



# Evaluierung eines Messpunkte-Clusters für Netzwerktests auf dem Campus der TU Clausthal

Christian Rebischke <christian.rebischke@tu-clausthal.de>

21.02.2019



## Agenda

- Motivation und Problemstellung
- Ziel
- Umsetzung
  - Speichern
  - Sammeln
  - Auswerten
  - Skalieren
- Fazit und Ausblick

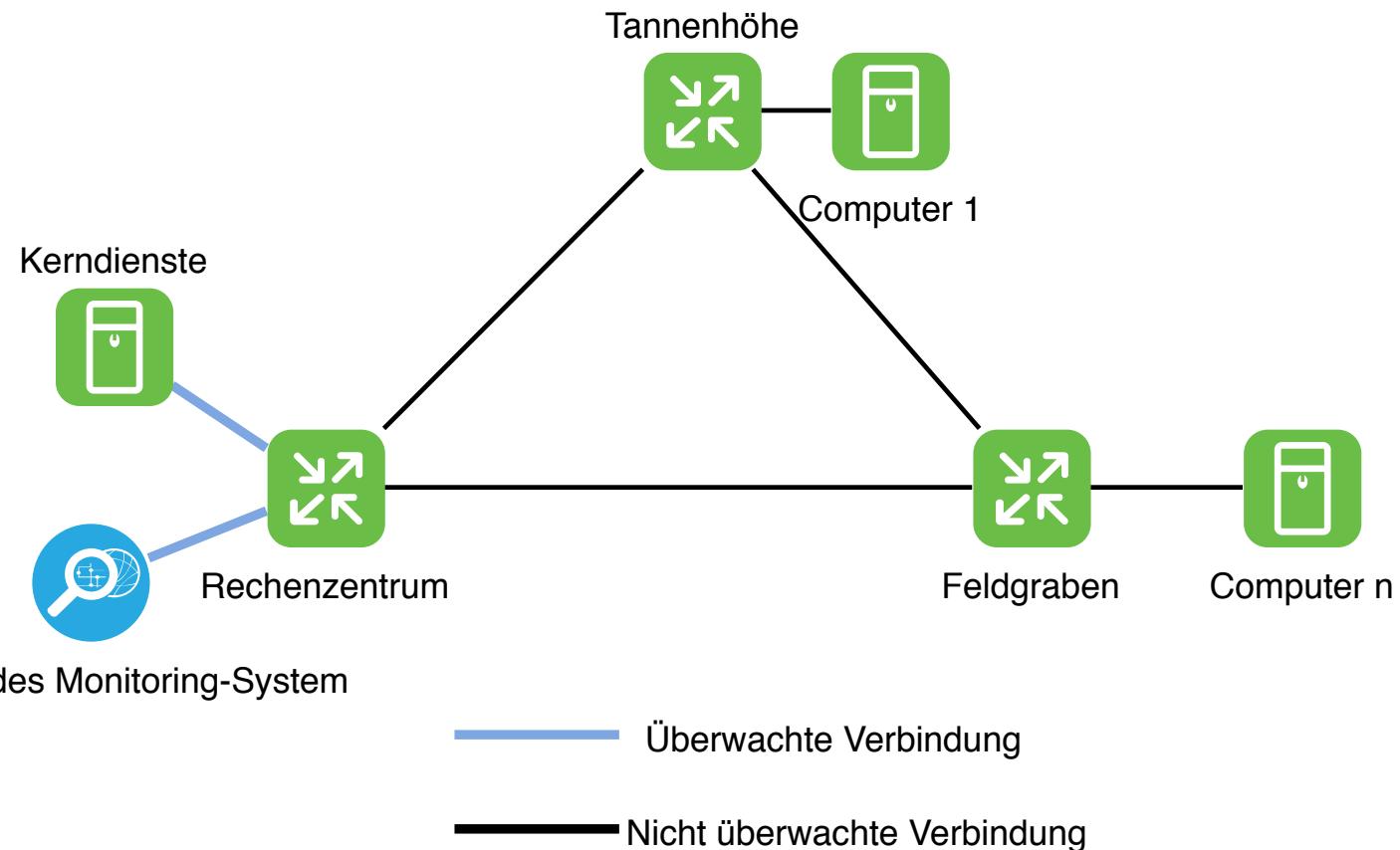


## Motivation und Problemstellung

- Rechnernetze werden immer größer und komplexer
- Anzahl der Web-Dienste steigt stetig an
- Alleine die TU Clausthal hat über 65.536 IPv4 Adressen
- Kerndienste, wie in etwa DNS, müssen von überall aus in gleicher Qualität erreichbar bleiben.

