IKU
SYSTEMS & SERVICES

# SLAC 2024

## MONITORING:
## PROMETHEUS / LOKI / GRAFANA

A. Niederländer

Differenzierte Metriken statt einfacher Checks

# Wozu Monitoring?

**01** Kein Selbstzweck, elementar wichtig

Ausfälle erkennen

Ausfälle vermeiden

Verfügbarkeit überprüfbar machen

**02** Allgemein

Vermeidung von (teuren) Schäden

**03** Genau

Service Level Agreements (SLA) von Services erfüllen

# Klassische Monitoring Produkte

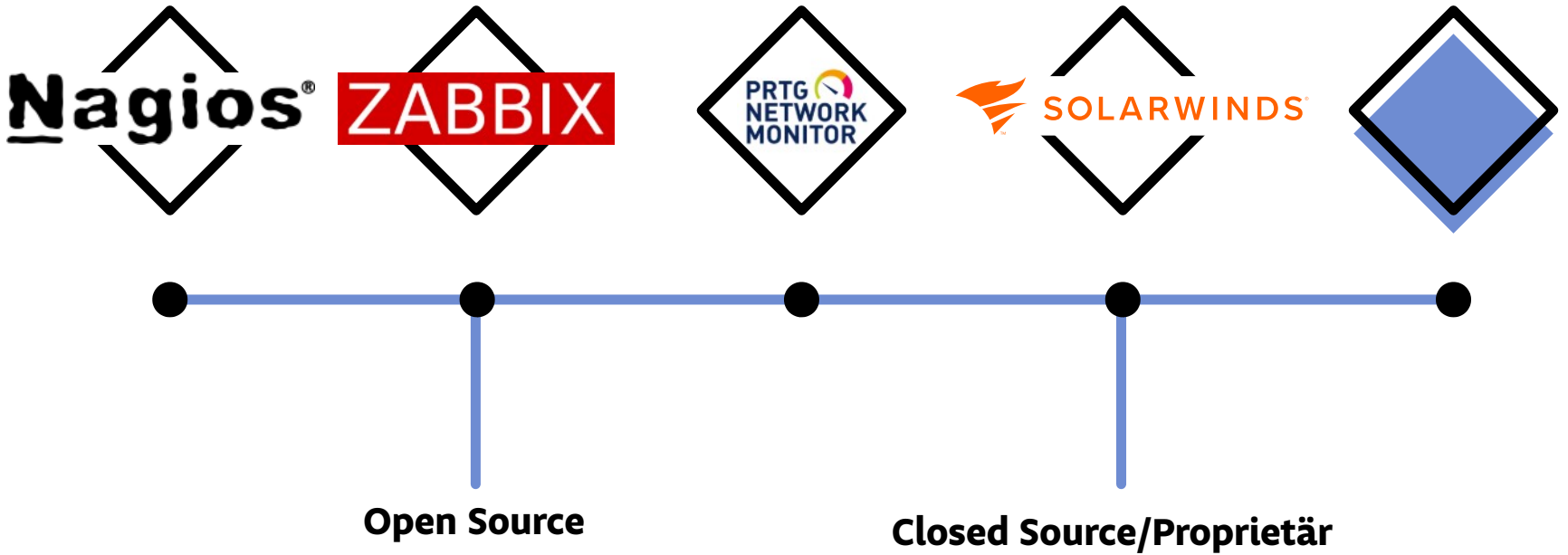
Nagios (Centreon,Icinga,Checkmk,...)

Zabbix

PRTG

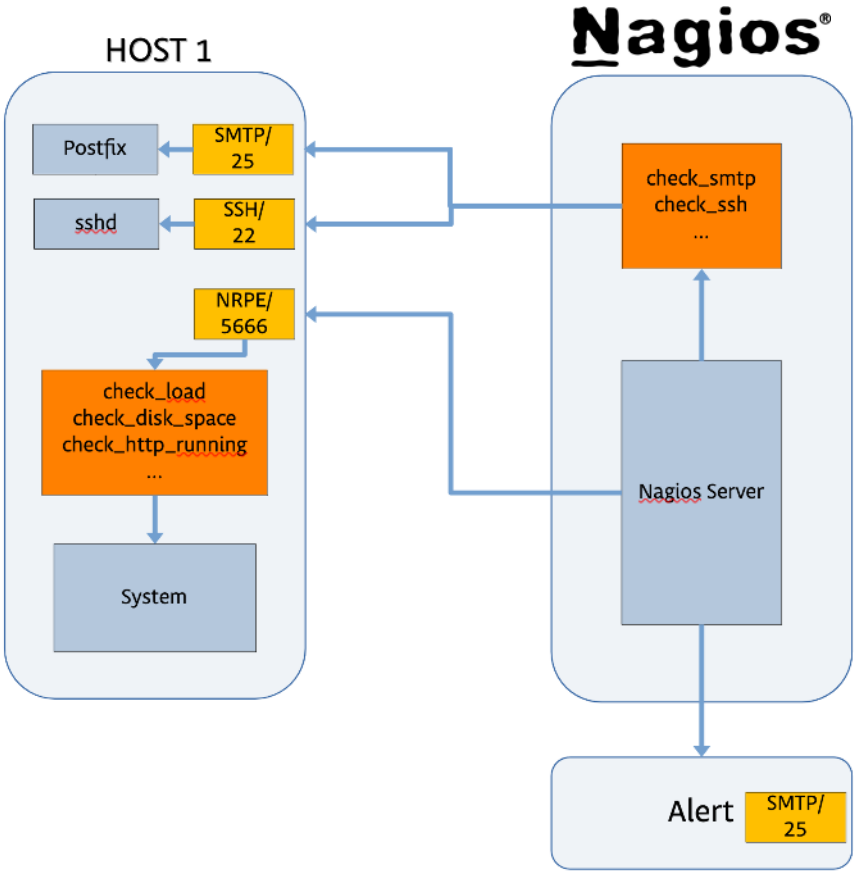
Solarwinds

Eigene Lösungen von OS / Infrastruktur-Herstellern



# Wie arbeitet Nagios?

- Agent: ***NRPE***
- Limits werden üblicherweise in NRPE gesetzt




- Server: ***Nagios***
- ‚pullt' Zustände
- „Passive" und „Active" checks
- Kennt nur 3 Zustände:
  - Rot
  - Gelb
  - Grün
- „undefined"
  → Ampel kaputt

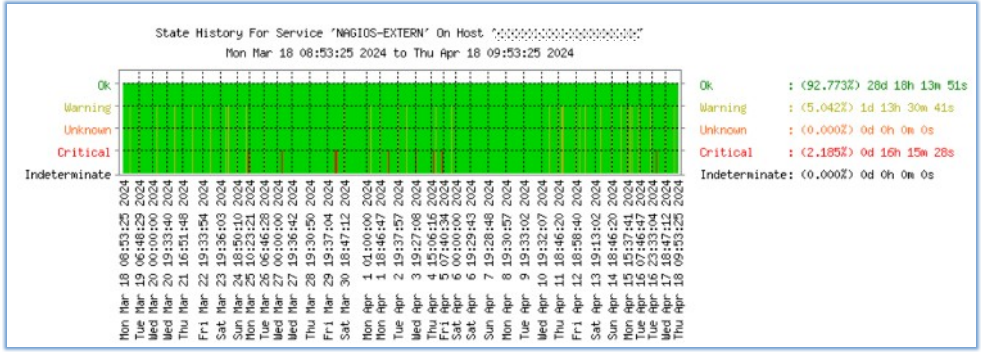
# Monitoring löst doch alles, oder?

