsspeicher

- **Disk-Usage**

- **var**
- **home**
- **root**
- **tmp**
- **sda1**
- **shm**

- **Processes**

- **dead** - Die Anzahl der toten Prozesse.
- **idle** - Die Anzahl der inaktiven Kernel-Threads ( $\geq 4,2$  Kernel).
- **interrupt**
- **lock**
- **runnable** - Die Anzahl der ausführbaren Prozesse (in der Ausführungswarteschlange).
- **running**
- **sleeping** - Die Anzahl der schlafenden Prozesse.
- **stopped** - Die Anzahl der gestoppten oder verfolgten Prozesse.
- **suspended**
- **uninterruptible** - Die Anzahl der unterbrechungsfreien Prozesse.
- **zombie** - Die Anzahl der nicht mehr existierenden Prozesse.

- **Process Priority**

- **high priority** - Die Anzahl der Prozesse mit hoher Priorität.
- **low priority** - Die Anzahl der Prozesse mit niedriger Priorität.

- **locked in memory**
- **Swap**
  - **in**
  - **out**
- **vmstat**
  - **io-sleep**
  - **other wait states**
- **netstat**
  - **active**
  - **established**
  - **failed**
  - **passive**
  - **resets**
- **Threads** - Die aktuelle Anzahl an Threads.
- **Load** - Durchschnittslast.
- **Entropy** - Die Anzahl an zufälligen, verfügbaren Bytes. Dies wird unter anderem von kryptographischen Anwendungen verwendet.
- **Open Files** - Die Anzahl der aktuell geöffneten Dateien.
- **Uptime** - Betriebszeit des Servers in Tagen

## **Disk Usage**

Da die Speicherauslastung der Festplatten von essentieller Bedeutung ist, wird diese bei jedem Server angezeigt. Zudem wird die Speicherauslastung zu Beginn des „Overview“ Dashboards mithilfe des „Mulitstat“ Panel dargestellt. Das „Mulitstat“ wird händisch konfiguriert und somit nicht mithilfe eines Templates erstellt. Ein Vorteil dieser Ansicht ist, dass auf einem Blick visualisiert ist, ob die Auslastung im grünen, gelben oder roten Bereich liegt. Im grünen ist alles in Ordnung, im gelben können präventiv Maßnahmen ergriffen werden, wie das Anschließen einer weiteren Festplatte oder das Löschen größerer Dateien. Im roten Bereich sollten Maßnahmen getroffen werden, da sonst Störungen auftreten können.

Zusätzlich wird die Auslastung bei jedem Gerät, mit dem Template other/munin\_generic in einem „Graph“ Panel dargestellt. Zu diesem Panel wird eine Alarmierung, wie in Kapitel 7 beschrieben, beigefügt. Hierfür wird „munin\_disk“ mit und „munin\_disk2“ ohne Warnung konfiguriert. Folgende Anpassungen sind daher zu treffen.

Wie schon in Unterkapitel 6.9.2 beschrieben, werden die Abschnitte "datasource", "id" und "gridPos" entfernt. Außerdem wird "targets" folgendermaßen abgeändert.

---

```
    "targets": {
      "$": {
        "function": "arraymap",
        "keyname": "diskName",
        "valuename": "diskValueUnused",
        "input": {
          "$": "$munin_df"
        },
        "value": {
          "alias": {
            "$": "$diskName"
          },
          "groupBy": [
            {
              "params": [
                "$__interval"
              ],
              "type": "time"
            },
            {
              "params": [
                "previous"
              ],
              "type": "fill"
            }
          ],
          "measurement": "munin_df",
          "orderByTime": "ASC",
          "policy": "default",
          "query": {
            "$": "SELECT mean(\"value\") FROM \"munin_df\" WHERE (\"hostaddress\" = '192.168.12.1' AND \"file_system\" = '$diskName$') AND $$timeFilter GROUP BY time($$__interval) fill(previous)"
          },
          "rawQuery": false,
          "refId": "A",
          "resultFormat": "time_series",
          "select": [
            [
              {
                "params": [
                  "value"
                ]
              }
            ]
          ]
        }
      }
    }
```

```

        ],
        "type": "field"
    },
    {
        "params": [],
        "type": "mean"
    }
]
],
"tags": [
{
    "key": "hostaddress",
    "operator": "=",
    "value": {
        "$": "$address"
    }
},
{
    "condition": "AND",
    "key": "file_system",
    "operator": "=",
    "value": {
        "$": "$diskName"
    }
}
]
}
}

```

#### 6.9.4 other/netbat-icon

In dem Template other/netbat-icon wird das Netbat-Logo erstellt und in Grafana visualisiert. Wie in Unterkapitel 6.9.2 beschrieben, werden die Abschnitte "datasource", "id" und "gridPos" weggelassen. Dass das NetBat-Logo nur in seinem eigenen Dashboard vorhanden ist, wird wie im Folgenden bewerkstelligt.

```

"dashboards": [
{
    "uid": "NetBat-Logo",
    "title": "NetBat Logo",
    "panels": [
        "netbat_logo"
    ]
}
]

```

## 7 Alarmierung

Um informiert zu werden, wenn ein Problem innerhalb des Netzwerks auftritt, wird eine Warnung zu den kritischen Geräten beigefügt. Hierbei hilft Grafana mit der Funktion „Alert“, bei der die Punkte „Rule“, „Conditions“, „No Data and Error Handling“ und „Notifications“ abgearbeitet werden. Derzeit ist diese Funktion ausschließlich bei dem „Graph Panel“ verfügbar. [6]

### 7.1 Regeln

Im Abschnitt „Rule“ werden der Name und das Auswertungsintervall der Alarmierungsregel spezifiziert. Bei „For“ ist zu beachten, dass dies nicht benutzt wird, wenn im Abschnitt „No Data and Error Handling“ der Bereich „If no data or all values are null“ auf „No Data“ gestellt ist. Ist für eine Benachrichtigungsregel ein „For“ konfiguriert und über- beziehungsweise unterschreitet die Abfrage den eingestellten Schwellenwert, wechselt dieses zuerst von „OK“ auf „Pending“. Wenn von „OK“ auf „Pending“ gewechselt wird, wird keine Benachrichtigung gesendet. Sobald die Alarmierungsregel länger als für die gewählte Dauer ausgelöst wird, wechselt sie zu „Altering“ und sendet somit eine Alarmbenachrichtigung. [6]

## 7.2 Bedingungen

Unter „Conditions“ werden die Bedingungen gesetzt, welche die Alarmierung konfigurieren. Derzeit ist ausschließlich der Typ „Query“ als Bedingung wählbar.

In Grafana könnte eine solche Konfiguration wie in Abbildung 7.1 aussehen.

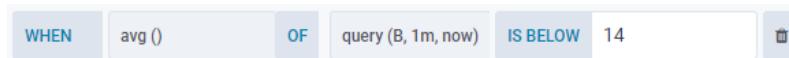


Abbildung 7.1: Warnungsbedingung in Grafana GUI

Die erste Bedingung beginnt immer mit „WHEN“. Ab der zweiten Bedingung werden „AND“- und „OR“-Operationen unterstützt und seriell ausgeführt.

Danach wird eine Aggregationsfunktion aus der folgenden Auflistung gewählt, mit welcher der Schwellenwert verglichen wird.

- avg()
- sum()
- count()
- last()
- min()
- max()
- median()
- diff()
- percent-diff()
- count-non-null()

Mit dem Buchstaben (A, B,...) wird angegeben, mit welcher Abfrage die Metriken ausgeführt werden. Die Parameter „1m und now“ definieren den Zeitbereich, in dem geprüft wird. In der Abbildung 7.1 geht dieser Bereich von vor einer Minute bis zum momentanen Zeitpunkt.

Weiters stehen die Auswahlmöglichkeiten „HAS NO VALUE“, „IS BELOW“, „IS ABOVE“, „IS OUTSIDE RANGE“ oder „IS WITHIN RANGE“ zur Verfügung. Bei der Auswahl von „HAS NO VALUE“ wird keine weitere Eingabe verlangt. Wird „IS BELOW“ oder „IS ABOVE“ gewählt, ist jeweils ein Wert (1, 14, ...) einzugeben. Fällt die Wahl auf „IS OUTSIDE RANGE“ oder „IS WITHIN RANGE“, ist ein Wertebereich durch Verwendung des Wortes „TO“ anzugeben. [6]

## 7.3 Fehlerbehandlungen

Die Fehlerbehandlung konfiguriert, wie Abfragen verarbeitet werden, wenn keine Daten oder null zurückgegeben werden. Hier kann aus den folgenden Optionen gewählt werden. [6]

- **NoData** - Setzt den Status der Alarmregel auf „NoData“.
- **Alerting** - Setzt den Status der Alarmregel auf „Alarm“.
- **Keep Last State** - Behält den aktuellen Status der Alarmregel bei.

Für das Feld „If execution error or timeout“ können die nachfolgenden Optionen gewählt werden.

- **Alerting** - Setzt den Status der Alarmregel auf „Alarm“.
- **Keep Last State** - Behält den aktuellen Status der Alarmregel bei.

Wenn die Datenbankabfragen unzuverlässig sind und fehlschlagen können, etwa durch eine Zeitüberschreitung, kann die Option „Keep Last State“ ausgewählt werden. Der letzte Status wird dann beibehalten, um die Zeitüberschreitung oder zufälliges Fehlschlagen zu ignorieren. [6]

## 7.4 Benachrichtigungen

Ändert eine Alarmierung den Zustand, so sendet diese eine Benachrichtigung. Um eine Benachrichtigung hinzufügen zu können, muss zuvor ein „Notification channel“ (Telegram, E-Mail, ...) konfiguriert werden. Ein solcher „Notification channel“ kann unter „Alerting“ auf der „Notification channel“ Seite erstellt werden. Die Schaltfläche „New channel“ verweist auf ein Menü, in dem der Name und Typ angegeben wird. Mit der Schaltfläche „Send Test“ wird eine Testnachricht versandt. [6]

Ist ein „Notification channel“ erstellt worden, so können Benachrichtigungen zu den Alarmierungsregeln hinzugefügt werden. Bei einer Benachrichtigung kann eine detaillierte Meldung zu einer konkreten Regel angegeben werden. Die Nachricht kann dabei jede Information darüber enthalten. Dies kann beispielsweise mit einem Link zum Benutzerhandbuch/Support oder mit der Information, wie das Problem zu beheben ist, bewerkstelligt werden. [6]

## 7.5 Beispiele

Im Rahmen des Projekts sind für RTT (siehe Unterkapitel 7.5.1) und Festplattenauslastung (siehe Unterkapitel 7.5.2), jeweils eine Alarmierung hinzugefügt worden. Die Alarmierungskonfiguration wird hierbei auf zwei unterschiedliche Varianten realisiert. An den folgenden beiden Beispielen wird ausgeführt, wie diese zu implementieren sind.

### 7.5.1 RTT

Für die Benachrichtigung von RTT wurde eine Methode gewählt, bei der im „Normalbetrieb“ keine Alarmierung zusehen ist. Dies ist in Abbildung 7.2 zu sehen. Im Alarmierungsfall wechselt der Hintergrund des Graphen aber automatisch auf rot.

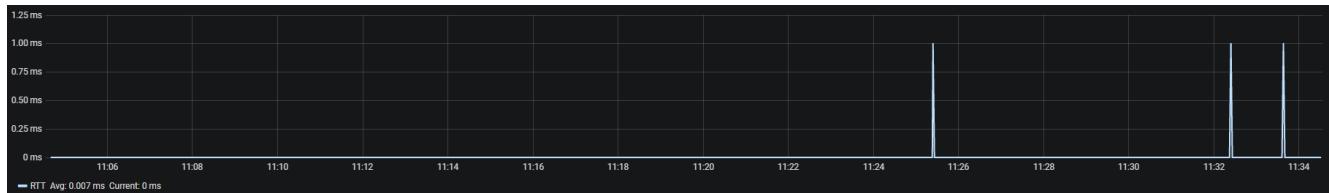


Abbildung 7.2: RTT mit Alarmierung in Grafana GUI

#### 7.5.1.1 Vorbereitung

Hierfür müssen bereits Vorbereitungen beim Erstellen der einzelnen Abfragen getroffen werden, da diese für die Alarmierungskonfiguration vorausgesetzt sind.