→ Überwachung von Rechnernetzen wird schwieriger zu skalieren

# Motivation und Problemstellung





## Motivation und Problemstellung

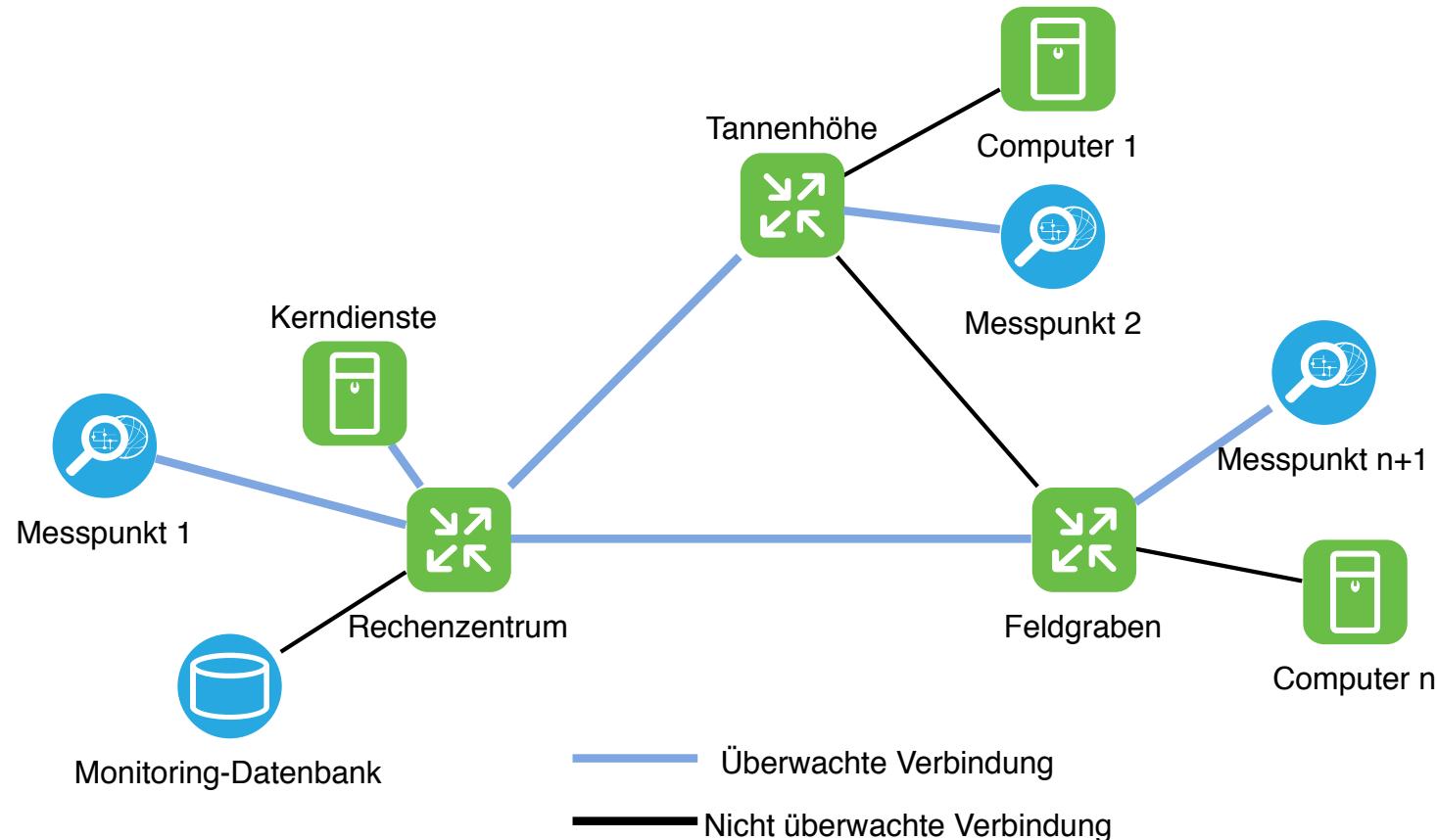
*Wie lassen sich Rechnernetze dezentral und horizontal skalierbar überwachen?*



## Ziel

*Erstellung eines Prototypen für das Rechenzentrum  
der TU Clausthal um die Machbarkeit einer solchen  
Lösung zu testen und genauer zu studieren*