GELÖST

- ✔ Monitoring funktioniert zuverlässig
- ✔ Es können sehr einfach auch komplexe Checks erstellt werden
- ✔ Alarmierung funktioniert
- ✔ Es können Statistiken bzgl. der Verfügbarkeit erstellt werden (rudimentär)

UNGELÖST

- ? Wie reagiert mein System im normalen Betrieb?
- ? Mehr als 3 Zustände
- ? Voraussagen über Last treffen können
- ? Mühelose Erstellung von Visualisierungen
- ? Relationen und Zusammenhänge besser erkennen
- ? ‚Deep Dive' (‚Traces')



# Was sind Metriken?

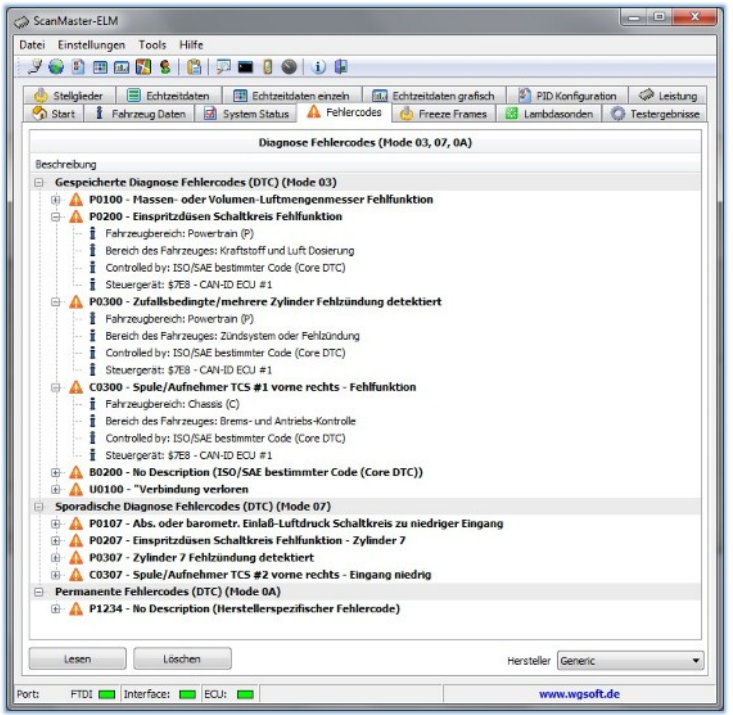
Nagios®

**Nagios: Zustände**


Vergleichbar mit Warnleuchte beim Auto Öl zu heiß

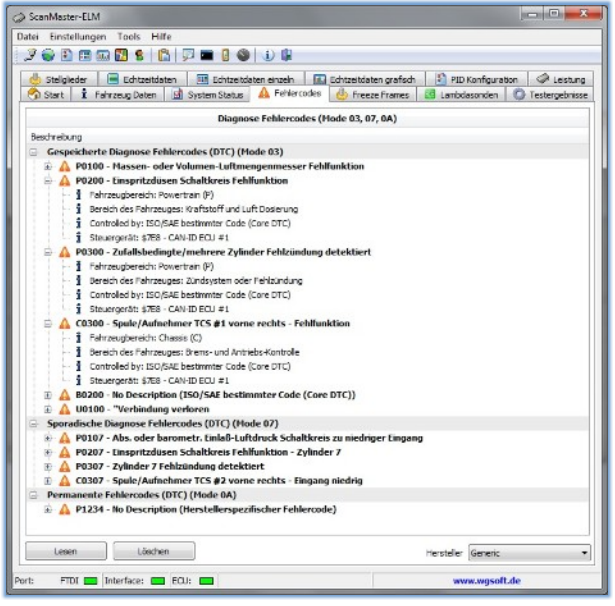
**Prometheus: Metriken**

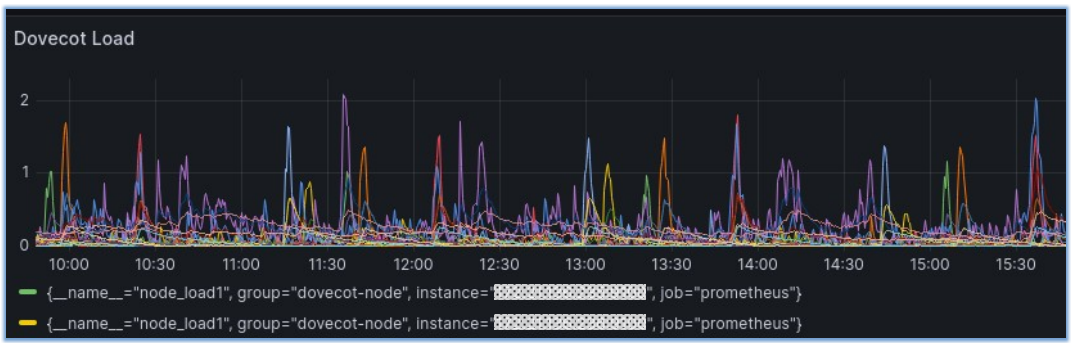
Vergleichbar mit Daten, die im Auto von den Steuergeräten verarbeitet werden.




# Unterschiede Metriken <-> Checks


| | | | | | | |
|---|---|---|---|---|---|---|
| LOAD | | OK | 04-04-2024 15:42:23 | 960d 1h 41m 57s | 1/3 | OK - load average: 0.09, 0.04, 0.01 |
| MEMORY | | OK | 04-04-2024 15:36:13 | 960d 1h 40m 41s | 1/3 | OK - 83.7% (13753204 kB) free. |
| PING | | OK | 04-04-2024 15:44:35 | 30d 17h 30m 21s | 1/3 | PING OK - Packet loss = 0%, RTA = 0.17 ms |
| Root Partition | | OK | 04-04-2024 15:35:44 | 960d 1h 38m 9s | 1/3 | DISK OK - free space: / 33 GB (75% inode=93%): |
| SSH | | OK | 04-04-2024 15:36:53 | 960d 1h 36m 53s | 1/3 | SSH OK - OpenSSH_7.9p1 Debian-10+deb10u2 (protocol 2.0) |
| SWAP | | OK | 04-04-2024 15:37:58 | 559d 19h 31m 3s | 1/3 | SWAP OK - 87% free (847 MB out of 979 MB) |



# Was versteht man unter Observability?

Als Observability bezeichnet man die Fähigkeit, die internen Zustände eines Systems zu messen, indem man seine Ausgabewerte untersucht. Die Observability eines Systems gilt als gegeben, wenn der aktuelle Zustand rein durch Informationen aus Ausgabewerten, also Sensordaten, bewertet werden kann...

In diesem Kontext werden zum Erreichen von Observability drei Arten von Telemetriedaten genutzt:

01 Metriken

02 Logs

03 Traces

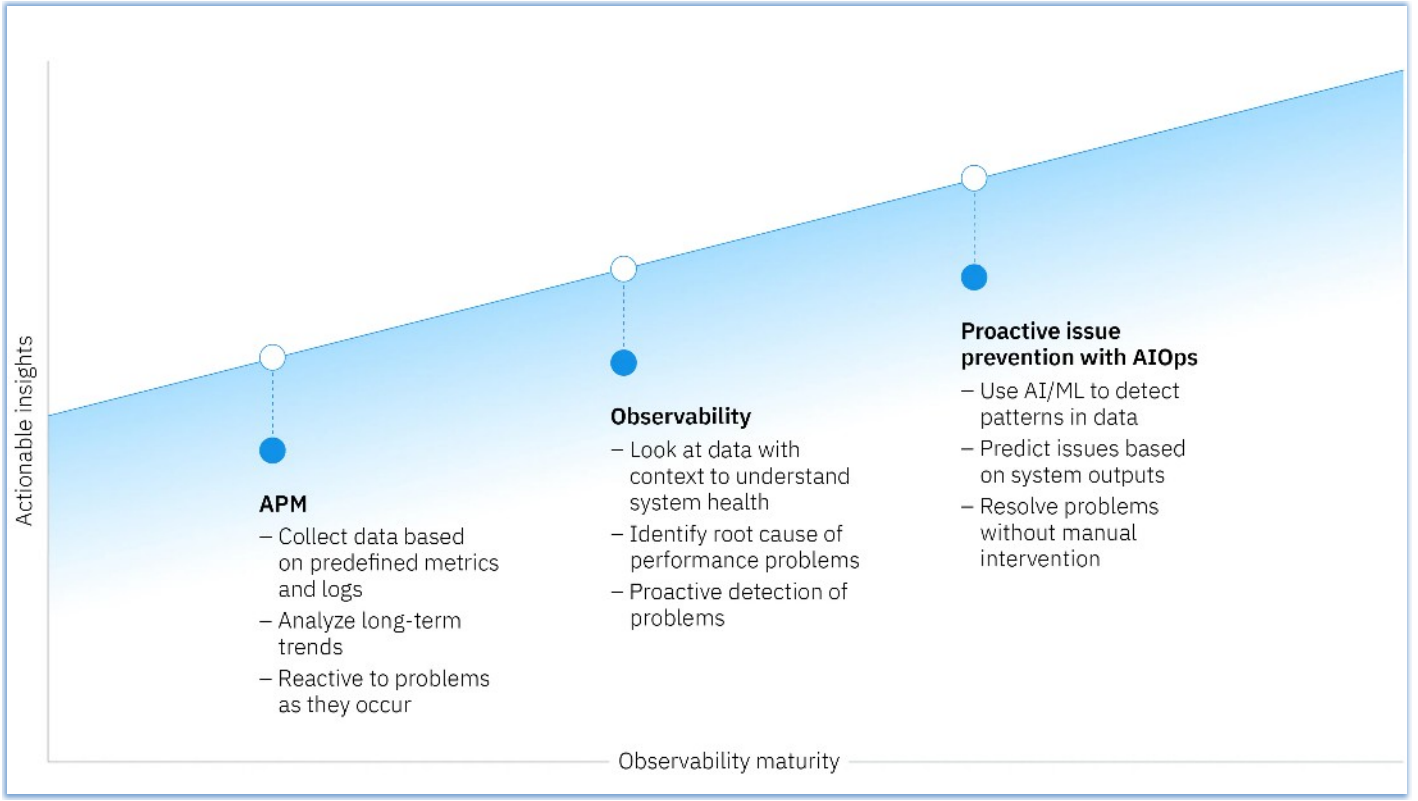
# Was versteht man unter Observability?