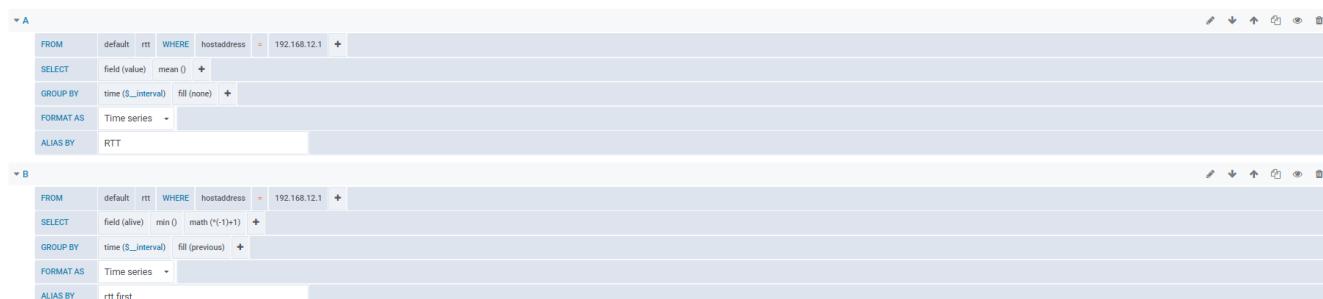


Abbildung 7.3: Query-Konfiguration zur Alarmierungsvorbereitung für RTT in Grafana GUI

Wie in der Abbildung 7.3 veranschaulicht wird, muss eine zweite RTT Abfrage, in diesen Fall Query „B“, erstellt werden, welche auf „alive“ gesetzt wird. Query „B“ ist ausschließlich für die Alarmierung notwendig und in der Panel-Ansicht nicht sichtbar. Daher werden einige Bedingungen in der „Visualization“ (siehe Abbildung 7.4) getroffen. Dementsprechend ist die Sichtbarkeit der rechten Y-Achse zu deaktivieren.



Abbildung 7.4: Visualisierungseinstellungen für RTT in Grafana GUI

Die rechte Y-Achse muss dennoch, wie in Abbildung 7.4 zu sehen, auf „linear“ gestellt werden und der Minimal- und Maximalwert einen Bereich von 0 bis 1 besitzen. Daraus folgt, dass nur Query „A“ samt linker Y-Achse im „Graph“ Panel dargestellt wird. Query „B“ besitzt ebenfalls eine lineare Skala und einen Minimalwert bei 0. Der Maximalwert befindet auf „auto“ und passt sich daher selbstständig an den Grafen an.

### 7.5.1.2 Alarmkonfigurationen für RTT

Nach den vorbereiteten Konfigurationen (Unterkapitel 7.5.1.1) kann die Benachrichtigung unter „Alert“ erstellt werden.

#### Regeln

Hier wird, wie in Unterkapitel 7.1 beschrieben, die Benennung und das Auswertungsintervall definiert. Die Benennung der Warnung soll dabei eindeutig sein. Das Auswertungsintervall ist in diesem Beispiel auf 1s gesetzt worden. Daraus resultiert, dass jede Sekunde geprüft wird. Dies ist in der Abbildung 7.5 veranschaulicht.



Abbildung 7.5: Alarmregeln für RTT in Grafana GUI

## Bedingungen

Anschließend sind die Bedingungen gesetzt worden, unter welchen Voraussetzungen eine Alarmierung getätigkt wird. Dies ist in Unterkapitel 7.2 beschrieben. In der Abbildung 7.6 ist zu sehen, dass auf drei Bedingungen geprüft wird.

WHEN	max ()	OF	query (B, 5s, now)	IS BELOW	-7000	
OR	min ()	OF	query (B, 5s, now)	IS ABOVE	0	
OR	last ()	OF	query (B, 1m, now)	HAS NO VALUE		

Abbildung 7.6: Alarmbedingungen für RTT in Grafana GUI

## Fehlerbehandlung

Bei der Fehlerbehandlung wird festgelegt, wie vorgegangen werden soll, wenn Fehler auftreten oder keine Daten vorhanden sind. Grundlegende Informationen diesbezüglich können im Unterkapitel 7.3 nachgelesen werden. Die Konfiguration ist in Abbildung 7.7 veranschaulicht.

If no data or all values are null	SET STATE TO	Keep Last State
If execution error or timeout	SET STATE TO	Alerting

Abbildung 7.7: „NoData“ und „Error Handling“ Optionen für RTT in Grafana GUI

In Abbildung 7.7 ist zu sehen, dass „If no data or all values are null“ auf „Keep Last State“ und „If execution error or timeout“ auf „Alerting“ gesetzt worden ist.

## Benachrichtigungen

In Abbildug 7.8 ist eine Benachrichtigung zu sehen, die an den „Notification channel“ „telegram-test-group“ mit der Nachricht „Server down > 5s“ geschickt wird. Es können auch Nachrichten an Einzelpersonen versendet werden.

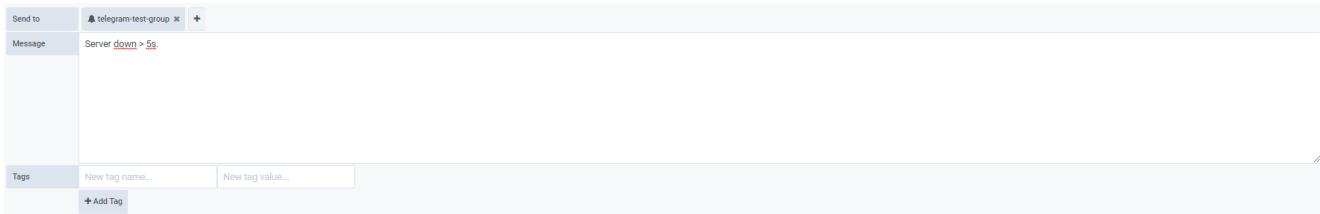


Abbildung 7.8: Benachrichtigung für RTT in Grafana GUI

Zudem ist es möglich, Nachrichten mit einem größeren Umfang zu erstellen und zu verschicken. Dies ist in Unterkapitel 7.4 beschrieben.

## JSON

Der Code für die Alarmierung sieht in behandelten Fall wie folgt aus:

```

"alert": {
    "alertRuleTags": {},
    "conditions": [
        {
            "evaluator": {
                "params": [
                    -7000
                ],
                "type": "lt"
            },
            "operator": {
                "type": "and"
            },
            "query": {
                "params": [
                    "B",
                    "5s",
                    "now"
                ]
            },
            "reducer": {
                "params": [],
                "type": "max"
            },
            "type": "query"
        },
        {
            "evaluator": {
                "params": [

```

```
          0
        ],
      "type": "gt"
    },
    "operator": {
      "type": "or"
    },
    "query": {
      "params": [
        "B",
        "5s",
        "now"
      ]
    },
    "reducer": {
      "params": [],
      "type": "min"
    },
    "type": "query"
  }, {
    "evaluator": {
      "params": [],
      "type": "no_value"
    },
    "operator": {
      "type": "or"
    },
    "query": {
      "params": [
        "B",
        "5s",
        "now"
      ]
    },
    "reducer": {
      "params": [],
      "type": "last"
    },
    "type": "query"
  }
],
"executionErrorState": "alerting",
"for": "10s",
"frequency": "1s",
"handler": 1,
"message": "Server down > 5s.",
"name": "Client rtt Alert",
"noDataState": "keep_state"
},
```

## 7.5.2 Festplattenauslastung

Die Festplattenauslastung, die in „Disk-Usage“ zu sehen ist, kann höchstens einhundert Prozent erreichen. Um dies zu verhindern, ist eine Benachrichtigung hinzugefügt worden, die ab der Grenze von neunzig Prozent aktiv wird. Bei dieser Methode wird bei der 90-Prozent-Marke ein roter Bereich angezeigt, wie in Abbildung 7.9 zu sehen ist. Unter dem Menüpunkt „Alert“ können die Konfigurationen der Alarmierung

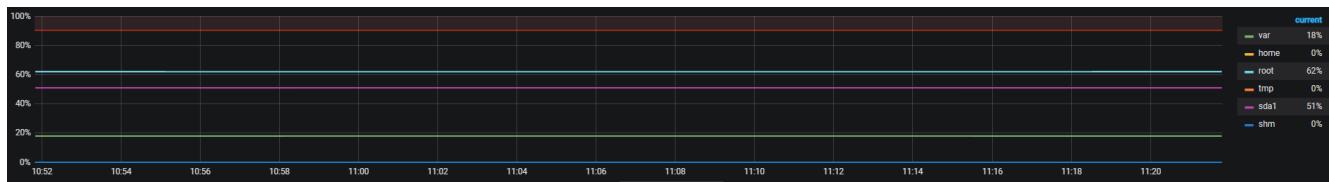


Abbildung 7.9: Festplattenauslastung mit Alarmierung in Grafana GUI

getroffen werden. Die Auslastung ist, wie im Unterkapitel 7.5.2.1 beschrieben, gesetzt.

### 7.5.2.1 Alarmkonfigurationen der Festplattenauslastung

#### Regeln

Zu Beginn ist der Benachrichtigung, wie schon bei dem vorherigen Beispiel im Unterkapitel 7.5.1, ein Name und ein Auswertungsintervall zugewiesen worden. Die Konfiguration ist in der Abbildung 7.5 bildlich veranschaulicht.

#### Bedingungen

Danach werden die Bedingungen der Benachrichtigung festgelegt, welche in Unterkapitel 7.2 beschrieben werden. Die Bedingungen der Auslastung sind in der folgenden Abbildung 7.10 zu sehen.

WHEN	avg ()	OF	query (A, 5s, now)	IS ABOVE	90	
AND	avg ()	OF	query (B, 5s, now)	IS ABOVE	90	
AND	avg ()	OF	query (C, 5s, now)	IS ABOVE	90	
AND	avg ()	OF	query (D, 5s, now)	IS ABOVE	90	
AND	avg ()	OF	query (E, 5s, now)	IS ABOVE	90	
AND	avg ()	OF	query (F, 5s, now)	IS ABOVE	90	

Abbildung 7.10: Alarmbedingungen für Festplattenauslastung in Grafana GUI

## Fehlerbehandlung

Die Bedingungen für den Fall, dass keine Daten oder Fehler auftreten, verhalten sich gleich wie in der Abbildung 7.7 und entsprechend der Bedingungen in Unterkapitel 7.3.

## Benachrichtigung

Die Benachrichtigung erfolgt sofort, wie in der Abbildung 7.8 zu sehen ist. Wie auch schon im Beispiel in Unterkapitel 7.5.1 kann auch hier die Nachricht an eine Einzelpersonen geschickt werden. Der Benachrichtigungstext ist hier auf „Disk > 90“ gesetzt worden.