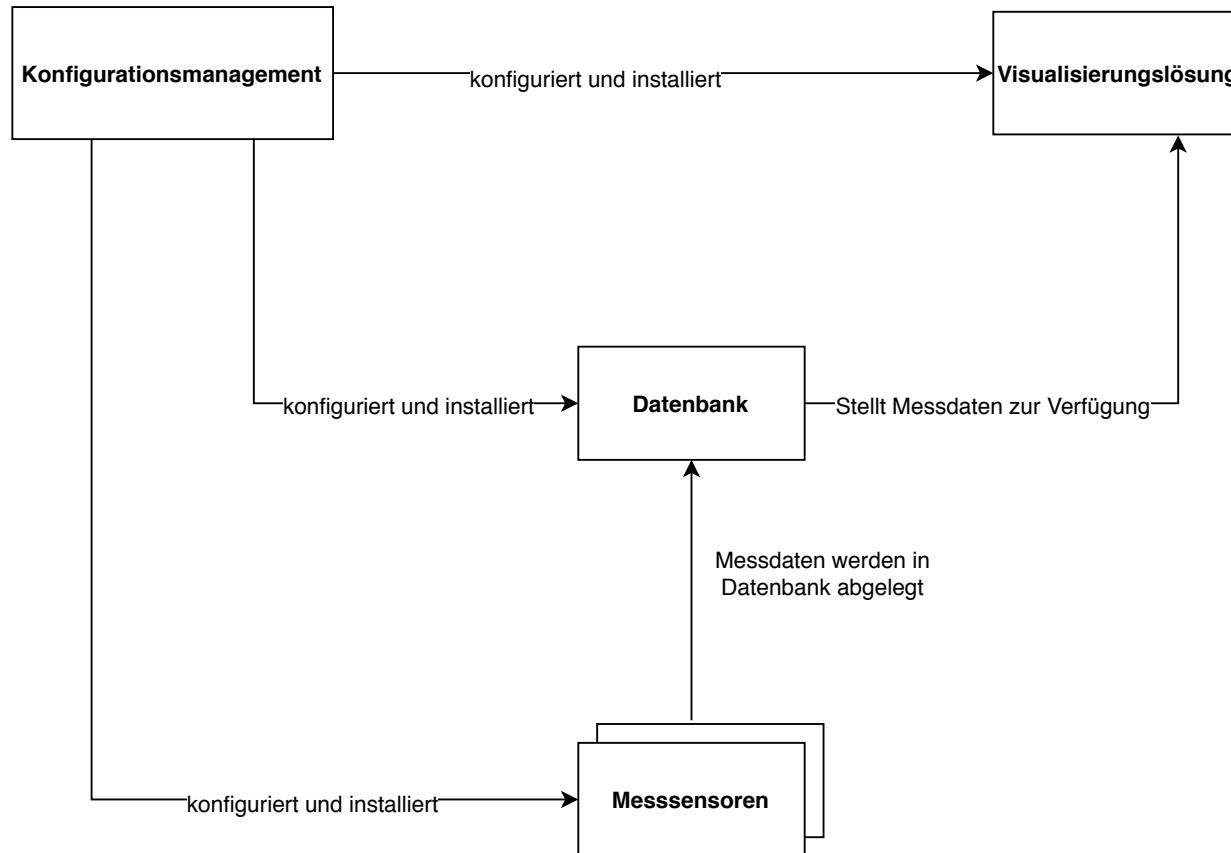
# Ziel



## Ziel

- Kerndienste müssen dezentral überwacht werden können: HTTP, HTTPS, DNS, CIFS
- Das überwachende System muss horizontal skalierbar sein
- Das überwachende System muss kontrollierbar bei horizontaler Skalierung sein
- Das überwachende System muss bestehende Infrastruktur nutzen (TCP/IP- oder UDP/IP-basierte Rechnernetze) und mit (logischer) Netzsegmentierung klar kommen (VLAN, VXLAN, etc)
- Gewonnene Daten müssen speicherbar und visualisierbar sein

# Umsetzung



## Speichern

- Daten sollen im sekundentakt gewonnen werden.
- Datenbank muss viele Daten im sekundentakt entgegennehmen (**INSERT** Operation) und speichern müssen.
- Datenbank muss skalierbar sein. (Was mit 10 Datensätzen pro Sekunde funktioniert muss auch für 1.000.000 Datensätze pro Sekunde funktionieren)

→ **Time Series Datenbanken (TSDBs)**

## Speichern

- Was sind **TSDBs**?

- Zeit als Index
- Optimiert auf viele **Insert**- und **Read**-Operationen für Reihen
- Optimiert auf sehr hohe Datenmengen
- Optimiert auf horizontale Skalierbarkeit

→ **TSDBs sind bestens geeignet für Monitoring.**  
**Bekannter Vertreter: Prometheus**

## Prometheus

- Monitoringanwendung mit eingebauter **TSDB**
- Entwickelt in der Programmiersprache **Golang** bei der Firma **Soundcloud** im Jahr 2012.
- Projekt der Cloud-Native-Computing-Foundation (**CNCF**).
- Weitverbreitet in horizontalskalierenden Cloud- und Container-Umgebungen.
- Open Source.