Die Telemetriedaten, die eine Observability-Plattform erfasst, können in verschiedene Typen unterteilt werden:

- **Metriken**
  - eine ***numerische*** Bewertung der Anwendungsleistung, der Ressourcennutzung und des allgemeinen Systemzustands über einen bestimmten Zeitraum.
- **Traces**
  - eine Aufzeichnung der End-to-End-Dienste für jede Benutzeranfrage, wenn Transaktionen von einem Dienst zum anderen wechseln.
- **Abhängigkeiten**
  - eine Bewertung, wie jede Anwendungskomponente von anderen Komponenten, Anwendungen und IT-Ressourcen abhängt.
- **Statusprüfungen**
  - regelmäßige Umfragen zu bestimmten Diensten. Wenn eine Statusprüfung fehlschlägt, wird daraus ein Problem.
- **Alerts**
  - Benachrichtigungen, die ausgelöst werden, wenn bestimmte vordefinierte Schwellenwerte überschritten werden.
- **Dashboards**
  - Anwendungsperspektiven, die visuelle, interaktive und verständliche Darstellungen bestimmter vorgegebener Metriken bieten.
- **Logs**
  - detaillierte, zeitgestempelte, vollständige und unveränderliche Aufzeichnungen von Anwendungsereignissen in Ihrem System.

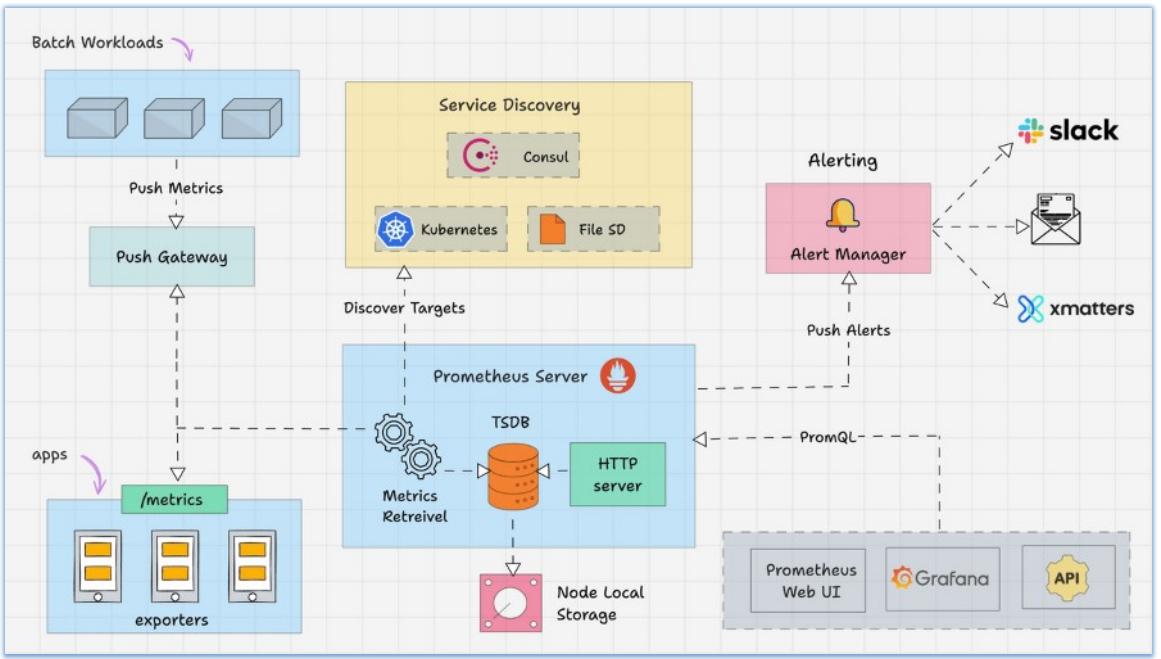
    Quelle: https://www.splunk.com/

# Was versteht man unter Observability?




Quelle: www.ibm.com/

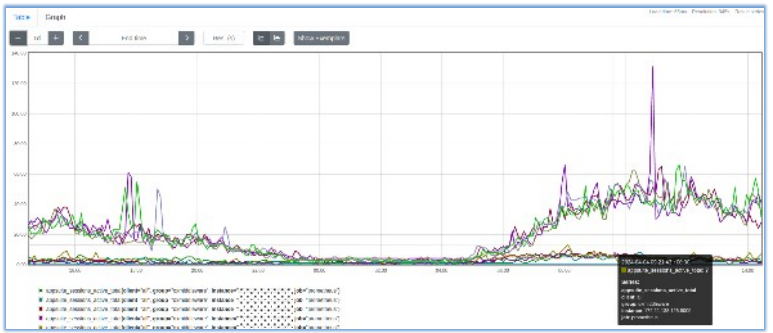
# Aufbau Prometheus

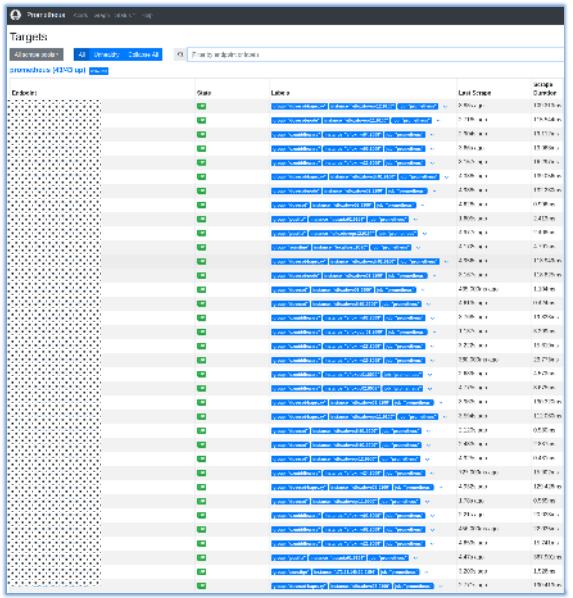


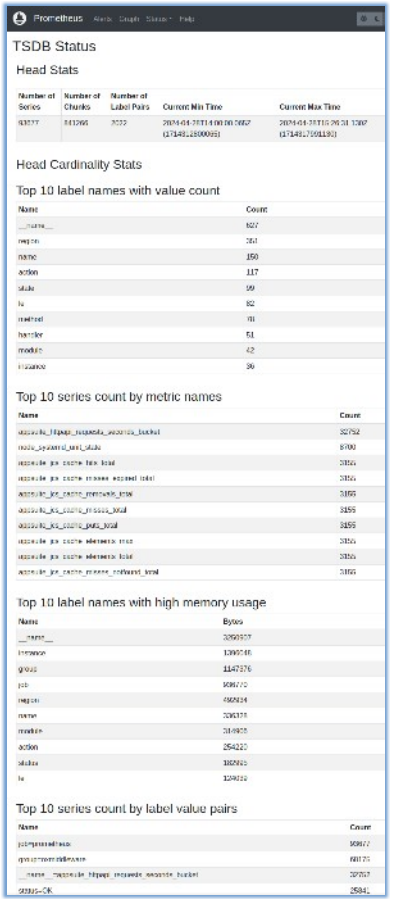

Quelle: devopscube.com

# Prometheus UI

- Keine Möglichkeit der Konfiguration
- Nur wenige Übersichten
- Keine richtigen Dashboards
  (nur ‚*console templates*')