## JSON

Der Code für diese Beispiel sieht wie folgt aus.

```
"alert": {
    "alertRuleTags": {},
    "conditions": [
        {
            "evaluator": {
                "params": [
                    90
                ],
                "type": "gt"
            },
            "operator": {
                "type": "and"
            },
            "query": {
                "params": [
                    "A",
                    "5s",
                    "now"
                ]
            },
            "reducer": {
                "params": [],
                "type": "avg"
            },
            "type": "query"
        },
        {
            "evaluator": {
                "params": [
                    90
                ],
                "type": "gt"
            },
            "operator": {
                "type": "and"
            },
        }
    ]
}
```

```
"query": {
    "params": [
        "B",
        "1m",
        "now"
    ]
},
"reducer": {
    "params": [],
    "type": "avg"
},
"type": "query"
}, {
    "evaluator": {
        "params": [
            90
        ],
        "type": "gt"
    },
    "operator": {
        "type": "and"
    },
    "query": {
        "params": [
            "C",
            "5s",
            "now"
        ]
    },
    "reducer": {
        "params": [],
        "type": "avg"
    },
    "type": "query"
}, {
    "evaluator": {
        "params": [
            90
        ],
        "type": "gt"
    },
    "operator": {
        "type": "and"
    },
    "query": {
        "params": [
            "D",
            "5s",
            "now"
        ]
    },
    "reducer": {
        "params": []
    }
}
```

```
        "type": "avg"
    },
    "type": "query"
}, {
    "evaluator": {
        "params": [
            90
        ],
        "type": "gt"
    },
    "operator": {
        "type": "and"
    },
    "query": {
        "params": [
            "E",
            "5s",
            "now"
        ]
    },
    "reducer": {
        "params": [],
        "type": "avg"
    },
    "type": "query"
}, {
    "evaluator": {
        "params": [
            90
        ],
        "type": "gt"
    },
    "operator": {
        "type": "and"
    },
    "query": {
        "params": [
            "F",
            "5s",
            "now"
        ]
    },
    "reducer": {
        "params": [],
        "type": "avg"
    },
    "type": "query"
}
],
"executionErrorState": "alerting",
"for": "10s",
"frequency": "1s",
"handler": 1,
```

```
"message": "Disk > 90",
"name": "Client Disk Usage Alert",
"noDataState": "keep_state",
"notifications": [
    {
        "uid": "Kyi8as5Zz"
    }
]
```

---

## 8 Das Testnetzwerk

### 8.1 Anforderungen an das Netzwerk

Das Netzwerk muss Komponenten verschiedener Hersteller enthalten, damit die Netbat-Software mit den Eigenheiten von Produkten möglichst vieler verschiedener Produzenten getestet wird. Es müssen Dienste von OSI-Layer 2 bis 4 zur Verfügung gestellt werden, damit das Monitoring alle Teile eines Netzwerks als Datenquellen hat. Diese Dienste sollen möglichst realistisch belastet werden, der „Testtraffic“ soll auch zwischen den verschiedenen Subnetzen „fließen“, um Routen monitoren zu können. Für den Zugriff von außen sind ein VPN für die Wartung und ein Webserver für die Livedaten notwendig. Für die Administration sind ein Dokumentationstool und ein Kontrollserver zwecks erleichterter Konfiguration nötig.

## 8.2 Netzwerktopologie

Das Netzwerk ist physisch im Rechenzentrum der conova communications GmbH aufgebaut. Auf einem VMWare Hypervisor laufen die VMs, diese sind über OSI-Layer 2 mit der Barracuda-Firewall verbunden. Die Barracuda-Firewall ist für dieses Netzwerk der Zugang zum Internet. Logisch wird dieses Netzwerk durch die Abbildung 8.1 beschrieben.

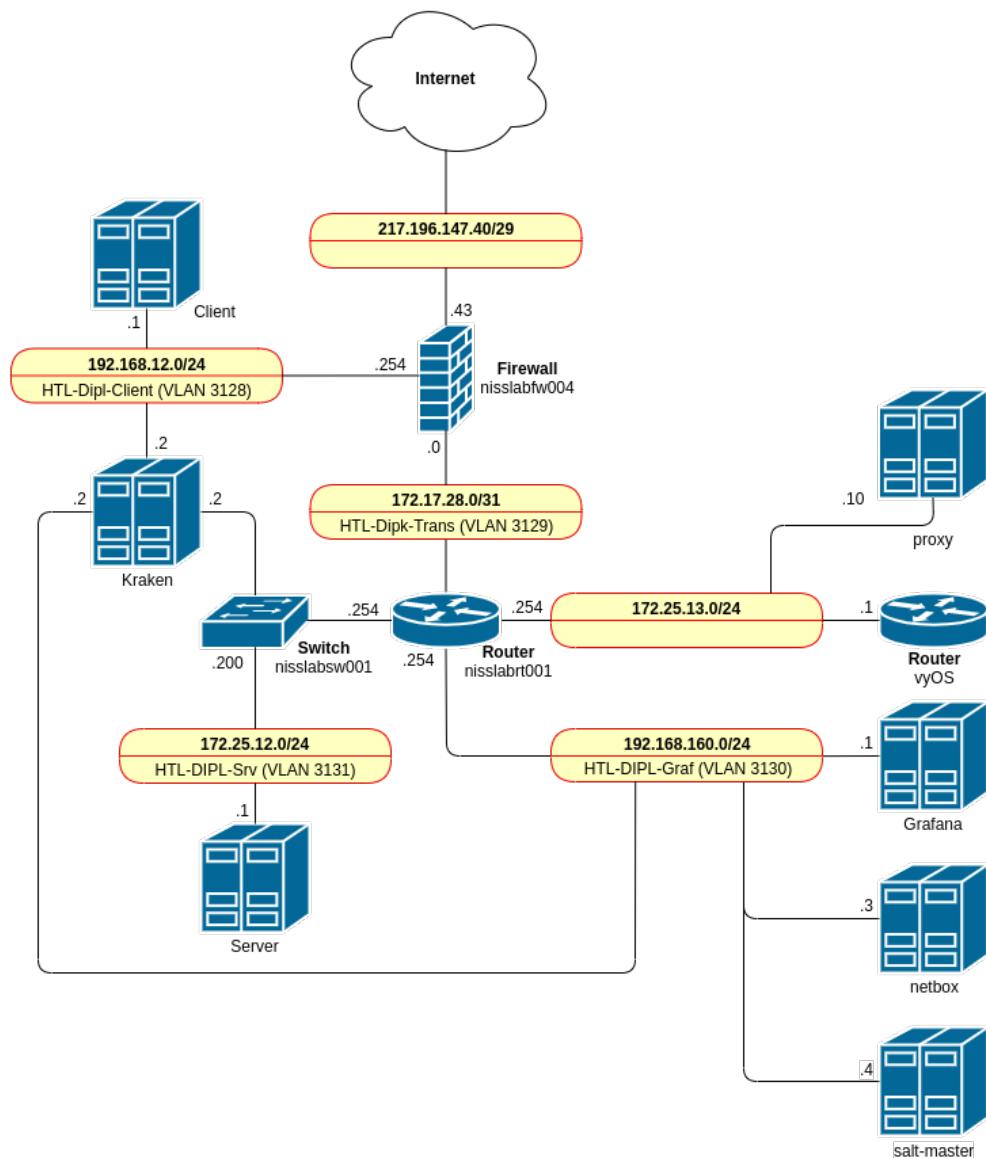


Abbildung 8.1: Topologie des Testnetzwerks

## 8.2.1 Router und Firewall

### 8.2.1.1 Barracuda

Die Barracuda-Firewall wird mit dem GUI-Tool „Firewall Admin“, welches in Version 8.0 verwendet worden ist, konfiguriert.

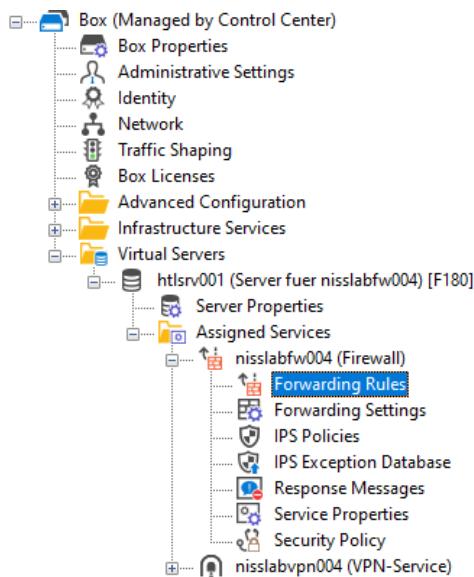


Abbildung 8.2: Das Menü des „Firewall Admin“ Tools

### Portforwarding

Da weniger öffentliche IPv4-Adressen zur Verfügung stehen, als im Netzwerk gebraucht werden, ist das Netzwerk mit privaten Adressen [21] aufgebaut. Damit trotzdem von außen auf die Dienste in diesem Netzwerk zugegriffen werden kann, werden mit NAT einzelne Ports weitergeleitet. Das sind für den Webserver 80 (HTTP) und 443 (HTTPS), siehe 8.3.2, und für den Wireguard VPN 51820, siehe 8.2.1.3.

### VPN

Promblem: Der VPN hört per default auf den Port 443; wenn ein anderer Port verwendet wird, muss 443 erst deaktiviert werden, um von einem anderen Service genutzt zu werden, dies ist in diesem Fall vergessen worden. So ist in unserem Fall nur die Portweiterleitung von Port 443 (HTTPS) und 80 (HTTP) richtig eingestellt gewesen, deswegen kamen nur HTTP-Pakete am Proxy-Server an. Dies wurde mit tcpdump festgestellt, siehe 8.3.4.2.

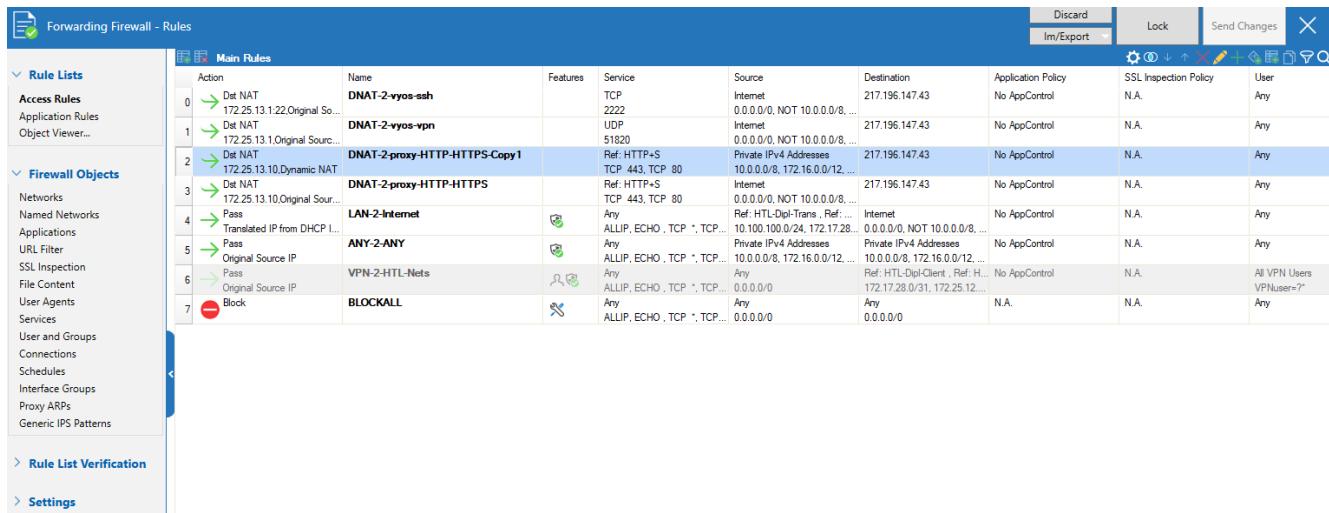


Abbildung 8.3: Die Portweiterleitungseinstellungen des „Firewall Admin“ Tools

### **8.2.1.2 Cisco**

Die „startup-config“ eines Cisco Routers beschreibt dessen Konfiguration nach einem Neustart. Ausschnitte dieser folgen nun mit Kommentaren:

Der Administratoraccount wird mit dem Namen „Admin“ und einem Passwort erstellt, dieses wurde durch einen Asterisk ersetzt.

```
username admin privilege 15 password 0 *
```

Das „GigabitEthernet0/0/1“ Interface ist mit dem Switch über ein Kabel verbunden. Die IP-Adresse und die Beschreibung des VLANs sind konfiguriert.

```
interface GigabitEthernet0/0/1
description HTL-Dipl-Srv
ip address 172.25.12.254 255.255.255.0
media-type rj45
negotiation auto
```

Die weiteren Interfaces sind nur logisch vorhanden. Die BDIs sind mit den VLAN IDs (3129, 3130 und 3133), den entsprechenden IPv4-Adressen und der passenden Netzwerkmaske konfiguriert.

```
interface BDI3129
description HTL-Dipl-Trans
ip address 172.17.28.1 255.255.255.254
```

```
interface BDI3130
description HTL-Dipl-Graf
ip address 192.168.160.254 255.255.255.0
```

```
interface BDI3133
description HTL-Dipl-Router
ip address 172.25.13.254 255.255.255.0
```

```
ip forward-protocol nd
```