## Cloud Native Computing Foundation (CNCF)

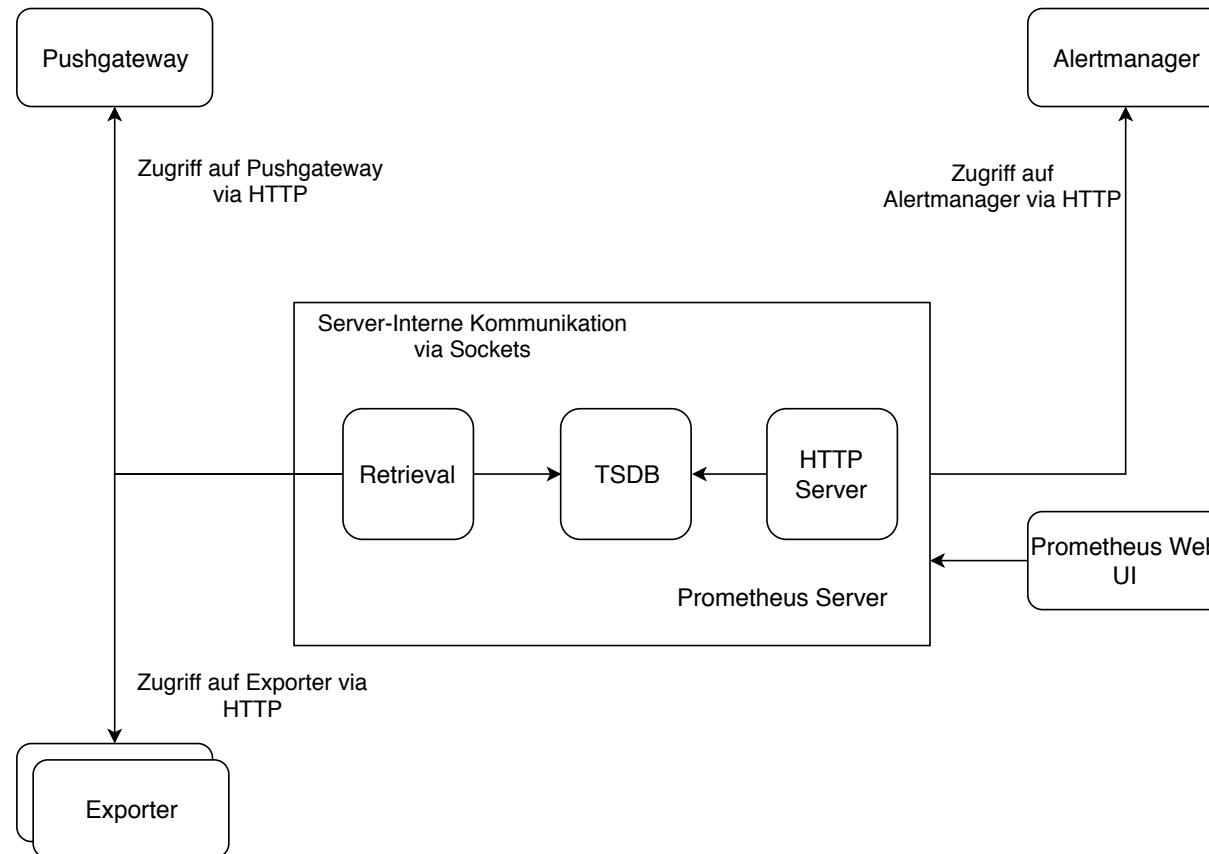
- Ableger der **Linux Foundation (LF)**.
- Nicht-Kommerzielle Vereinigung zur Förderung von Open Source Technologien im Cloud- und Container-Bereich.
- Bekannte Mitglieder: **Google, Amazon AWS, Microsoft, IBM, Huawei, Docker, Cisco, Ebay, Facebook, Twitter** und mehr.



## Prometheus Datenformat

- Datenbankeinträge werden als Zweier-Tupel aus einem millisekunden genauen Timestamp und Float64 Daten gebildet.
- Auf diese Daten wird über folgendes Format zugegriffen:  
 $\langle metric\ name \rangle \{ \langle label\ name \rangle = \langle label\ value \rangle, \dots \}$
- Beispiel:  
 $http\_requests\_total \{ server = "www.tu-clausthal.de" \}$

# Prometheus Komponenten



## Sammeln

- Einplatinenrechner als Messpunkte
- Natives 1 Gigabit Ethernet
- Vollwertiges Linux-Betriebssystem
- Durch Zuweisen einer festen IP-Adresse mobil einsetzbar
- Bei Bedarf erweiterbar
- Günstig im Preis (50€ das Stück)



Odroid XU4

## Sammeln von Daten über Kerndienste

- Einsatz von Prometheus Exportern auf Einplatinenrechner
- *Node-Exporter*: Sammeln von Daten über den Einplatinenrechner (CPU, RAM, Netzwerk Traffic, etc)
- *Blackbox-Exporter*: Überprüft Verfügbarkeit von Kerndiensten über den Einplatinenrechner hin zum eigentlichen Dienst (DNS, HTTP, HTTPS)

## Sammeln von Daten über Kerndienste

- Einsatz von Prometheus Exportern auf Einplatinenrechner
- *Node-Exporter*: Sammeln von Daten über den Einplatinenrechner (CPU, RAM, Netzwerk Traffic, etc)
- *Blackbox-Exporter*: Überprüft Verfügbarkeit von Kerndiensten über den Einplatinenrechner hin zum eigentlichen Dienst (DNS, HTTP, HTTPS)

→ Problem: CIFS Unterstützung fehlt

# Idee: CIFS Unterstützung im Node-Exporter

- Zugriff auf CIFS-Statistiken via Linux ProcFS:  
[`/proc/fs/cifs/Stats`](#)
- Parsen der CIFS-Statistiken via ProcFS  
Node-Exporter Komponente in Golang

```
Resources in use
CIFS Session: 1
Share (unique mount targets): 2
SMB Request/Response Buffer: 1 Pool size: 5
SMB Small Req/Resp Buffer: 1 Pool size: 30
Operations (MIDs): 0

0 session 0 share reconnects
Total vfs operations: 16 maximum at one time: 2

1) \\server1\share1
SMBs: 9 Oplocks breaks: 0
Reads: 0 Bytes: 0
Writes: 0 Bytes: 0
Flushes: 0
Locks: 0 HardLinks: 0 Symlinks: 0
Opens: 0 Closes: 0 Deletes: 0
Posix Opens: 0 Posix Mkdirs: 0
Mkdirs: 0 Rmdirs: 0
Renames: 0 T2 Renames 0
FindFirst: 1 FNext 0 FClose 0

2) \\server2\share2
SMBs: 20
Negotiates: 0 sent 0 failed
SessionSetups: 0 sent 0 failed
Logoffs: 0 sent 0 failed
TreeConnects: 0 sent 0 failed
TreeDisconnects: 0 sent 0 failed
Creates: 0 sent 2 failed
Closes: 0 sent 0 failed
Flushes: 0 sent 0 failed
Reads: 0 sent 0 failed
Writes: 0 sent 0 failed
Locks: 0 sent 0 failed
IOCTLs: 0 sent 0 failed
Cancels: 0 sent 0 failed
Echos: 0 sent 0 failed
QueryDirectories: 0 sent 0 failed
ChangeNotifies: 0 sent 0 failed
QueryInfos: 0 sent 0 failed
SetInfos: 0 sent 0 failed
OplockBreaks: 0 sent 0 failed
```