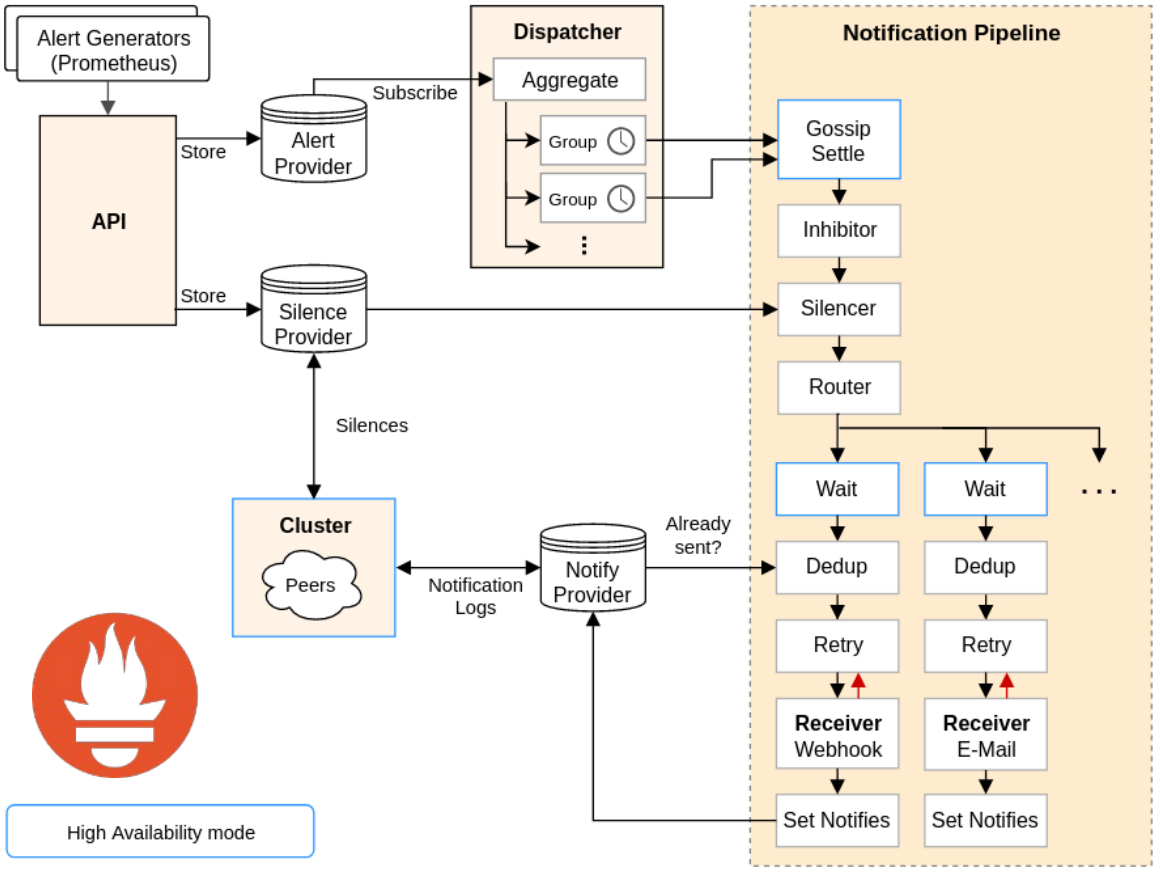




# Alertmanager

- Sendet Alerts an verschiedene Backends, z.B.:
  - Mail
  - MS Teams
  - Telegram
  - Webhook (generic)
- Konfiguration der Backends über alertmanager
- Management auch über CLI (***‚amtool‘***)
- Konfiguration der Benachrichtigungsregeln ***‚Rules‘***



# Was kann Prometheus denn besser?

- Verwendung von (zentralen) Regeln für Alerts: ***‚rules‘***
- Altertmanager getrennt
  Allgemein: Modularer Aufbau
- Timeseries-DB als Backend
- Zugriff über PromQL

```
sum(rate(http_requests_total{job="prometheus"}[5d]))
```

- **Verarbeitet Metriken**
- Skalierbarkeit schon im Produkt:
  - Föderation hierarchisch
  - Föderation ‚Cross-Service‘
  - HA auf allen Ebenen integriert

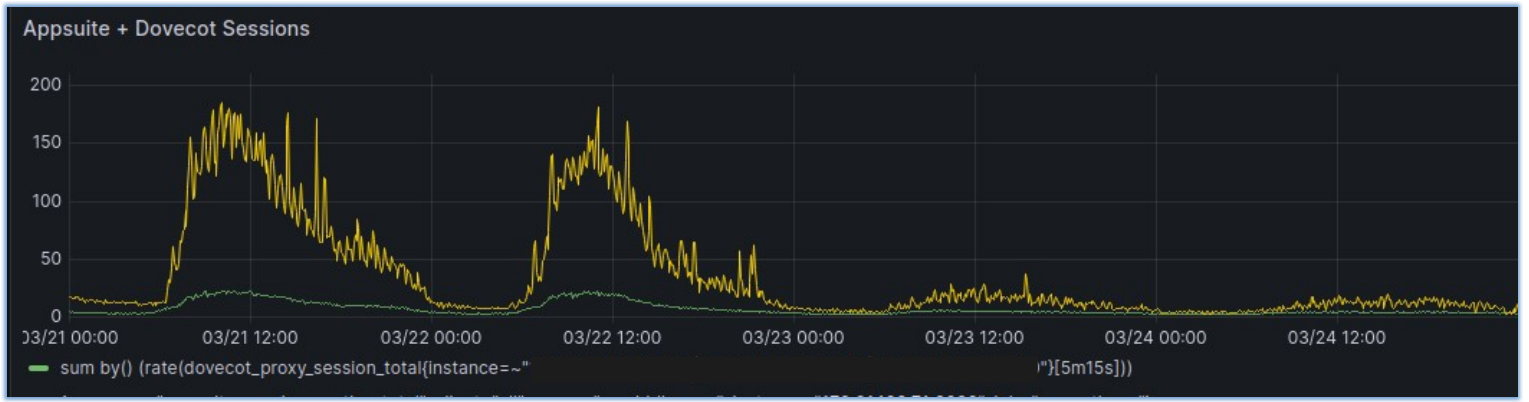
# Die geniale Erweiterung: Grafana

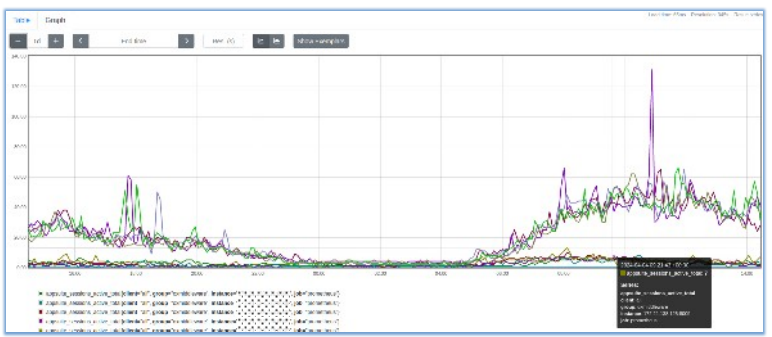
Einfache Erstellung von Dashboards

Benutzer-, Gruppen- und Rollenverwaltung

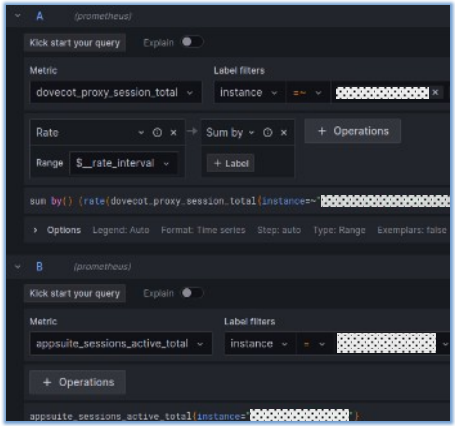
Visualisierung vieler weiteren Daten(banken) möglich