Der Router ist mit einer Standardroute zu der Barracuda-Firewall konfiguriert. Bis auf das „Wireguard-VPN-Subnet“, welches über den „VyOS-Router“ erreichbar ist, sind alle weiteren Subnets direkt mit diesem Router verbunden. Um den Router zu konfigurieren, wird SSH in der Version 2 eingesetzt.

```
ip ssh version 2
ip route 0.0.0.0 0.0.0.0 172.17.28.0
ip route 10.100.100.0 255.255.255.0 172.15.13.1
```

### 8.2.1.3 VyOS

Zuerst wird das Ethernet-Interface „eth0“ mit einer IP- und MAC-Adresse konfiguriert.

```
set interfaces ethernet eth0 address '172.25.13.1/24'  
set interfaces ethernet eth0 description 'managementinterface'  
set interfaces ethernet eth0 hw-id '00:50:56:84:0b:59'  
set interfaces loopback lo
```

Für den Wireguard VPN wird ein Interface erstellt, diesem wird eine IP-Adresse zugewiesen und das Subnet der Peers (Geräte, die mit dem VPN verbunden werden) festgelegt. Damit die Verbindung aufrecht bleibt, auch wenn NAT auf dem Pfad zum Peer eingesetzt wird, was in diesem Netzwerk der Fall ist, siehe 8.2.1.1, wird „persistent-keepalive“ auf 25 gesetzt.

```
set interfaces wireguard wg0 address '10.100.100.1/24'  
set interfaces wireguard wg0 peer foo allowed-ips '10.100.100.0/24'  
set interfaces wireguard wg0 peer foo persistent-keepalive '25'
```

Der öffentliche Schlüssel des ersten Peers „foo“, welcher hier durch einen Asterisk symbolisiert wird, wird gesetzt und der Port 51820 für ihn zum Verbinden geöffnet. Es wird ein Schlüsselpaar generiert und dann genutzt. Am Peer „foo“ müssen der passende öffentliche und private Schlüssel konfiguriert sein, siehe 8.2.1.4. Zuletzt wird eine Standardroute vom VPN-Subnet zum Cisco Router konfiguriert.

```
set interfaces wireguard wg0 peer foo pubkey '*'  
set interfaces wireguard wg0 port '51820'  
run generate wireguard named-keypairs KP01  
set interfaces wireguard wg0 private-key 'KP01'  
set protocols static route 0.0.0.0/0 next-hop 172.25.13.254  
    next-hop-interface 'eth0'
```

Der Router wird mit „LAB-HTL-Dipl-vyos“ benannt. Danach wird ein Nutzer mit dem Namen „htladmin“ erstellt und erhält ein Passwort, das zuvor schon gehasht wurde (hier durch einen Asterisk ersetzt). Außerdem wird ein öffentlicher SSH-Schlüssel, der auf einer Smartcard liegt, hinzugefügt. Da der Login mit einem SSH-Schlüssel sicherer ist als mit einem Passwort, wird der Passwortlogin über SSH deaktiviert. Hier ist sicherzustellen, dass wirklich der richtige SSH-Schlüssel angegeben wurde, da man sich sonst mit diesem Schritt aus dem VyOS-Router aussperrt. Zuletzt werden die Rechte des Nutzers „htladmin“ auf den Level „admin“ gesetzt.

```
set system host-name 'LAB-HTL-Dipl-vyos'
set system login user htladmin authentication encrypted-password '*'
set system login user htladmin authentication plaintext-password ''
set system login user htladmin authentication public-keys cardno:000609925877
key '*'
set system login user htladmin authentication public-keys cardno:000609925877
type 'ssh-rsa'
set service ssh disable-password-authentication
set system login user htladmin level 'admin'
```

Damit der Router Domains auslösen kann, wird der DNS-Server konfiguriert. Damit die Logdateien des VyOS-Routers mit denen des übrigen Netzwerks verglichen werden können, werden eine Zeitzone und NTP-Server gesetzt. Außerdem wird der gewünschte Loglevel gesetzt.

```
set system name-server '46.182.19.48'
set system ntp server 0.pool.ntp.org
set system ntp server 1.pool.ntp.org
set system ntp server 2.pool.ntp.org
set system syslog global facility all level 'info'
set system syslog global facility protocols level 'debug'
set system time-zone 'UTC'
```

### 8.2.1.4 Laptop des Administrators

Auf einem Laptop mit dem Betriebssystem „Arch Linux“ wurden die Pakete „wireguard-tools“ und „wireguard-arch“ installiert, um das Wireguard-Kernel-Modul und einen einfachen Weg, Wireguard zu konfigurieren, zu erhalten.

Mit dem folgendem Befehl werden öffentliche und private Schlüssel in die jeweils passende Datei geschrieben.

```
$ wg genkey | tee privatekey | wg pubkey > publickey
```

Diese müssen nun auf dem VyOS Router, siehe 8.3.1.2, und in die Datei „/etc/wireguard/netbat.conf“ eingefügt werden. Diese Datei folgt, allerdings sind die Schlüssel durch einen Asterisk ersetzt:

[Interface]

```
Address = 10.100.100.2/24
```

```
PrivateKey = *
```

[Peer]

```
PublicKey = *
```

```
AllowedIPs = 0.0.0.0/0
```

```
Endpoint = 217.196.147.43:51820
```

Nun kann die VPN-Verbindung mit folgendem Befehl aufgebaut werden:

```
# sudo wg-quick up netbat
```

Das Tool „wg-quick“ führt folgende Befehle aus:

```
# ip link add netbat type wireguard
# wg setconf netbat /dev/fd/63
# ip -4 address add 10.100.100.2/24 dev netbat
# ip link set mtu 1420 up dev netbat
# wg set netbat fwmark 51820
# ip -4 route add 0.0.0.0/0 dev netbat table 51820
# ip -4 rule add not fwmark 51820 table 51820
# ip -4 rule add table main suppress_prefixlength 0
# sysctl -q net.ipv4.conf.all.src_valid_mark=1
# iptables-restore -n
```

## 8.2.2 Server (VMs)

Alle Server in diesem Netzwerk sind virtuelle Maschinen, auf welchen Debian 9 Stretch als Betriebssystem eingesetzt wird, obwohl Debian 10 Buster die aktuellste Version zu dem Zeitpunkt ist, als dieses Netzwerk aufgebaut worden ist. Diese Entscheidung wurde getroffen, da ein Versionsupgrade Komplikationen mit sich bringen könnte und Neuerungen von Buster nicht gebraucht werden. [22]

Die Directoys „/“, „/home“, „/tmp“ und „/var“ werden in verschiedene LVM-Volumes gemountet, um bei einem plötzlichen Anstieg von Daten nur ein überfülltes Directory und nicht einen nicht mehr verwendbaren Rechner zu haben.

Um die VMs leichter zu administrieren, wird auf allen VMs das Paket „salt-minion“ installiert und in die Datei „/etc/salt/minion“ die IP-Adresse des „Salt-Servers“ als „Master“ eingetragen.

master: 192.168.160.4

Um diese Änderung zu aktivieren, muss der Service „salt-minion.service“ neu gestartet werden. Die Konfiguration des Salt Masters ist in 8.2.2.7 beschrieben.

### 8.2.2.1 *Grafana*

Auf der „Grafana-VM“ laufen die InfluxDB, die Grafana-Software und die NetBat-Software.

Die Influx DB kann mit apt installiert werden, da sich das Paket in den Debian Stretch Repositorys befindet. Danach muss der Service „influxdb.service“ gestartet werden.

Da sich die neueste Version von Grafana nicht in den Debian Stretch Repositorys befunden hat, ist das Repository von Grafana hinzugefügt. Dafür muss der Schlüssel von Grafana ([packages.grafana.com/gpg.key](https://packages.grafana.com/gpg.key)) zu den vertrauten hinzugefügt werden. Danach muss das Repository hinzugefügt werden. Dies ist mit folgendem Befehl möglich:

```
$ sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"
```

Nachdem die Repositorys upgedatet worden sind, kann Grafana mit apt installiert werden. Danach kann der Service „grafana-server“ mit systemctl gestartet werden. [23]

Die NetBat Software wird mit „git pull“ in das Directory „/srv/netbat/“ geladen.

Probleme: Die NetBat-Software zeigte an, alle Geräte wären offline, da sie scheinbar nicht mehr pingbar waren. Da die Geräte allerdings noch online waren und einwandfrei funktionierten, musste ein Problem an der „Grafana-VM“ vorliegen. Auf diesem eingelogt, gab der ping-Befehl immer folgende Meldung aus:

```
connect: Resource temporarily unavailable
```

Nach einem Neustart der „Grafana-VM“ war das Problem scheinbar gelöst. Nach 12 bis 24 Stunden trat das Problem wieder auf, ließ sich allerdings mit einem Neustart des Services „pm2-root“ lösen. Dies legte einen Bug in der NetBat-Software nahe. Mit dem Befehl „netstat -alpn | grep node | less | wc -l“ wurde erkannt, dass NetBat die maximale Anzahl an UDP-Ports überschritt. Damit sich in Zukunft nur NetBat aufhängt und nicht alle Programme, die von NetBat genutzt werden, sind in der Datei „/etc/systemd/system/pm2-root.service“ die drei Zeilen mit „Limit =infinity“ auskommentiert.

```
[Service]
Type=forking
User=root
#LimitNOFILE=infinity
#LimitNPROC=infinity
#LimitCORE=infinity
Environment=PM2_HOME=/root/.pm2
PIDFile=/root/.pm2/pm2.pid
Restart=on-failure
```

Da in der Datenbank viele Werte gespeichert werden, siehe 4, und sich diese im Directory „/var“ befand, wurde dieses Directory überfüllt. Dies lag zum einen an einem zu langsamen Reagieren des Administrators und zum anderen an einem noch nicht funktionierenden Monitoring für die Festplatten- auslastung. Die Trennung in einzelne „Volumegroups“ sorgte dafür, dass in so einem Fall nur ein Riss in den Logdateien entstand und nicht der Rechner abstürzte. Das Problem wurde behoben, indem der VM ein größerer Speicherplatz zugeteilt wurde und die NetBat-Software um eine Funktion erweitert wurde, die alte Daten komprimiert.

## Telegram

Die Grafana-Software liefert einen Telegram-Bot mit, dieser benötigt allerdings ein „BOT API Token“, welches mit folgender Telegram-Nachricht an „<https://t.me/botfather>“ generiert werden kann:

/newbot

Weiters ist eine „Chat ID“ nötig, um das Benachrichtigen über Telegram zu aktivieren. Diese ist nur vorhanden, wenn der Bot zuvor die Nachricht „/start“ von einem Nutzer erhalten hat. Dann erhält man sie auf der Website „[https://api.telegram.org/bot\\*/getUpdates](https://api.telegram.org/bot*/getUpdates)“, welche die Interaktion mit dem Telegram-Bot als JSON ausgibt und den die Chat-ID unter „result.0.message.from.id“ angibt.

### 8.2.2.2 Proxy

Der HAProxy soll das Grafana-Dashboard aus dem Internet über standardisierte Ports (80&443) erreichbar machen. Da Passwörter übertragen werden, ist eine TLS-verschlüsselte Verbindung nötig, um diese für Angreifer unleserlich zu machen, allerdings kann Grafana keine TLS-Verschlüsselung aufbauen und läuft auf dem TCP-Port 3000.

Deshalb kommuniziert dieser Server verschlüsselt mit Nutzern über das Internet und unverschlüsselt mit dem der „Grafana-VM“.

Da sich das Paket „haproxy“ in den Debian Stretch Repositorys befindet, kann es mit apt installiert werden. Die Version von HAProxy kann mit dem Befehl „haproxy -v“ ausgegeben werden. Als dieses Dokument geschrieben wurde, wurde auf dem Server die Version 1.7.5-2 betrieben. Die Konfiguration von HAProxy wird in „/etc/haproxy/haproxy.cfg“ gespeichert, welche nachfolgend zu sehen ist. Nach einer Änderung dieser Datei muss der HAProxy-Service neu gestartet werden.