**Header**

**SMB v1  
Statistiken**

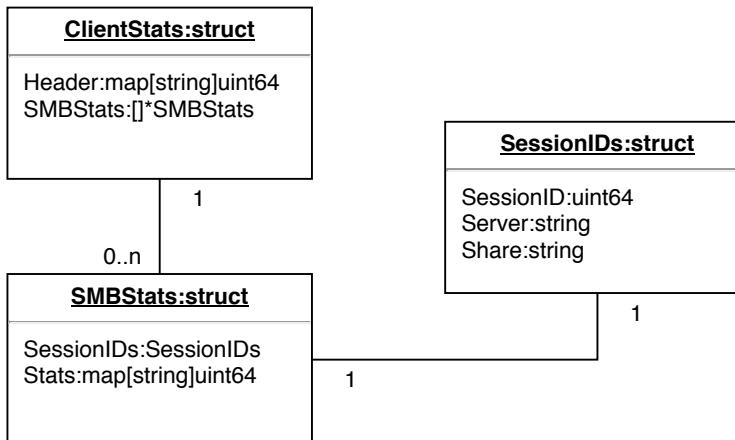
**SMB v2 / v3  
Statistiken**

## Golang

- Entwickelt bei **Google**.
- Sicherer (Memory Management) und portierbarer (Statische Binärdateien) als **C**.
- Weitverbreitet im Container- und Cloud-Bereich: **Docker**, **Kubernetes**, **Etcd**, **Fleet**, **gRPC**, **Grafana**, diverse **Openstack** und **Amazon AWS Bindings**.
- Unterstützt durch die **CNCF**.
- Open Source.



# Datenmodell für die CIFS-Statistiken



```
1 // model for the SMB statistics
2 type SMBStats struct {
3     SessionIDs SessionIDs
4     Stats      map[string]uint64
5 }
6
7 // model for the Share sessionID "number) \\server\share"
8 type SessionIDs struct {
9     SessionID uint64
10    Server   string
11    Share    string
12 }
13
14 // model for the CIFS header statistics
15 type ClientStats struct {
16     Header      map[string]uint64
17     SMBStatsList []*SMBStats
18 }
```

# Parsen von CIFS-Statistiken via RegEx

```
1 // Array with fixed regex for parsing the SMB stats header
2 var regexpHeaders = [...]*regexp.MustCompile{
3     regexp.MustCompile('CIFS Session: (?P<sessions>\d+)'),
4     regexp.MustCompile('Share \unique mount targets\): (?P<shares>\d+)'),
5     regexp.MustCompile('SMB Request/Response Buffer: (?P<smbBuffer>\d+) Pool size: (?P<smbPoolSize>\d+)'),
6     regexp.MustCompile('SMB Small Req/Resp Buffer: (?P<smbSmallBuffer>\d+) Pool size: (?P<smbSmallPoolSize>\d+)'),
7     regexp.MustCompile('Operations \MIDs\): (?P<operations>\d+)'),
8     regexp.MustCompile('(?P<sessionCount>\d+) session (?P<shareReconnects>\d+) share reconnects'),
9     regexp.MustCompile('Total vfs operations: (?P<totalOperations>\d+) maximum at one time: (?P<totalMaxOperations>\d+)'),
10 }
```

```
Resources in use
CIFS Session: 2
Share \unique mount targets\): 4
SMB Request/Response Buffer: 2 Pool size: 6
SMB Small Req/Resp Buffer: 2 Pool size: 30
Operations \MIDs\): 0

0 session 0 share reconnects
Total vfs operations: 90 maximum at one time: 2
```

## Parsen des Headers

```
11 // parseHeader parses our SMB header
12 func parseHeader(line string, header map[string]uint64) error {
13     for _, regexpHeader := range regexpHeaders {
14         match := regexpHeader.FindStringSubmatch(line)
15         if match == nil {
16             continue
17         }
18         for index, name := range regexpHeader.SubexpNames() {
19             if index == 0 || name == "" {
20                 continue
21             }
22             value, err := strconv.ParseUint(match[index], 10, 64)
23             if nil != err {
24                 return fmt.Errorf("invalid value in header")
25             }
26             header[name] = value
27         }
28     }
29     return nil
30 }
```