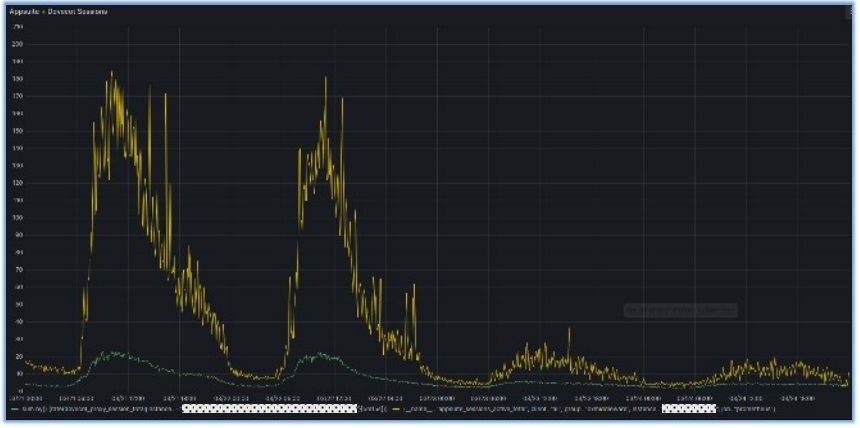
# Die geniale Erweiterung: Grafana

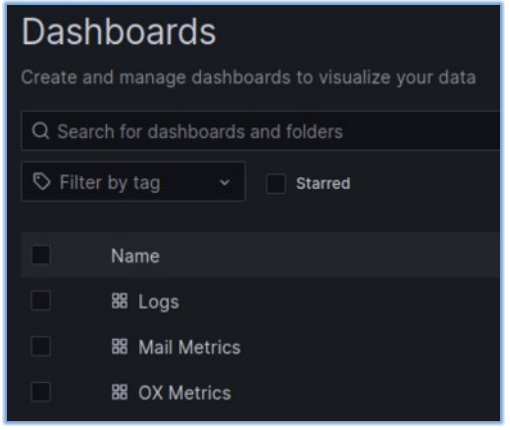


# Die geniale Erweiterung: Grafana

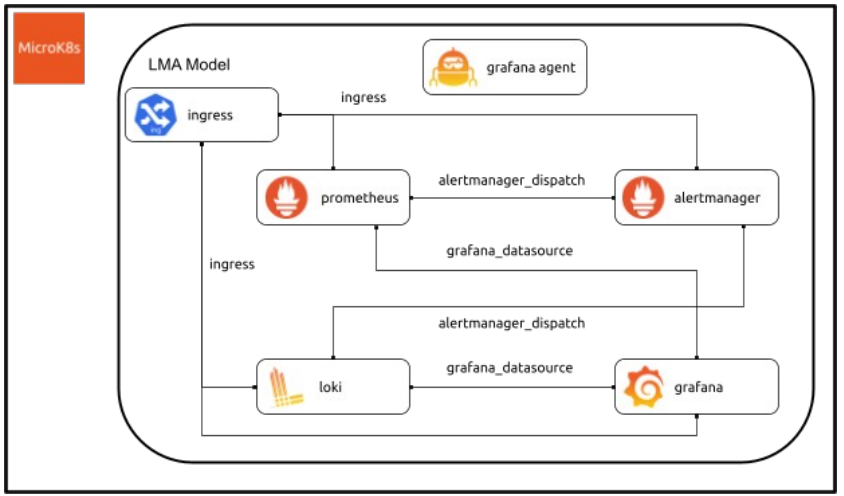





# Zentrales Log-Management mit Grafana-LOKI

Zentrales Log-Management mit Grafana-LOKI

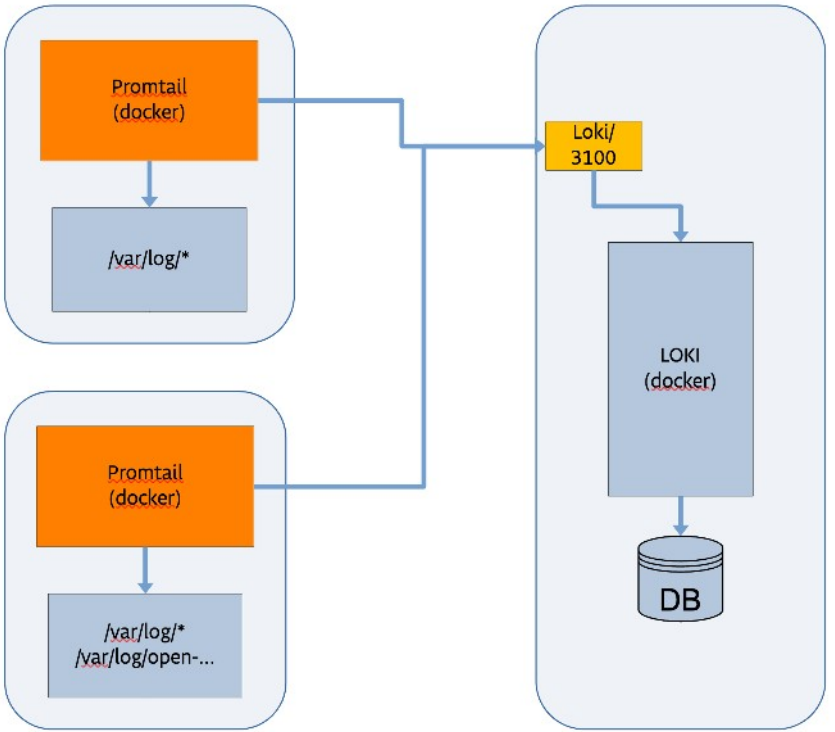
Grafana-Loki im Kontext:




Quelle: https://ubuntu.com/blog/tag/grafana-loki

# Zentrales Log-Management mit Grafana-LOKI

- Serverdienst: ***Grafana-LOKI***
- Agent: ***Promtail***
- „beobachtet“ Logfiles
- Sendet Log-Einträge an LOKI
- Kommt mit vielen Log-Formaten zurecht
- In docker-Container mit Zugriff auf /var/log



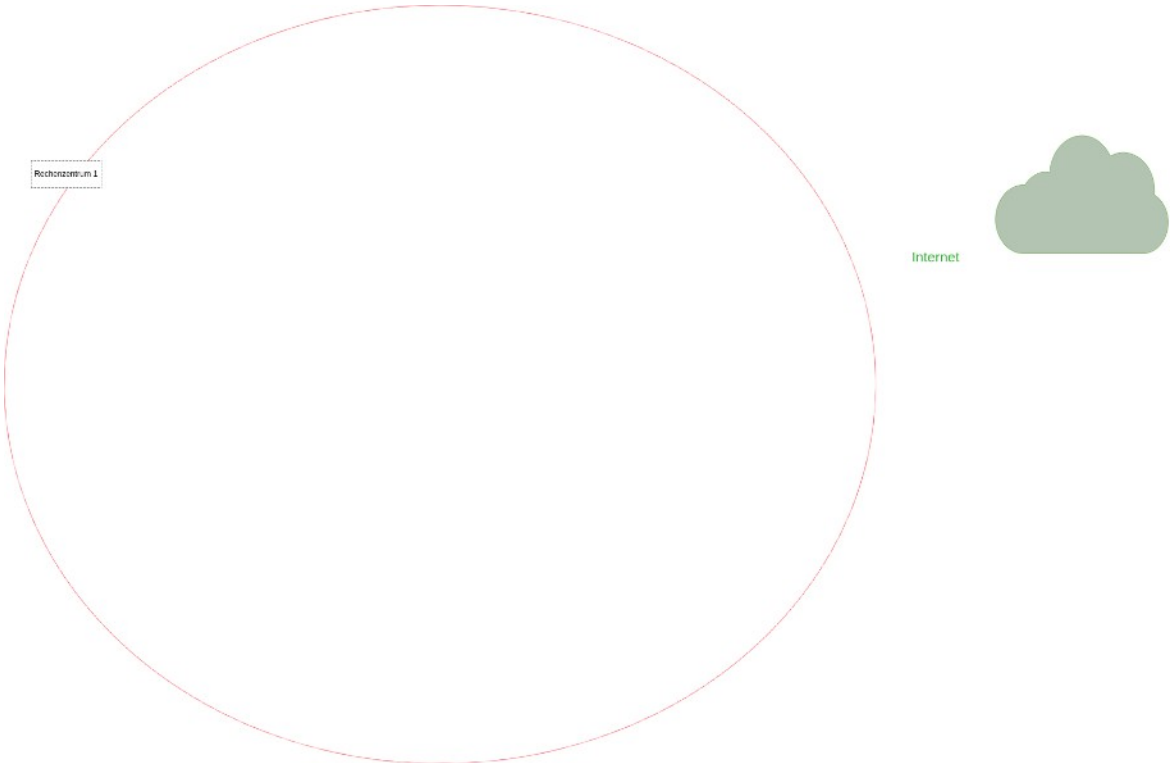
# Beispiel: Mail- / Groupwarestack

- Mail- / Groupwaresystem für ca. 12.000 BenutzerInnen
- Zentral genutzte Techniken:
  - Dovecot Pro (IMAP)
  - Postfix
  - OpenLDAP (Univention Corporate Server)
  - HAProxy
  - Apache (als Reverse Proxy)
  - OX Appsuite (Mail-/Groupware)
- ca. 30 VMs auf vSphere-Umgebung
- Abgegrenzte Umgebung, weitestgehend unabhängig von externen Systemen