Es wird ein „frontend“ auf dem Port 80 (HTTP) hinzugefügt, welches auf jede IP-Adresse hört. Es leitet auf HTTPS weiter, außer wenn eine „letsencrypt challenge“ angefragt wird. Dann leitet es zum „backend“ „letsencrypt-backend“ weiter. Auf Port 443 wird ein „frontend“ hinzugefügt, welches eine TLS-verschlüsselte Verbindung mit den Schlüsseln, die in „/etc/haproxy/certs/netb.at.pem“ gespeichert sind, aufbaut. Außerdem leitet es zum „www-backend“ weiter.

```
frontend www-http
bind *:80
reqadd X-Forwarded-Proto:\ http
default_backend www-backend
```

```
frontend www-https
bind *:443 ssl crt /etc/haproxy/certs/netb.at.pem

#      reqadd X-Forwarded-Proto:\ https

acl letsencrypt-acl path_beg /.well-known/acme-challenge/
use_backend letsencrypt-backend if letsencrypt-acl
default_backend www-backend
```

Das „www-backend“ leitet HTTP-Anfragen auf HTTPS-Anfragen weiter. HTTPS-Anfragen leitet es an die „Grafana-VM“ auf Port 3000 weiter. Das „letsencrypt-backend“ leitet Anfragen auf den Port 54321 weiter.

```
backend www-backend
redirect scheme https if !{ ssl_fc }
server grafana 192.168.160.1:3000 check
```

Ein Weg, ein Zertifikat zu erlangen, welchem alle modernen Browser vertrauen, ist mit Let's Encrypt. Dieses stellt das Programm Certbot zur Verfügung, welches ebenfalls in den Debian Stretch Repositories zu finden ist und deswegen mit apt installiert werden kann. Mit dem folgenden Befehl werden die Zertifikatsdateien in dem Directory „/etc/letsencrypt/live/netb.at/“ abgelegt:

```
# certbot certonly --standalone --prefered-challenges
http-01-port 80 pd netb.at
```

Problem: Nachdem das Zertifikat ausgestellt worden war, konnte zwar lokal eine Verbindung hergestellt werden, über die Domain allerdings nicht, siehe 8.2.1.1. Da das Zertifikat auf die Domain und nicht auf die Lokale IP-Adresse ausgestellt worden ist, ist es ungültig. Um zu überprüfen, ob das Zertifikat gültig ist, wurde an einem Rechner, der mit dem Testnetz verbunden ist, der Eintrag für die Domain „netb.at“ in der „/etc/hosts“ mit „172.25.13.10“ überschrieben.

### 8.2.2.3 Server und Client

Wie in Abbildung 8.1 zu sehen ist, liegen die zwei VMs „Server“ und „Client“ an zwei unterschiedlichen Stellen in dem Testnetzwerk. Deswegen kann mit Traffic, der zwischen diesen beiden VMs übertragen wird, das Netzwerk getestet werden, da die „Barracudafirwall“, der „Cisco-Router“ und der „Cisco-Switch“ belastet werden.

### Netzwerktesten mit hping3

Mit folgendem Befehl, welcher auf der „Server-VM“ ausgeführt wird, werden so viele ICMP Pakete an die „Client-VM“ verschickt, wie es die CPU der „Server-VM“ zulässt. Mehr Informationen zu hping unter 8.3.3.1.

```
hping3 --flood -1 192.168.12.1
```

In Abbildung 8.4 ist das Grafana Dashboard zu sehen, welches die RTT und die CPU-Auslastung für die „Server-VM“ und die „Client-VM“ anzeigt. Für den Cisco-Router werden der Status und der Durchsatz für das Interface 1 und 2 angezeigt. In dieser Abbildung ist zu sehen, dass hping3 in diesem Modus zwar die CPU des Servers völlig auslastetet, das Netzwerk aber noch freie Kapazitäten hat.

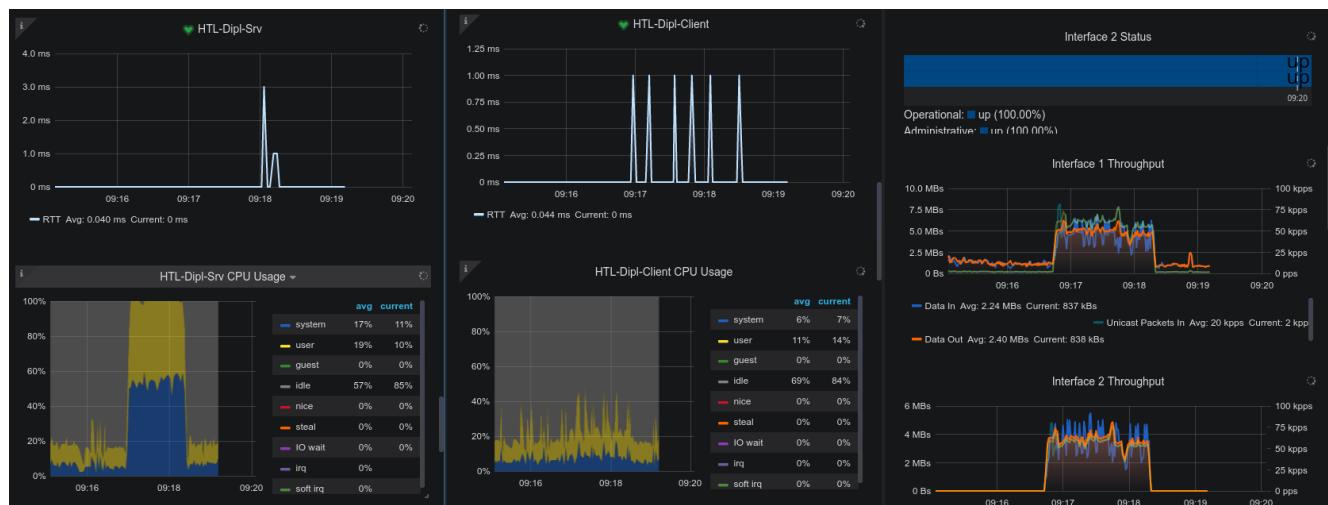


Abbildung 8.4: Belastungen bei einem Netzwerktest mit hping3

Da allerdings die maximale Auslastung des Netzwerks und nicht der CPU getestet werden soll, muss eine andere Lösung gefunden werden.

## Netzwerktesten mit iperf

Das in 8.3.3.2 näher beschriebene Tool iperf kann das Netzwerk effizienter auslasten als hping3. So wird im folgenden auf der „Server-VM“ der iperf Server gestartet:

```

1 # iperf -s
2 -----
3 Server listening on TCP port 5001
4 TCP window size: 85.3 KByte (default)
5 -----
6 [ 4] local 172.25.12.1 port 5001 connected with 192.168.12.1 port 42152
7 [ ID] Interval Transfer Bandwidth
8 [ 4] 0.0-10.0 sec 1.06 GBytes 909 Mbits/sec

```

Auf der „Client-VM“ wird der iperf Client ausgeführt.

```

1 # iperf -c 172.25.12.1
2 -----
3 Client connecting to 172.25.12.1, TCP port 5001
4 TCP window size: 85.0 KByte (default)
5 -----
6 [ 3] local 192.168.12.1 port 42152 connected with 172.25.12.1 port 5001
7 [ ID] Interval Transfer Bandwidth
8 [ 3] 0.0-10.0 sec 1.06 GBytes 911 Mbits/sec

```

Abbildung 8.5 zeigt das Grafana Dashboard, welches die RTT und die CPU-Auslastung für die „Server-VM“ und die „Client-VM“ anzeigt. Für den Cisco-Router werden der Status und der Durchsatz für das Interface 1 und 2 angezeigt. Es ist zu erkennen, dass iperf den Netzwerkdurchsatz vollkommen auslastet, während die CPU nicht viel mehr ausgelastet ist.

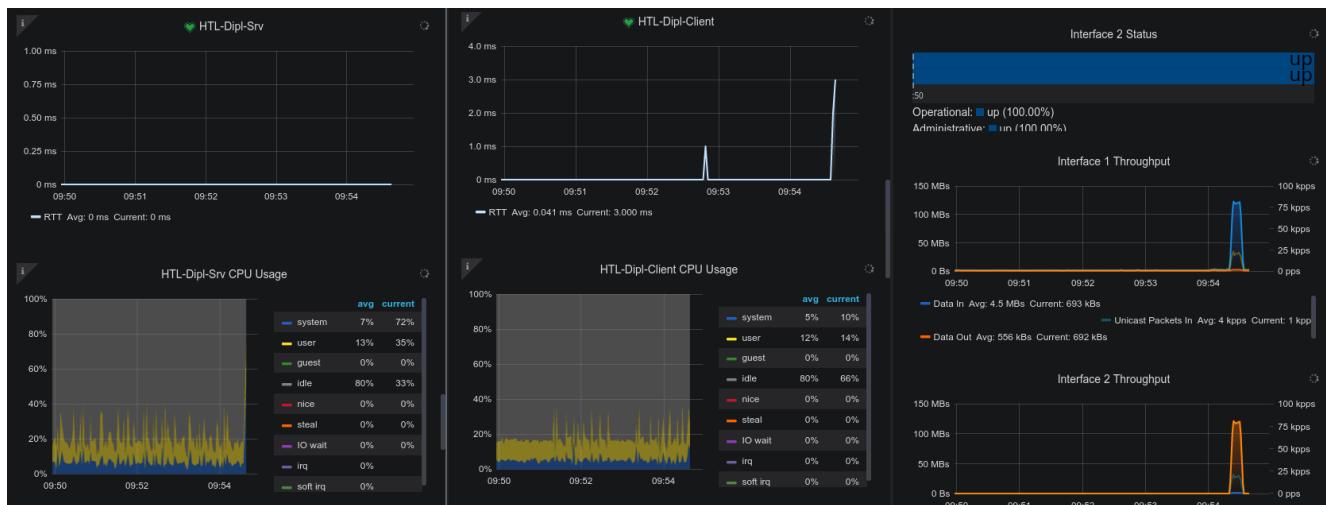


Abbildung 8.5: Belastungen bei einem Netzwerktest mit iperf

### 8.2.2.4 Kraken

Da das Netzwerk unter möglichst reeller Last getestet werden soll, wird auf der „Kraken-VM“ TRex von Cisco eingesetzt. Mehr zu diesem Programm unter 8.3.3.3. Da diese VM über SSH gesteuert wird und TRex zwei Netzwerkkarten braucht, verfügt die „Kraken-VM“ über drei Netzwerkkarten, wie in der Abbildung 8.1 zu sehen ist.

## Installation

Um die neueste Version von TRex zu installieren, wird in das Directory „/opt/trex“ der Inhalt von „<https://trex-tgn.cisco.com/trex/release/latest>“ geladen. Im Testnetz wird die Version v2.73 eingesetzt.

Um das Kernel-Modul zu installieren, welches für den eingesetzten Kernel benötigt wird, werden folgende Befehle ausgeführt:

```
# apt install linux-headers-`uname -r` build-essential  
# mkdir -p /tmp/trex-ko  
# cp -r ./ko/src /tmp/trex-ko  
# cd /tmp/trex-ko/src  
# make  
# make install
```

Nun kann das Programm „setup ports“ ausgeführt werden, welches die letzten benötigten Komponenten selbstständig kompiliert, nachdem es erkannt hat, dass diese nicht vorhanden sind.

---

```
1  # ./dpdk_setup_ports.py  
2  Trying to bind to vfio-pci ...  
3  Trying to compile and bind to igb_uio ...  
4  ERROR: We don't have precompiled igb_uio.ko module for your kernel version  
5  Will try compiling automatically...  
6  Success.  
  
8  /usr/bin/python3 dpdk_nic_bind.py --bind=igb_uio 0000:0b:00.0 0000:13:00.0  
9  The ports are bound/configured.
```

---

Mit folgendem Befehl werden alle verfügbaren Ports ausgegeben. Der Port, welcher mit „\*Active“ gekennzeichnet ist, darf nicht verwendet werden. Dies ist jener, über den mit SSH zugegriffen wird. Falls dies doch geschieht, muss auf die VM mit der Konsole zugegriffen werden.