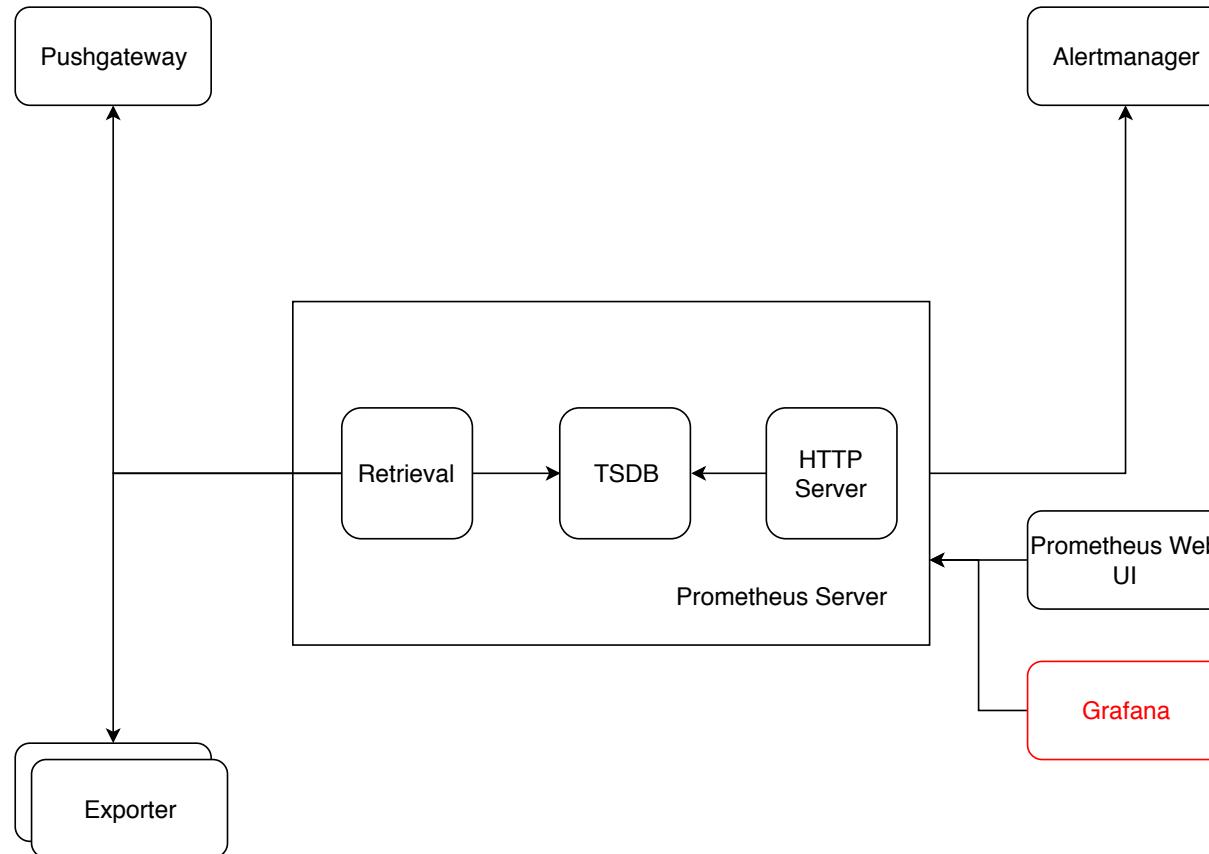
1

## Auswerten: Grafana

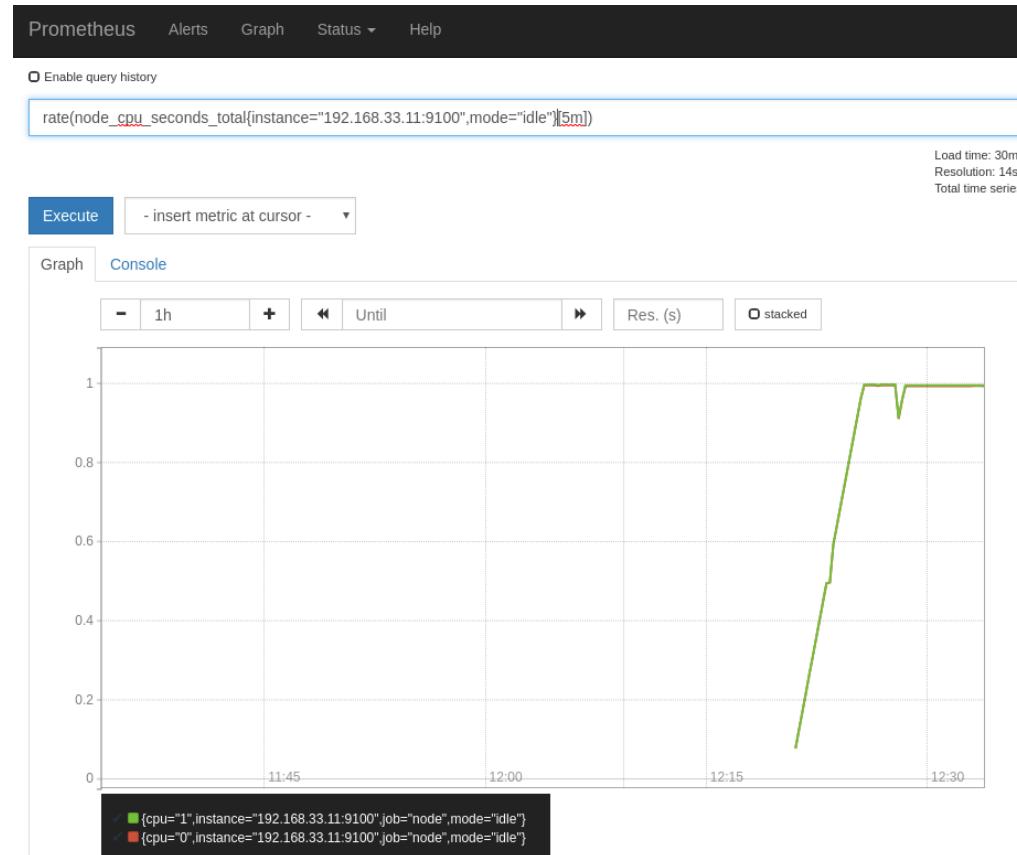
- Plattform zur Visualisierung von Daten.
- Entwickelt in der Programmiersprache **Golang**.
- Einfache Provisionierung und Deployment
- Wird benutzt bei: **Paypal, Ebay, CERN, Booking.com** und mehr.
- Voller **Prometheus** Support.
- Open Source.