→ **Sehr gutes Beispiel für Monitoring vs. Observability**

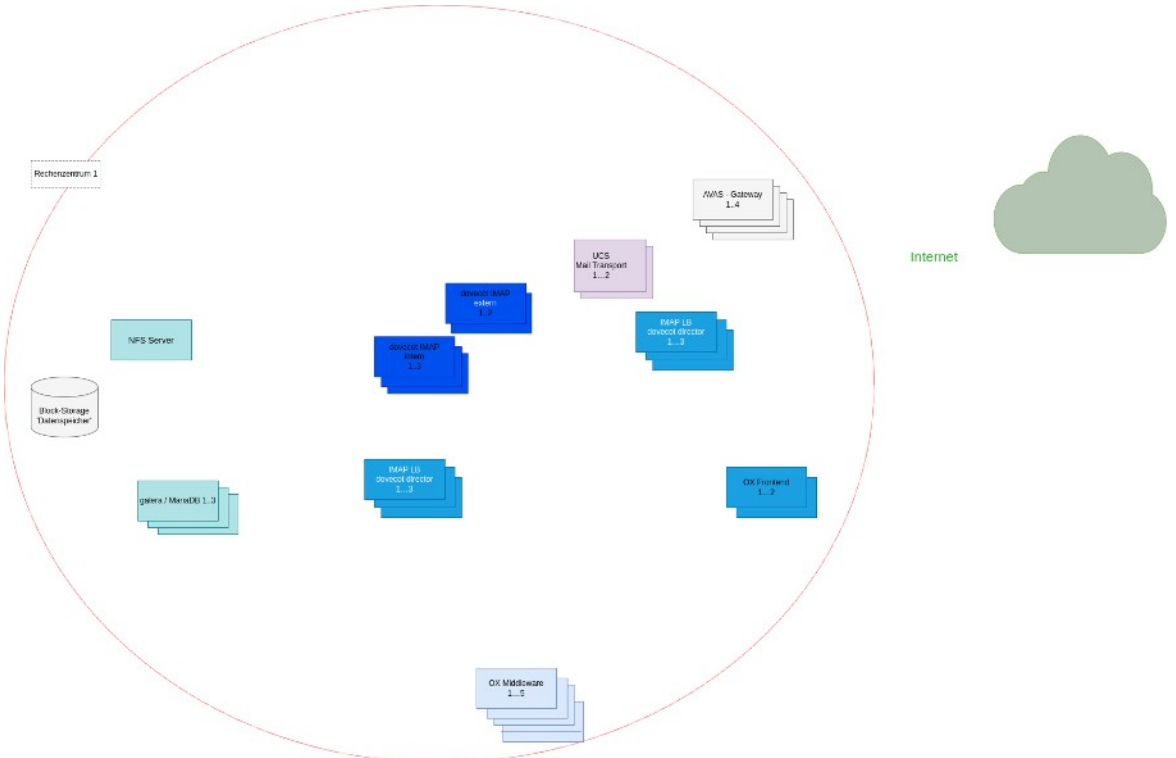
# Beispiel: Mail- / Groupwarestack

## Übersicht



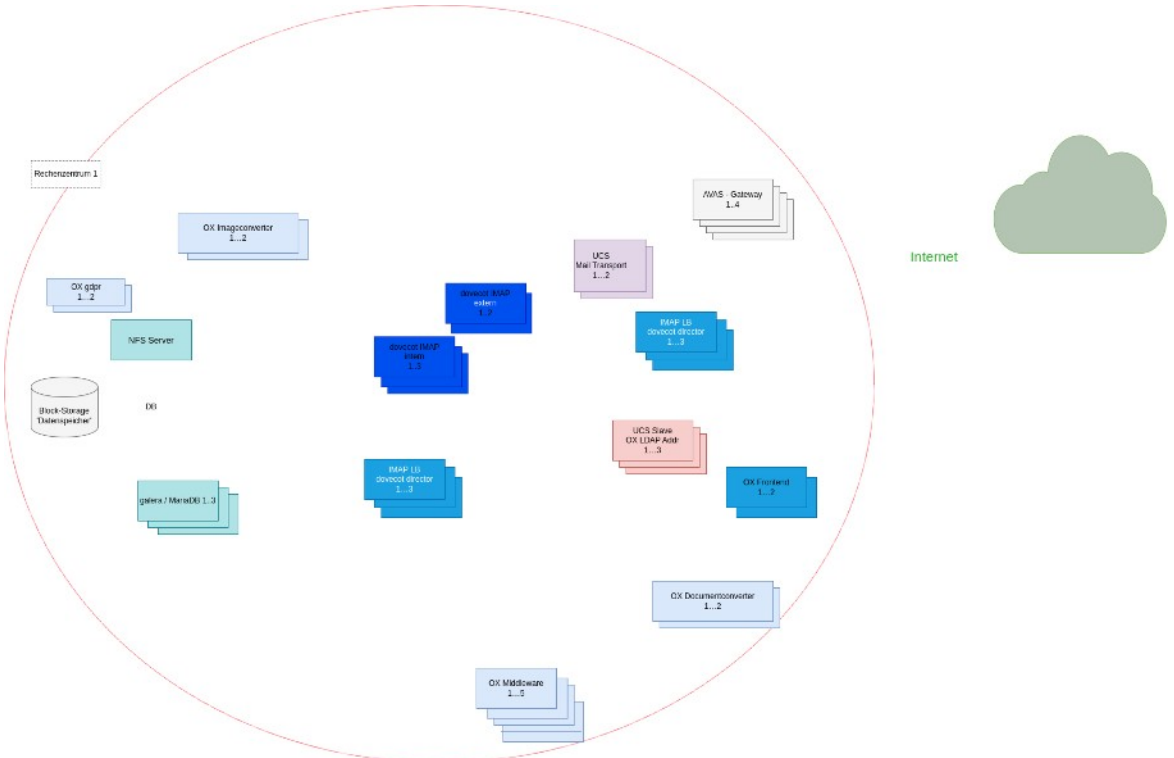
# Beispiel: Mail- / Groupwarestack

## Übersicht



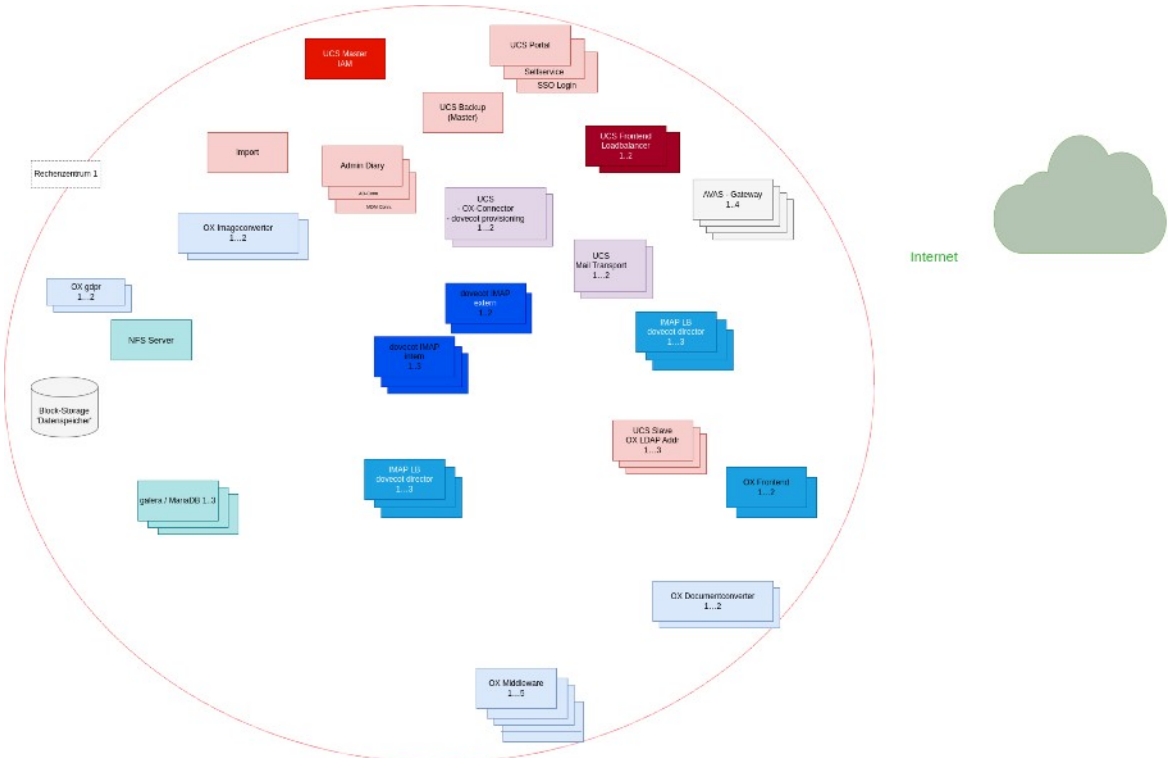
# Beispiel: Mail- / Groupwarestack

## Übersicht



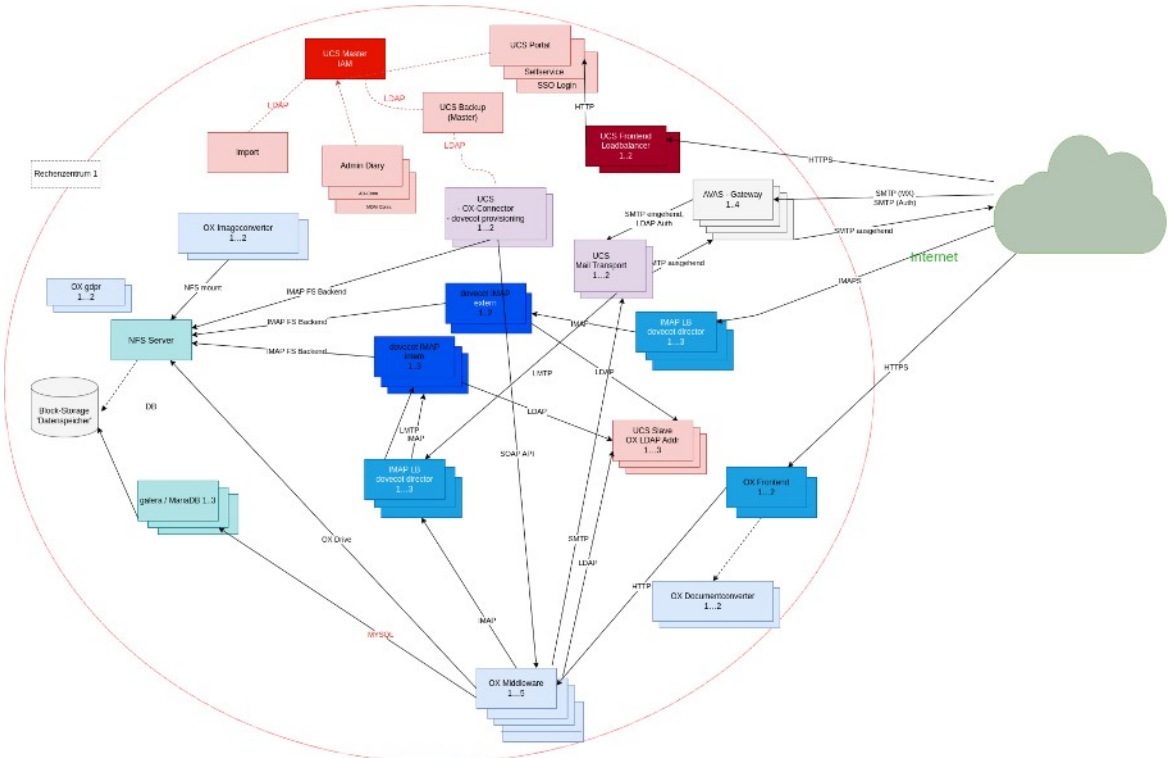