---

```
1 # ./dpdk_setup_ports.py -s

3     Network devices using DPDK-compatible driver
4 =====
5     0000:0b:00.0 'VMXNET3 Ethernet Controller' drv=igb_uio unused=vmxnet3,vfio-
6         pci,uio_pci_generic
7     0000:13:00.0 'VMXNET3 Ethernet Controller' drv=igb_uio unused=vmxnet3,vfio-
8         pci,uio_pci_generic

9     Network devices using kernel driver
10 =====
11    0000:1b:00.0 'VMXNET3 Ethernet Controller' if=ens256 drv=vmxnet3 unused=
12        igb_uio,vfio-pci,uio_pci_generic *Active*
```

---

In der Datei „/etc/trex\_cfg.yaml“, welche nachfolgend zu sehen ist, wird die Defaultkonfiguration der Ports angegeben. Wenn das Netzwerk wie in unserem Fall mit dem Internet verbunden ist, müssen private oder IP-Adressen, welche einem selbst gehören, angegeben werden, da sonst eine DOS-Attacke auf den Besitzer der angegebenen IP -Adressen mit TRex ausgeführt wird:

---

```
1 - version: 2
2 interfaces: ['0b:00.0', '13:00.0']
3 port_info:
4 - ip: 172.25.12.2
5 default_gw: 172.25.12.254
6 - ip: 192.168.12.2
7 default_gw: 192.168.12.254

9 platform:
10 master_thread_id: 0
11 latency_thread_id: 1
12 dual_if:
13 - socket: 0
14 threads: [2,3]
```

---

Diese Datei kann auch mit folgendem Befehl erstellt werden:

```
# ./dpdk_setup_ports.py -i
```

## Testen mit CLI

Mit folgendem Befehl, welcher im Directory „/opt/trex/v2.73“ ausgeführt werden muss, werden zum ersten Mal Pakete mit TRex verschickt.

```
# ./t-rex-64 -f cap2/dns_one_server.yaml -m 80 -d 10
```

Der Parameter „-m 80“ multipliziert die versendeten Daten auf das 80-fache. Der Parameter „-d 10“ lässt den Test 10 Sekunden dauern. Im mit TRex mitgelieferten Test „cap2/simple-http.yaml“ werden die IP-Adressen wie folgt bearbeitet:

---

```
1 - duration : 0.1
2   generator :
3     distribution : "seq"
4     clients_start : "192.168.12.2"
5     clients_end   : "192.168.12.254"
6     servers_start : "172.25.12.2"
7     servers_end   : "172.25.12.254"
8     #clients_start : "16.0.0.1"
9     #clients_end   : "16.0.0.255"
10    #servers_start : "48.0.0.1"
11    #servers_end   : "48.0.255.255"
12    clients_per_gb : 201
13    min_clients   : 101
14    dual_port_mask : "1.0.0.0"
15    tcp_agging    : 0
16    udp_agging    : 0
17  cap_ipg      : true
18  cap_info :
19    - name: avl/delay_10_http_browsing_0.pcap
20      cps : 2.776
21      ipg : 10000
22      rtt : 10000
23      w   : 1
```

---

Dann wird er mit folgendem Befehl ausgeführt:

```
# ./t-rex-64 -f cap2/http_simple.yaml -m 10 -d 10
```

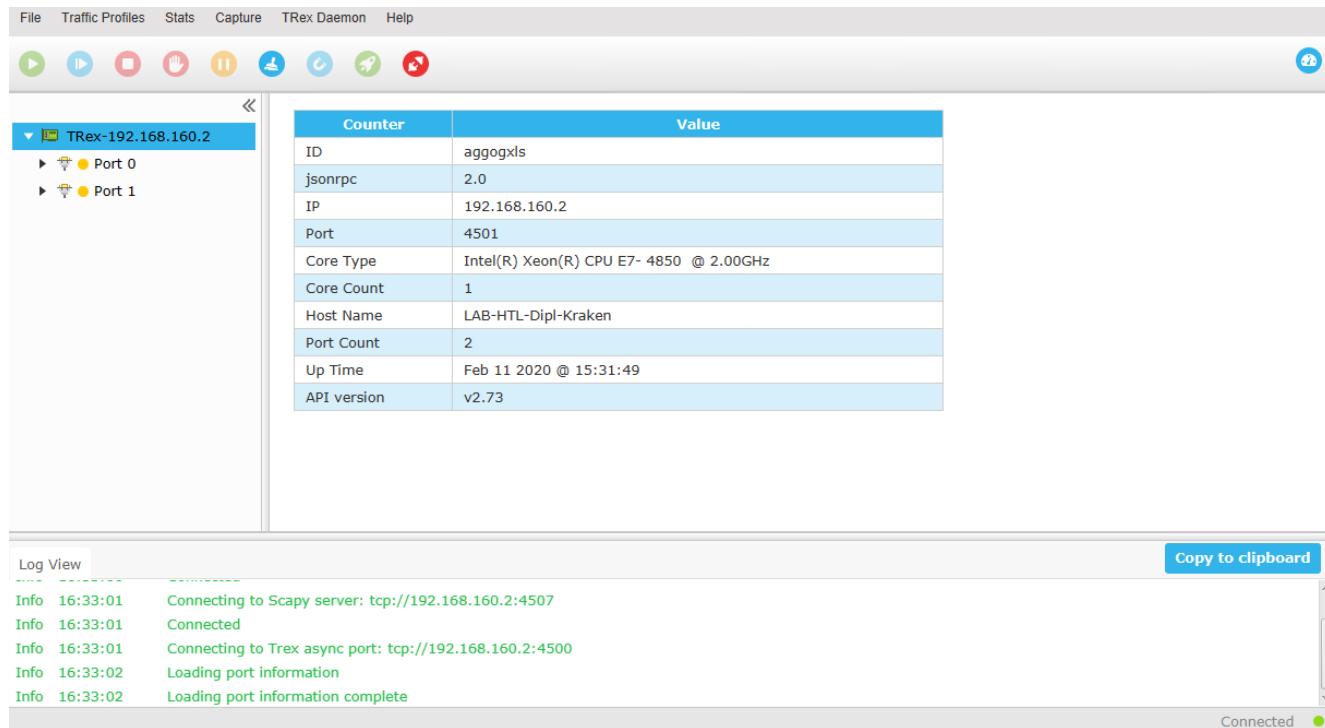


Abbildung 8.6: Die grafische Benutzeroberfläche von TRex

### 8.2.2.5 Testen mit GUI

Für TRex wird unter „<https://github.com/cisco-system-traffic-generator/trex-stateless-gui>“ auch eine grafische Benutzeroberfläche zur Verfügung gestellt. Auf der „Kraken-VM“ wird mit folgendem Befehl ein Server geöffnet, mit dem sich die GUI, welche auf dem Laptop des Administrators gestartet wird, verbinden kann:

```
# ./t-rex-64 -i
```

In der GUI wird unter „File“ der Reiter „Connect“ ausgewählt. In dem sich öffnenden Dialogfenster wird die IP-Adresse der „Kraken-VM“ (192.168.160.2) eingegeben. Danach kann mit der in Abbildung 8.6 zu sehenden Benutzeroberfläche weiter gearbeitet werden.

<input type="checkbox"/>	Name	Status	Cluster	Role	Tenant	VCPUs	Memory (MB)	Disk (GB)	IP Address
<input type="checkbox"/>	Client	Active	LAB-HTL-Dipl	Server	—	1	2000	16	192.168.12.1
<input type="checkbox"/>	grafana	Active	LAB-HTL-Dipl	Server	—	4	2000	100	192.168.160.1
<input type="checkbox"/>	Kraken	Active	LAB-HTL-Dipl	Server	—	4	7999	16	192.168.160.2
<input type="checkbox"/>	Netbox	Active	LAB-HTL-Dipl	Server	—	1	2000	16	192.168.160.3
<input type="checkbox"/>	Proxy	Active	LAB-HTL-Dipl	Server	—	1	2000	16	172.25.13.10
<input type="checkbox"/>	salt-master	Staged	LAB-HTL-Dipl	Server	—	1	2000	16	192.168.160.4
<input type="checkbox"/>	Server	Active	LAB-HTL-Dipl	Server	—	1	2000	16	172.25.12.1
<input type="checkbox"/>	VyOs	Active	LAB-HTL-Dipl	Router	—	1	2000	17	172.25.13.1

[+ Add Components](#) [Edit Selected](#) [Delete Selected](#)
Showing 1-8 of 8

Abbildung 8.7: Liste aller VMs in der GUI von NetBox.

### 8.2.2.6 NetBox

Es wurden alle Schritte auf „<https://netbox.readthedocs.io/en/stable/installation/>“ befolgt. Danach steht ein Webserver auf Port 80 (HTTP) zur Verfügung. Über diesen kann NetBox bedient werden. Dort wurden über die Website (GUI) alle Daten zum Netzwerk eingetragen.

In Abbildung 8.7 ist die Übersicht der VMs in NetBox zu sehen. Es sind die CPU-Kerne, der Arbeitsspeicher, die Festplattenkapazität und die wichtigste IP-Adresse zu sehen.

Abbildung 8.8 zeigt die genaueren Informationen über die „Proxy-VM“. In dieser Ansicht sind auch die einzelnen Services zu sehen und über welchen Port sie erreichbar sind. Hier können diese auch bearbeitet, gelöscht oder hinzugefügt werden.

The screenshot displays the NetBox interface for managing a virtual machine. It is organized into three main sections: Resources, Services, and Interfaces.

**Resources:**

Resource Type	Value
Virtual CPUs	1
Memory	2000 MB
Disk Space	16 GB

**Services:**

Service Name	Protocol & Port	IP Address	Action Buttons
ssh	TCP/22	All IPs	[Edit] [Delete]
http	TCP/80	172.25.13.10	[Edit] [Delete]
munin	TCP/4949	172.25.13.10	[Edit] [Delete]

**Interfaces:**

Name	LAG	Description	Mode	Connection	Status	Action Buttons
ens192	—	—	—	—	Virtual interface	[Add] [Edit] [Delete]
IP Address		Status/Role	VRF	Description		
172.25.13.10/24		Active	Global	—		[Edit] [Delete]

Buttons at the bottom of the interfaces section include: + Add interfaces, Rename, Edit, and Delete.

Abbildung 8.8: VM-Detailansicht in der GUI von NetBox.

### 8.2.2.7 *Salt master*

Auf der „Salt-Master-VM“ ist das sich in den Repositorys befindende „salt-master“ Paket installiert. Um mit den Minions zu kommunizieren, muss auf diesen das „salt-minion“ Paket installiert werden und die Salt Master VM als Master angegeben sein, siehe 8.2.2. Mit folgendem Befehl werden alle Debian Rechner, die sich von dem Salt Master steuern lassen, aufgelistet.

```
#salt-key -L
```

Accepted Keys:

```
LAB-HTL-Dipl-Client  
LAB-HTL-Dipl-Grafana  
LAB-HTL-Dipl-Kraken  
LAB-HTL-Dipl-Netbox  
LAB-HTL-Dipl-Server  
LAB-HTL-Dipl-proxy
```

Denied Keys:

Unaccepted Keys:

Rejected Keys:

Wenn es „Unaccepted Keys“ gibt, können diese mit „salt-key -A“ zu den „Accepted Keys“ hinzugefügt werden, welche auf die Konfigurationsbefehle und Dateien des Salt Masters zugreifen können und diese auch umsetzen. [24]

Der Salt Master kann nun Befehle auf einem, mehreren oder allen Minions gleichzeitig ausführen, zum Beispiel Updates auf allen Rechnern einspielen: [25]

```
# salt -G 'os:Debian' cmd.run 'apt update && apt upgrade -y'
```

Wenn gleiche Aktionen mehrmals ausgeführt werden müssen, können „States“ angelegt werden, welche bestimmte Zustände herstellen. Diese werden am Salt Master in dem Directory „/srv/salt/“ gespeichert und in YAML-Syntax geschrieben.