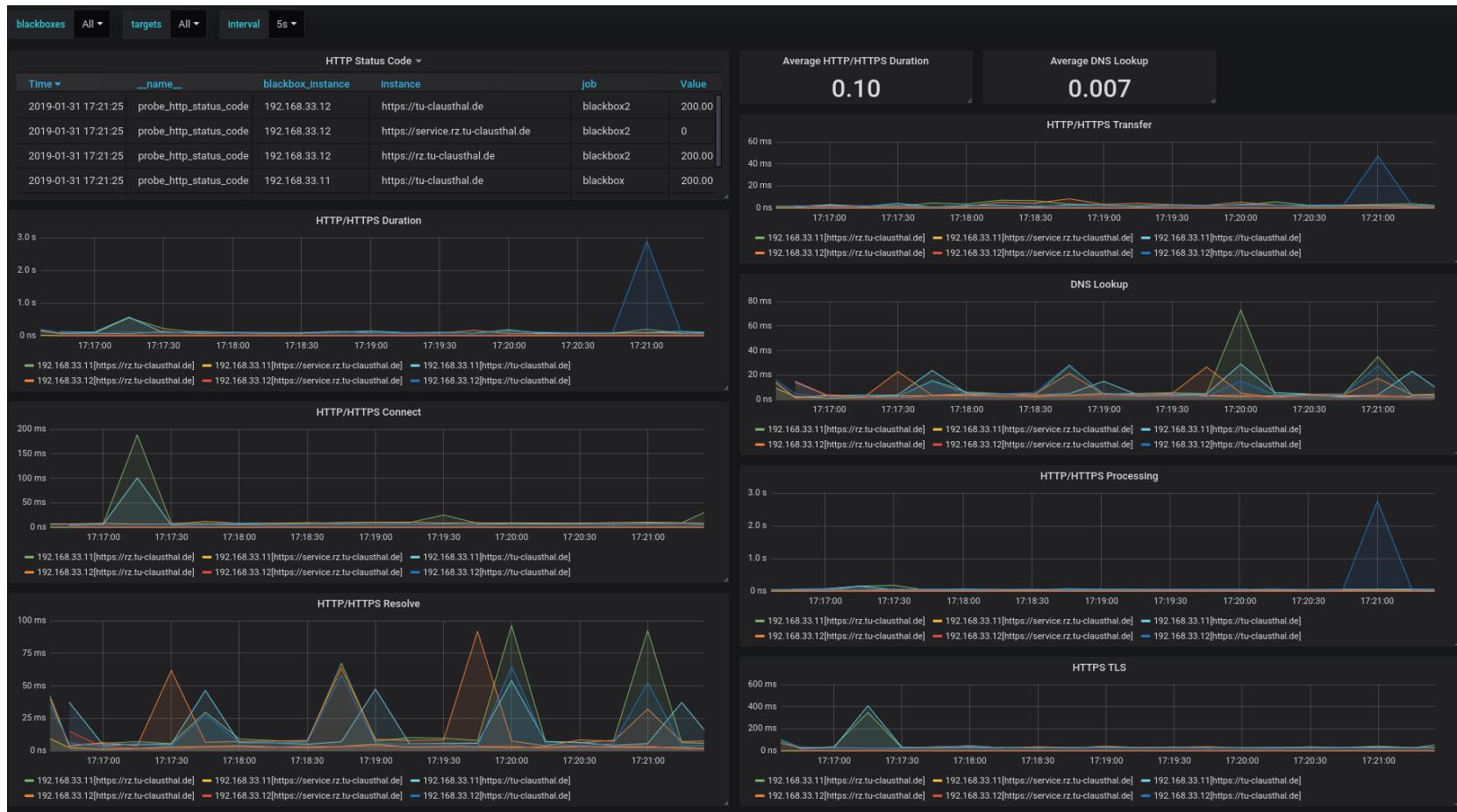
# Grafana und Prometheus



# Prometheus Web UI



# Grafana Web UI



## Skalieren: Ansible

- Werkzeug für Konfigurationsmanagement.
- Entwickelt in der Programmiersprache **Python**.
- Generiert aus **YAML** (Markup-Sprache) via **Jinja2** (Template-Sprache) vollwertige Konfigurationsdateien für Software.
- Arbeitet **Agent-less** via **OpenSSH**.
- Besitzt Module für diverse Aufgaben, inklusive Cloud und Container.
- Open Source.