# Beispiel: Mail- / Groupwarestack

## Übersicht



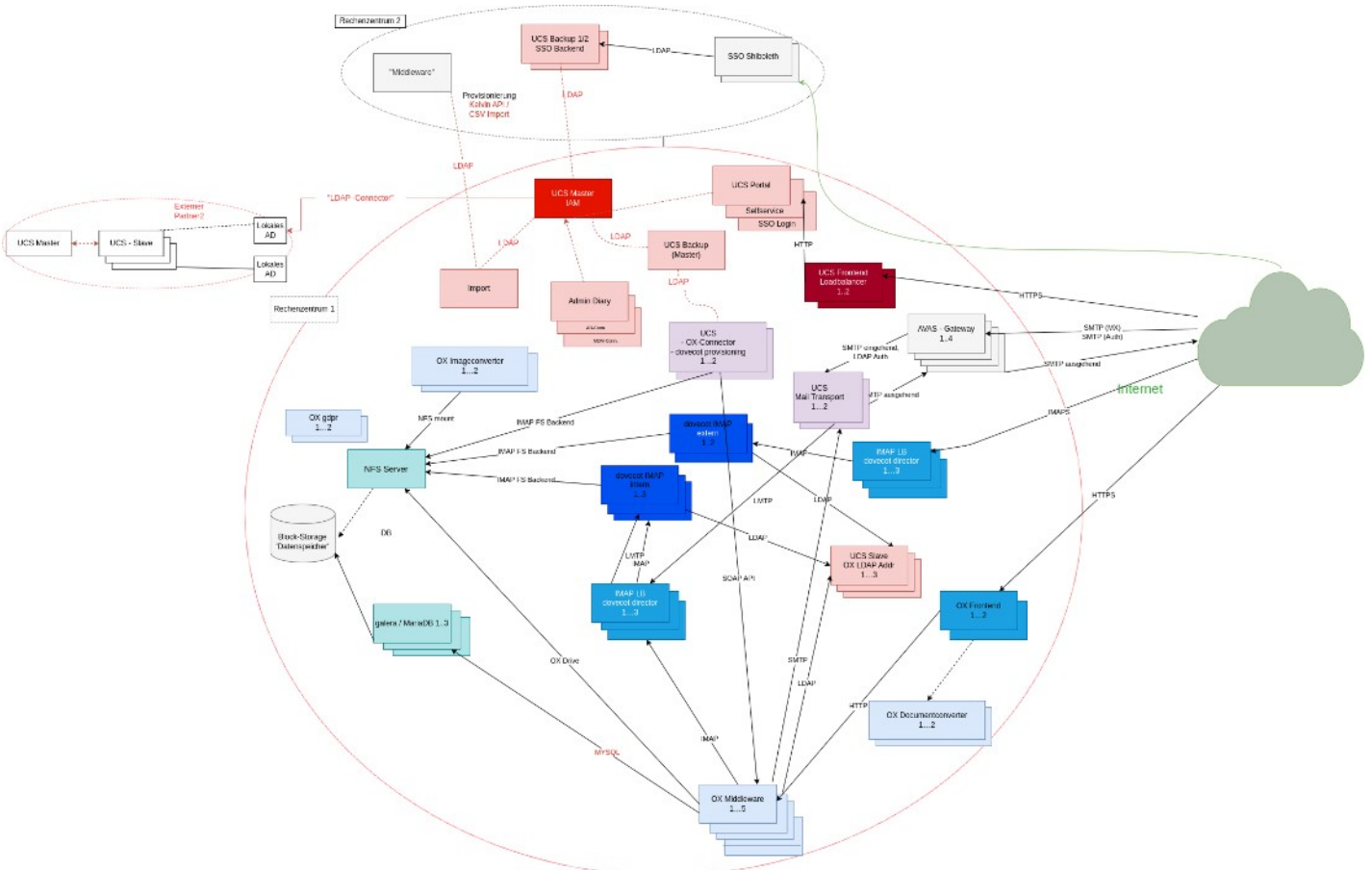
# Beispiel: Mail- / Groupwarestack

## Übersicht



# Beispiel: Mail- / Groupwarestack

## Übersicht



# Beispiel: Mail- / Groupwarestack

**01**

Welche Systeme sind zu beobachten?

Alle Systeme, die mit dem Betrieb in Relation stehen (im Beispiel ca. 25)

**02**

Welche Dienste sind zu beobachten?