Der folgende „State“ „tools.sls“ installiert die Pakete rsync, curl, vim, tmux, zsh und mtr-tiny.

---

```
1 install_packages:
2     pkg.installed:
3         - pkgs:
4             - rsync
5             - curl
6             - vim
7             - tmux
8             - zsh
9             - mtr-tiny
```

---

Der folgende „State“, „sshkey.sls“ fügt den Inhalt der Datei „sshkeyHair“ am Ende der Datei „root/.ssh/authorized\_keys“ hinzu, falls dieser noch nicht vorhanden ist. Da sich in der „sshkeyHair“ der öffentliche SSH-Schlüssel des Administrators befindet, kann sich dieser nun ohne ein Passwort auf den entsprechenden Minions als root einloggen. [25]

---

```
1 /root/.ssh/authorized_keys:
2     file.append:
3         - sources:
4             - salt://sshkeys/sshkeyHair
```

---

## 8.3 Verwendete Programme und Alternativen

### 8.3.1 Betriebssysteme

#### 8.3.1.1 Cisco IOS

IOS von Cisco ist ein proprietäres Betriebssystem, das auf den meisten Produkten des Netzwerkherstellers Cisco zum Einsatz kommt. Änderungen treten sofort in Kraft. Damit diese nach dem Neustart eines Gerätes bestehen bleiben, muss die „running-config“ in die „startup-config“ kopiert werden. [26]

#### 8.3.1.2 VyOS

Im Gegensatz zum Cisco IOS treten Änderungen bei VyOS nicht sofort in Kraft, sondern müssen erst mit dem Befehl „commit“ aktiv geschalten werden. Wenn die Änderung die gewünschte Wirkung erzielt, werden mit dem Befehl „save“ die Änderungen dauerhaft gespeichert. Falls die Änderungen nicht wie gewünscht funktionieren, wird mit einem Neustart des Rechners VyOS auf den letzten gespeicherten Stand zurückgesetzt. Dies ist besonders hilfreich, wenn VyOS über SSH konfiguriert wird und der Router mit der neuen Konfiguration nicht erreichbar ist.[27]

### 8.3.2 Webserver und Proxys

Ein Web-Proxy empfängt Daten von einem Client, verarbeitet sie und leitet sie dann weiter. Laut Netcraft, einem Forschungs- und Sicherheitsunternehmen, welches monatlich den Marktanteil von Webservern veröffentlicht, hatte Apache im Januar 2020 35% und nginx 32%. [28] [29]

### 8.3.2.1 *Apache*

Apache zeichnet eine modulare Architektur aus. Dynamische Module können geladen und weitere Authentifizierungsmodi, Sicherheitsverbesserungen oder andere Funktionen hinzugefügt werden. Apache wird mit „Directives“ konfiguriert. [30]

### 8.3.2.2 *NGINX*

Da ein Webserver häufig länger auf die Antwort von einer Festplatte wartet, als er eigentliche Arbeit verrichtet, verwendet NGINX ein Event-basiertes System, bei welchem ein paar Arbeitsprozesse alle Anfragen gleichzeitig abarbeiten. Wenn eine Antwort von einer Festplatte kommt, ein Event, verarbeitet dies schnell ein „Arbeiter“, nur um danach sofort ein weiteres Event zu bearbeiten. [31]

### 8.3.2.3 *HAProxy*

HAProxy ist die am weitesten verbreitete Open Source „load-balancing“-Software. Es unterstützt TLS, IPv6 und viele weitere Funktionen. [31]

## 8.3.3 Netzwerktraffic generieren

### 8.3.3.1 *hping3*

Hping3 verschickt eine beliebige Anzahl an TCP/IP Paketen, soweit dies die CPU und das Netzwerk zulassen. [32]

### 8.3.3.2 *iperf*

Mit iperf kann der Netzwerkdurchsatz getestet werden. Dafür wird auf zwei Rechnern, zwischen denen der Durchsatz getestet werden soll, iperf gestartet, auf dem einen als Server (iperf -s), auf dem anderen als Klient (iperf -c „Server ip“). Ohne weitere Angaben nutzt iperf TCP, mit dem Parameter „-u“ werden UDP Pakete verschickt. [33]

### 8.3.3.3 *TRex*

TRex ermöglicht es, das Netzwerk mit zuvor aufgezeichneten Paketmitschnitten zu belasten. Dies ermöglicht nicht nur den theoretisch höchsten Durchsatz, wie mit iperf zu ermitteln, sondern den tatsächlichen. Da, um das Monitoring zu testen, das Netzwerk bis über seine Grenzen belastet werden muss, ist ein solches Werkzeug nötig.

## 8.3.4 Kommandozeilen Werkzeug

### 8.3.4.1 *Terminal*

#### **tmux**

Tmux ist ein Terminal Multiplexer. Es können mehrere Terminals auf einem Bildschirm geöffnet werden. Zusätzlich kann ein Terminal im Hintergrund laufen oder von mehreren SSH Sessions gleichzeitig verwendet werden. [34]].

#### **zsh**

Zsh ist ein UNIX-Command-Interpreter (eine Shell). Es ist nicht kompatibel mit POSIX oder anderen Shells. Zsh verfügt unter anderem über eine Kommandovervollständigung und einen History-Mechanismus. [35]

### 8.3.4.2 *Netzwerk debugging*

#### **ping, traceroute und mtr**

Alle diese Tools versenden ICMP-Pakete, Ping immer an dieselbe IP-Adresse. Traceroute sendet ICMP-Pakete an die gegebene IP-Adresse mit steigender TTL. Dadurch erhält man eine Antwort eines jeden Routers, der zwischen dem sendenden und dem empfangenden Server liegt. Mtr (My traceroute) macht das selbe wie Traceroute, mit dem Unterschied, dass es die ICMP-requests mehrmals schickt. Es zeigt die durchschnittliche Zeit an, die ein Paket von und zu jedem der Hosts braucht und wieviel Prozent der Pakete bei einem Host verlorengehen.

## tcpdump

Tcpdump gibt eine Beschreibung der Pakete, die an einem Netzwerkinterface anliegen, auf der Konsole aus. Es ist in den Stretch Repositorys verfügbar und deswegen mit apt installierbar. Da die Gesamtheit der Pakete meist zu unübersichtlich ist, können diese weiter gefiltert werden, etwa nur bestimmte Ports „tcpdump port 80 or Port 443“ für HTTP(S) Traffic, der auf die Standard Ports geschickt wird. [36]

### 8.3.4.3 Configuration Management

Um die Konfiguration auf den Server zu vereinfachen, wird ein „Configuration Management System“ eingesetzt. Die am weitesten verbreiteten sind: Chef, Puppet, Ansible und Salt.

Chef ist erst ab mehreren hundert oder tausend zu konfigurierenden Servern zu empfehlen, weswegen es für dieses Projekt nicht geeignet ist. Puppet ist relativ alt und bringt daher Altlasten mit sich, welche für dieses Projekt auch nicht förderlich sind. Salt wurde gewählt, weil es der Administrator davor schon einmal eingesetzt hatte und es keine Nachteile gegenüber Ansible hat.[37]

## Literaturverzeichnis

- [1] J. Postel, *Internet Control Message Protocol*, RFC 792 (Internet Standard), RFC, Updated by RFCs 950, 4884, 6633, 6918, Fremont, CA, USA: RFC Editor, 1981-09. doi: 10.17487/RFC0792. [Online]. Adresse: <https://www.rfc-editor.org/rfc/rfc792.txt> [besucht am 2020-02-23].
- [2] O. Jacobsen und D. Lynch, *A Glossary of Networking Terms*, RFC 1208 (Informational), RFC, Fremont, CA, USA: RFC Editor, 1991-03. doi: 10.17487/RFC1208. [Online]. Adresse: <https://www.rfc-editor.org/rfc/rfc1208.txt> [besucht am 2020-02-23].
- [3] J. Case, M. Fedor, M. Schoffstall und J. Davin, *Simple Network Management Protocol (SNMP)*, RFC 1157 (Historic), RFC, Fremont, CA, USA: RFC Editor, 1990-05. doi: 10.17487/RFC1157. [Online]. Adresse: <https://www.rfc-editor.org/rfc/rfc1157.txt> [besucht am 2020-02-23].
- [4] J. Case, K. McCloghrie, M. Rose und S. Waldbusser, *Introduction to Community-based SNMPv2*, RFC 1901 (Historic), RFC, Fremont, CA, USA: RFC Editor, 1996-01. doi: 10.17487/RFC1901. [Online]. Adresse: <https://www.rfc-editor.org/rfc/rfc1901.txt> [besucht am 2020-02-23].
- [5] R. Olbricht, *Overpass API User's Manual*, 2019. [Online]. Adresse: <https://dev.overpass-api.de/overpass-doc/en/index.html> [besucht am 2020-02-23].
- [6] Grafana Labs, *Grafana Documentation*, 2020. [Online]. Adresse: <https://grafana.com/docs/grafana/latest> [besucht am 2020-02-23].
- [7] The Munin project and its contributors, *Munin Documentation*, Version 2.999.10, 2018. [Online]. Adresse: [http://guide.munin-monitoring.org/\\_/downloads/en/latest/pdf/](http://guide.munin-monitoring.org/_/downloads/en/latest/pdf/) [besucht am 2020-02-23].
- [8] ISO/IEC JTC 1/SC 32, „Information technology — Database languages — SQL,“ International Organization for Standardization, Geneva, CH, Standard, 2016-12.

- [9] InfluxData Inc., *InfluxDB 1.7 documentation*, Version 1.7, 2020. [Online]. Adresse: <https://docs.influxdata.com/influxdb/v1.7/> [besucht am 2020-02-23].
- [10] the V8 project authors, *Documentation · V8*, 2014. [Online]. Adresse: <https://v8.dev/docs> [besucht am 2020-02-23].
- [11] OpenJS Foundation. (2020). Node.js, [Online]. Adresse: <https://nodejs.org/> [besucht am 2020-02-23].
- [12] npm, Inc., *npm Documentation*, 2020. [Online]. Adresse: <https://docs.npmjs.com/> [besucht am 2020-02-23].
- [13] ——, (2020). npm public registry, [Online]. Adresse: <https://registry.npmjs.org/> [besucht am 2020-03-01].
- [14] OpenJS Foundation, *Node.js v12.16.1 Documentation*, Version 12.16.1, 2020. [Online]. Adresse: <https://nodejs.org/docs/latest-v12.x/api/documentation.html> [besucht am 2020-02-23].
- [15] ECMA, „ECMAScript® 2019 Language Specification,“ Ecma International, Geneva, CH, Standard, 2019-06. [Online]. Adresse: <http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM>.
- [16] F. Aboukhadijeh, *simple-get*, 2018. [Online]. Adresse: <https://github.com/feross/simple-get/blob/master/README.md> [besucht am 2020-02-23].
- [17] Grafana Labs. (2020). Clock Panel Plugin for Grafana, [Online]. Adresse: <https://github.com/grafana/clock-panel/blob/master/README.md> [besucht am 2020-03-30].
- [18] R. McKinley und A. Velikiy. (2020). Discrete Panel, [Online]. Adresse: <https://github.com/NatelEnergy/grafana-discrete-panel/blob/master/README.md> [besucht am 2020-03-30].
- [19] algenty, *Flowcharting for Grafana*, 2020. [Online]. Adresse: <https://algenty.github.io/flowcharting-repository> [besucht am 2020-03-30].
- [20] michaeldmoore. (2020). michaeldmoore-multistat-panel, [Online]. Adresse: <https://github.com/michaeldmoore/michaeldmoore-multistat-panel/blob/master/README.md> [besucht am 2020-03-30].