# Ansible Ordnerstruktur

```
.  
└── ansible.cfg      // Ansible Konfigurationsdatei  
└── group_vars       // Einstellungen für Gruppen  
└── hosts            // Relation zwischen Hosts und Gruppen  
└── host_vars         // Individuelle Einstellungen für Hosts  
└── playbooks         // Relation zwischen Rollen und Gruppen/Hosts  
└── roles             // Verzeichnis für Rollen  
    └── files           // Statische Dateien  
    └── tasks            // Die eigentliche Programmlogik und abzuarbeitende Aufgaben  
    └── templates        // Dynamisch veränderbare Dateivorlagen
```

## Ansible Beispiel „hosts“ Datei

```
[prometheus]
192.168.33.10
```

```
[grafana]
192.168.33.10
```

```
[nodeexporter]
192.168.33.10
192.168.33.11
192.168.33.12
```

```
[blackboxexporter]
192.168.33.11
192.168.33.12
```

- Relation zwischen Gruppen und Hosts
- Dateiformat: INI
- Beliebig skalierbar

## Zusammenspiel von YAML und Jinja2

```
1 # Auszug aus prometheus/install.yml
2 - name: create prometheus data directory
3   file:
4     # {{ prometheus_db_dir }} ist Inline Jinja2
5     path: "{{ prometheus_db_dir }}"
6     state: directory
7     owner: prometheus
8     group: prometheus
9     mode: 0755
```

```
1 ---
2 prometheus_version: 2.6.0
3
4 prometheus_config_dir: /etc/prometheus
5 prometheus_db_dir: /var/lib/prometheus
6
7 prometheus_web_listen_address: "0.0.0.0:9090"
8 prometheus_web_external_url: ''
```

- Arbeitsanweisungen sind aus einem Mix aus YAML und Jinja2 gehalten
- Konfiguration und Flusssteuerung verläuft via YAML

## Fazit und Ausblick

- Eine dezentrale und skalierbare Überwachung ist möglich
- Die Programmiersprache Golang setzt sich immer mehr durch (besonders im Cloud und Container-Umfeld)
- Konfigurationsmanagement mit Tools wie Ansible, Chef, Saltstack, Puppet etablieren zunehmend
- Der Einsatz von Ansible ist auch in anderen Bereichen denkbar. Zum Beispiel: Switch- und Netzwerkkonfiguration

## Literatur

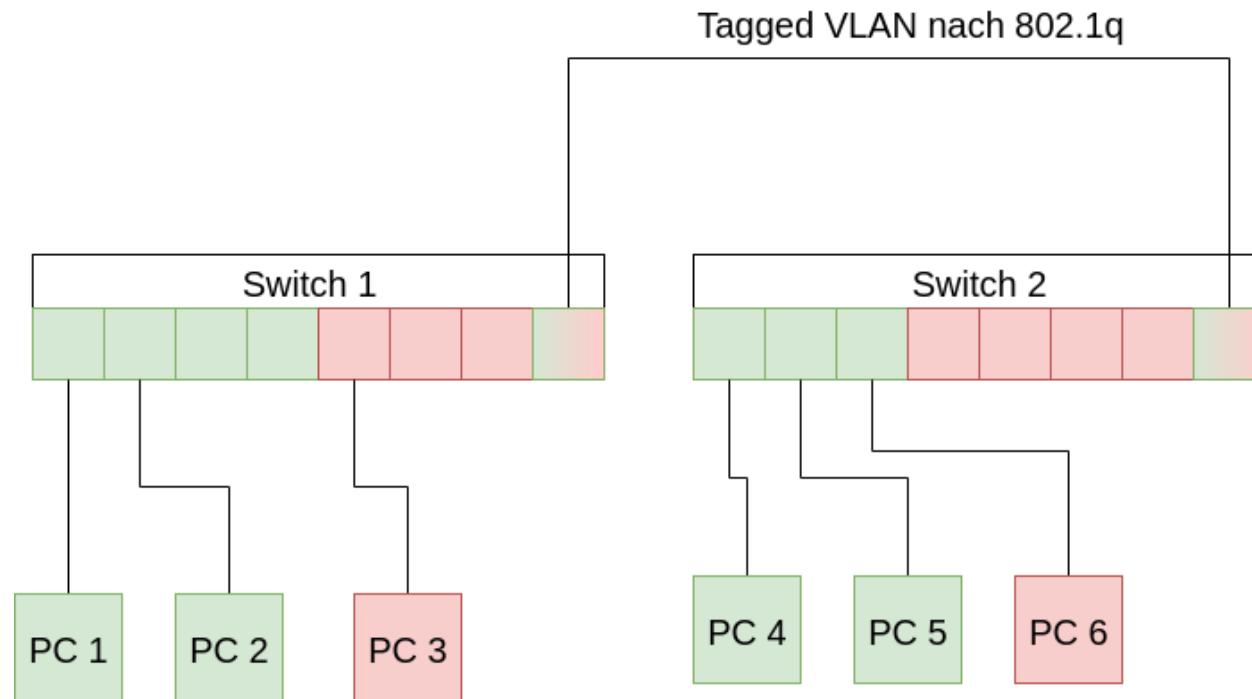
- <https://prometheus.io/docs/introduction/overview/>
- <https://grafana.com/docs/>
- <https://docs.ansible.com/>
- <https://www.cncf.io/>
- <https://golang.org/>
- Alle Grafiken sind erstellt via <https://draw.io/> oder Screenshots von existierendem Code



**Danke für die Aufmerksamkeit!**

## VLAN

- Standardisiert durch IEEE 802.1Q
- Bestandteil des IEEE 802.3 Ethernet Standards



## VXLAN

- Problem: Zu wenig IPv4 Adressen
- Lösung: Ethernet-Encapsulation in UDP