- [21] Y. Rekhter, B. Moskowitz, D. Karrenberg und G. de Groot, *Address Allocation for Private Internets*, RFC 1597 (Informational), RFC, Obsoleted by RFC 1918, Fremont, CA, USA: RFC Editor, 1994-03. doi: 10.17487/RFC1597. [Online]. Adresse: <https://www.rfc-editor.org/rfc/rfc1597.txt> [besucht am 2020-02-23].
- [22] D. Community. (2019). Release Notes for Debian 10 (buster), [Online]. Adresse: <https://www.debian.org/releases/stable/i386/release-notes/ch-upgrading.en.html> [besucht am 2020-03-30].
- [23] Grafana Labs. (2020). Install on Debian/Ubuntu | Grafana Labs, [Online]. Adresse: <https://grafana.com/docs/grafana/latest/installation/debian/> [besucht am 2020-02-19].
- [24] T. S. Hatch, *salt-key(1) Salt*, 2019.
- [25] I. SaltStak, *Salt Documentation*, 2020, 5 ff. [Online]. Adresse: <https://docs.saltstack.com/en/pdf/Salt-3000.pdf> [besucht am 2020-03-24].
- [26] W. Odom, *CCNA Routing and Switching*. 2016.
- [27] V. maintainers. (2020), [Online]. Adresse: <https://vyos.readthedocs.io/en/latest/index.html> [besucht am 2020-03-30].
- [28] Netcraft Ltd. (2020). January 2020 Web Server Survey, [Online]. Adresse: <https://news.netcraft.com/archives/2020/01/21/january-2020-web-server-survey.html> [besucht am 2020-02-18].
- [29] E. Nemeth, G. Snyder und T. R. Hein, *Unix and Linux system administration handbook*. 2018, S. 1091.
- [30] ——, *Unix and Linux system administration handbook*. 2018, S. 1111.
- [31] ——, *Unix and Linux system administration handbook*. 2018, S. 1120.
- [32] S. Sanfilippo, *hping3(8) System Manager's Manual*, 2001.
- [33] J. Dugan, *iperf(1) User Manuals*, 2008.
- [34] N. Marriott, *tmux(1) BSD General Commands Manual*, 2020.
- [35] P. Falstad, *zsh(1) General Commands Manual*, 2017.
- [36] V. Jacobson, *tcpdump(8) System Manager's Manual*, 2017.
- [37] E. Nemeth, G. Snyder und T. R. Hein, *Unix and Linux system administration handbook*. 2018, S. 1331.

# Abbildungsverzeichnis

2.1	Projektteam . . . . .	16
	S. Gruber	
2.2	Zeitplan von Marcus Edlinger . . . . .	17
	M. Edlinger	
2.3	Zeitplan von Rafael Haigermoser . . . . .	18
	R. Haigermoser	
2.4	Zeitplan von Anna Victoria Krupa . . . . .	19
	A. V. Krupa	
5.1	Die Ordnerstruktur des NetBat Servers . . . . .	32
	M. Edlinger	
6.1	Topologie in Grafana GUI . . . . .	94
	A. V. Krupa	
6.2	Query Darstellung in Grafana GUI . . . . .	94
	A. V. Krupa	
6.3	NetBat-Logo in Grafana GUI . . . . .	96
	A. V. Krupa	
7.1	Warnungsbedingung in Grafana GUI . . . . .	105
	A. V. Krupa	
7.2	RTT mit Alarmierung in Grafana GUI . . . . .	107
	A. V. Krupa	
7.3	Query-Konfiguration zur Alarmierungsvorbereitung für RTT in Grafana GUI . . . . .	107
	A. V. Krupa	
7.4	Visualisierungseinstellungen für RTT in Grafana GUI . . . . .	108
	A. V. Krupa	
7.5	Alarmregeln für RTT in Grafana GUI . . . . .	108
	A. V. Krupa	
7.6	Alarmbedingungen für RTT in Grafana GUI . . . . .	109
	A. V. Krupa	

7.7	„NoData“ und „Error Handling“ Optionen für RTT in Grafana GUI . . . . .	109
	A. V. Krupa	
7.8	Benachrichtigung für RTT in Grafana GUI . . . . .	110
	A. V. Krupa	
7.9	Festplattenauslastung mit Alarmierung in Grafana GUI . . . . .	112
	A. V. Krupa	
7.10	Alarmbedingungen für Festplattenauslastung in Grafana GUI . . . . .	112
	A. V. Krupa	
8.1	Topologie des Testnetzwerks . . . . .	118
	A. V. Krupa und R. Haigermoser	
8.2	Das Menü des „Firewall Admin“ Tools . . . . .	119
	R. Haigermoser	
8.3	Die Portweiterleitungseinstellungen des „Firewall Admin“ Tools . . . . .	120
	R. Haigermoser	
8.4	Belastungen bei einem Netzwerktest mit hping3 . . . . .	129
	R. Haigermoser	
8.5	Belastungen bei einem Netzwerktest mit iperf . . . . .	130
	R. Haigermoser	
8.6	Die grafische Benutzeroberfläche von TRex . . . . .	134
	R. Haigermoser	
8.7	Liste aller VMs in der GUI von NetBox. . . . .	135
	R. Haigermoser	
8.8	VM-Detailansicht in der GUI von NetBox. . . . .	136
	R. Haigermoser	

# Tabellenverzeichnis

3.1	NetBat Logo Datensätze	23
5.1	Ergebnisse der Geschwindigkeitsmessung für das Zusammenhängen von Arrays	55

# Abkürzungsverzeichnis

**API** Application Programming Interface.

**BDI** Bridge Domain Interfaces.

**CPU** Central Processing Unit.

**CQ** Continuous Query.

**DNS** Domain Name System.

**DOS** Denial of Service.

**GUI** Graphical User Interface.

**HTTP** Hypertext Transfer Protocol.

**HTTPS** Hypertext Transfer Protocol Secure.

**ICMP** Internet Control Message Protocol.

**IETF** Internet Engineering Task Force.

**IP** Internet Protocol.

**IPv4** Internet Protocol Version 4.

**IPv6** Internet Protocol Version 6.

**ISO** International Organization for Standardization.

**JIT-Compiler** Just-in-time-Compiler.

**JSON** JavaScript Object Notation.

**LVM** Logical Volume Manager.

**MAC** Mandatory Access Control.

**NAT** Network Address Translation.

**NTP** Network Time Protocol.

**OID** Object Identifier.

**OSI** Open Systems Interconnection model.

**RAM** Random-Access Memory.

**RFC** Request for Comments.

**RP** Retention Policy.

**RTT** Round Trip Time.

**SNMP** Simple Network Management Protocol.

**SNMPv2c** Community-Based Simple Network Management Protocol Version 2.

**SQL** Structured Query Language.

**SSH** Secure Shell.

**SVG** Scalable Vector Graphics.

**TCP** Transmission Control Protocol.

**TLS** Transport Layer Security.

**TTL** Time to live.

**UDP** User Datagram Protocol.

**URL** Uniform Resource Locator.

**UTC** Coordinated Universal Time.

**VLAN** Virtual Local Area Network.

**VM** virtuelle Maschine.

**VPN** Virtual Private Network.

**YAML** YAML Ain't Markup Language.

## Begleitprotokoll

**Name:** Hr. Marcus Edlinger

**Diplomarbeitstitel:** NetBat

KW	Beschreibung	Zeitaufwand
38	Teamfindung, Diplomarbeitsantrag erstellen	10h
39	Diplomarbeitsantrag erstellen & einreichen	10h
40	Projektplanung, Aufgabenverfeinerung, Projektplan	10h
41	Git Repository Initialisieren	10h
42	Experimentieren mit dem SNMP Interface	10h
43	Experimentieren mit dem Routertemplates und ICMP Interface	10h
44	Arbeit am Programmkernel, Wiederverwertung der Experimente	10h
45	Arbeit am Programmkernel: Datenbankabstraktion, InfluxDB Implementierung	10h
46	Arbeit am Programmkernel: Laden der Konfiguration, Munin Modul	10h
47	Arbeit am Programmkernel: Laden der Konfiguration	10h
48	Arbeit am Programmkernel, Downsampling, NetBat Logo Modul	10h
49	Logo-Design	10h
50	Vorbereitung der Techniker Präsentation	10h
51	Arbeit am Programmkernel: Laden der Konfiguration	10h
52	---	0h
1	---	0h
2	Arbeit am Programmkernel	10h
3	Arbeit am Programmkernel, HTTP Modul, Erweiterung des Munin Templates	10h
4	Design von Folder und Plakat	10h
5	Arbeit am Programmkernel, „stdout“ Datenbankimplementierung	10h
6	Arbeit am Programmkernel	10h
7	---	0h
8	Arbeit am Programmkernel: automatische Grafana Konfiguration	10h
9	Diplomarbeit schreiben	10h
10	Diplomarbeit schreiben, Arbeit am Programmkernel	10h
11	Diplomarbeit schreiben	10h
12	Diplomarbeit schreiben	10h
13	Diplomarbeit schreiben, cmd Modul	10h
14	Diplomarbeit schreiben	10h
<b>Gesamt</b>		260h

KW ...Kalenderwoche

## Begleitprotokoll

**Name:** Hr. Rafael Haigermoser

**Diplomarbeitstitel:** NetBat

KW	Beschreibung	Zeitaufwand
38	Teamfindung, Diplomarbeitsantrag erstellen	10h
39	Diplomarbeitsantrag erstellen & einreichen	10h
40	Projektplanung, Aufgabenverfeinerung, Projektplan	10h
41	Setup Barracuda FW + VPN	10h
42	Setup VMs (Grafana, Cisco Router)	10h
43	Grafana Addons, Dokumentation lesen (scappy, iperf)	10h
44	Dokumentation lesen iperf hping, Serververwaltung	10h
45	Problemsuche: „Ping unavailable“ bei Grafana VM	10h
46	Setup TRex	10h
47	Grafana VM „Ping Problem“ gelöst	10h
48	Setup VyOs	10h
49	Setup NetBox	10h
50	Vorbereitung der Techniker-Präsentation, Flyer	10h
51	Setup Wireguard	10h
52	---	0h
1	---	0h
2	Setup Salt	10h
3	HAproxy	10h
4	Design von Folder und Plakat	10h
5	Backup	10h
6	TRex, iperf hping3 nochmals testen, Diplomarbeit schreiben	10h
7	---	0h
8	Diplomarbeit schreiben	10h
9	Diplomarbeit schreiben	10h
10	Diplomarbeit schreiben	10h
11	Diplomarbeit schreiben	10h
12	Diplomarbeit schreiben	10h
13	Diplomarbeit schreiben	10h
14	Diplomarbeit schreiben	10h
<b>Gesamt</b>		260h

**KW** ...Kalenderwoche

## Begleitprotokoll

Name: Fr. Anna Victoria Krupa

Diplomarbeitstitel: NetBat

KW	Beschreibung	Zeitaufwand
38	Teamfindung, Diplomarbeitsantrag erstellen	10h
39	Diplomarbeitsantrag erstellen & einreichen	10h
40	Projektplanung, Aufgabenverfeinerung, Projektplan	10h
41	Recherchieren & Experimentieren mit Graphen	10h
42	Recherchieren & Experimentieren mit Alarmierung	10h
43	Wiederverwertung der Experimente	10h
44	Recherchieren & Experimentieren mit FlowCharting-Panel	10h
45	Erstellung der Topologie	10h
46	Wiederverwertung der Experimente	10h
47	Konfiguration der ICMP Interfaces	10h
48	Ergänzen der Topologie, Logo-Design	10h
49	Logo-Design	10h
50	Vorbereitung der Techniker Präsentation	10h
51	Vorbereitung der Techniker Präsentation	10h
52	---	0h
1	---	0h
2	Darstellung der Munin Daten	10h
3	Darstellung der Munin Daten	10h
4	Design von Folder und Plakat	10h
5	Recherche & Konfiguration des Multistat-Panels	10h
6	Konfiguration des Multistat-Panels	10h
7	---	0h
8	Diplomarbeit schreiben	10h
9	Diplomarbeit schreiben	10h
10	Device-Templates	10h
11	Diplomarbeit schreiben	10h
12	Diplomarbeit schreiben	10h
13	Diplomarbeit schreiben	10h
14	Diplomarbeit schreiben	10h
<b>Gesamt</b>		260h

KW ...Kalenderwoche