- Mail-Schnittstellen (SMTP, IMAP)
- Gateways
- Backend-Dienste (NFS, MariaDB, LDAP)
- Frontend-Dienste (OX Appsuite)

Es fehlt an ‚best practises‘

**03**

Was ist interessant?

- Systemzustand der VM
- Anzahl Aktiver Sitzungen
- Anzahl Fehler
- Anzahl transportierter Mails
- Größe der Mail-Queues

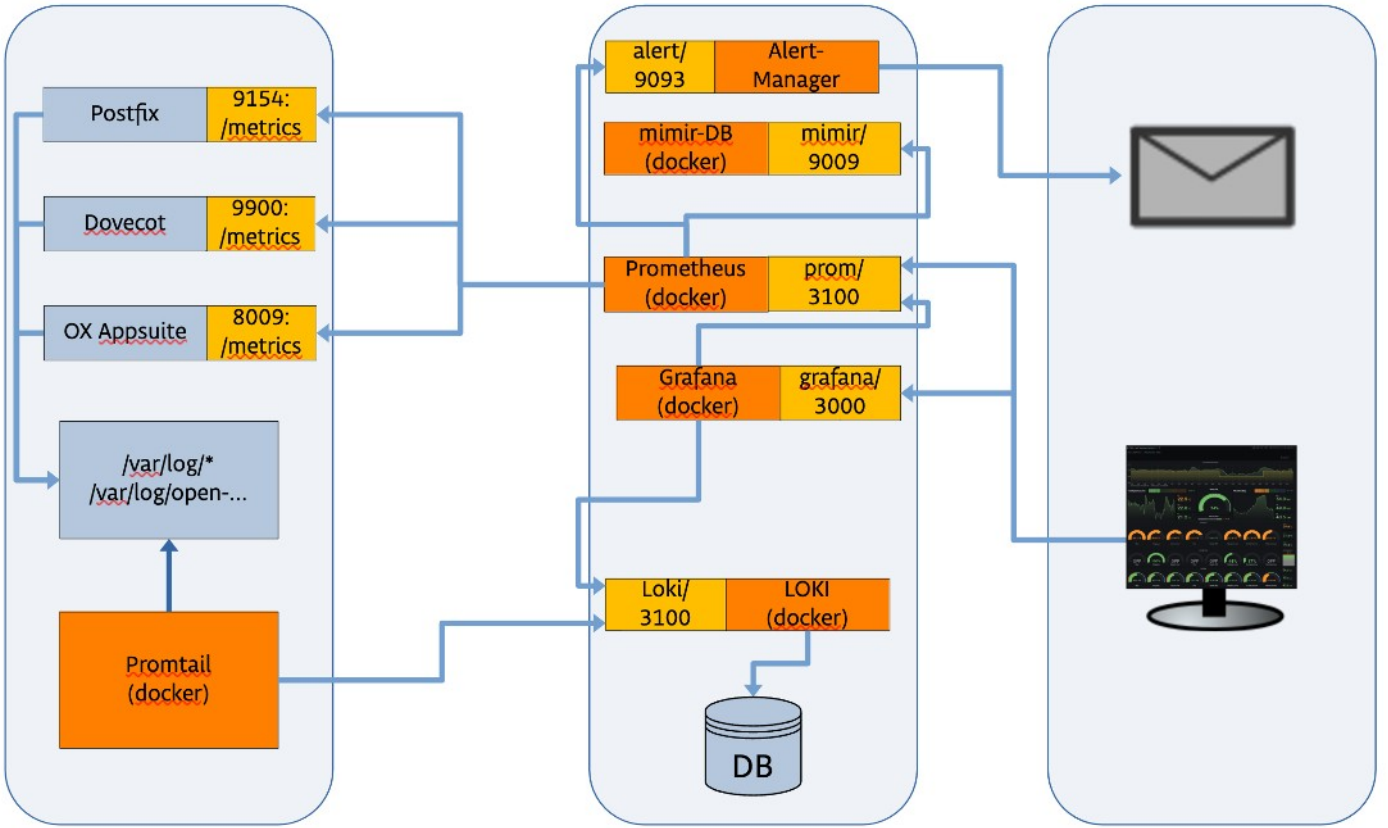
**04**

Wozu gibt es Metriken?

- Große Anzahl an Metriken verfügbar → *Unübersichtlich*
- Dokumentation ‚durchwachsen‘
- Teilweise abhängig von Herstellern → *schlecht*

# Beispiel: Mail- / Groupwarestack

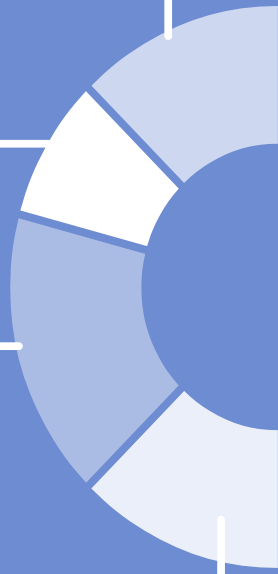
## Übersicht



# Beispiel: Mail- / Groupwarestack

## Konfigurationsbeispiele und Walkthrough

- Alle Serverdienste über Docker Container bereit gestellt
- Verwendung von ‚docker compose' zur Verwaltung der Container/Images
- Minimaler Installationsaufwand
- Für alle Betriebssysteme (die docker unterstützen) identisch



# Beispiel: Mail- / Groupwarestack

## Konfigurationsbeispiele und Walkthrough

- Prometheus und Mimir-DB

```
---
version: "3.9"

networks:
  grafanet:

services:
  mimir:
    image: grafana/mimir:latest
    volumes:
      - ./config/mimir-prometheus.yaml:/etc/mimir/mimir-prometheus.yaml
      - /var/lib/mimir:/var/lib/mimir
    ports:
      - 9009:9009
    command: --config.file=/etc/mimir/mimir-prometheus.yaml
    networks:
      - grafanet

  prometheus:
    image: prom/prometheus
    volumes:
      - ./config/prometheus.yml:/etc/prometheus/prometheus.yml
      - /var/lib/prometheus:/prometheus
    ports:
      - 9090:9090
    networks:
      - grafanet
```

# Beispiel: Mail- / Groupwarestack

## Konfigurationsbeispiele und Walkthrough

- Grafana Mimir-DB als Backend

```
multitenancy_enabled: false

blocks_storage:
  backend: filesystem
  bucket_store:
    sync_dir: /var/lib/mimir/tsdb-sync
  filesystem:
    dir: /var/lib/mimir/data/tsdb
  tsdb:
    dir: /var/lib/mimir/tsdb

compactor:
  data_dir: /var/lib/mimir/compactor
  sharding_ring:
    kvstore:
      store: memberlist

distributor:
  ring:
    instance_addr: 127.0.0.1
    kvstore:
      store: memberlist
```

```
ingester:
  ring:
    instance_addr: 127.0.0.1
    kvstore:
      store: memberlist
    replication_factor: 1

ruler_storage:
  backend: filesystem
  filesystem:
    dir: /var/lib/mimir/rules

server:
  http_listen_port: 9009
  log_level: error

store_gateway:
  sharding_ring:
    replication_factor: 1
```

# Beispiel: Mail- / Groupwarestack

## Konfigurationsbeispiele und Walkthrough

- Prometheus und Mimir-DB (statische Konfiguration)

```
global:
  scrape_interval:     15s
  external_labels:
    monitor: 'codelab-monitor'
remote_write:
  - url: http://                    /api/v1/push

scrape_configs:
  - job_name: 'prometheus'

    scrape_interval: 5s

    static_configs:
      - targets: ['… …']
        labels:
          group: '…'
        …
```

# Beispiel: Mail- / Groupwarestack

## Konfigurationsbeispiele und Walkthrough

- Prometheus und mimir-DB (dynamische Konfiguration)
- Konfiguration über json-File(s)
- Reload der Konfiguration erfolgt automatisch bei Änderung

```
- job_name: 'prometheus'
  scrape_interval: 5s
  file_sd_configs:
  - files:
    - scrape-configs/*.json
```

```
prometheus-dovecot-haproxy.json
prometheus-dovecot.json
prometheus-dovecot-node.json
prometheus-oxmiddleware.json
prometheus-postfix.json
prometheus-sonstige.json
```

```
[
  {
    "Labels": {
      "group": "postfix"
    },
    "targets": [
      "ucsmta01:9154", "ucsmta02:9154",
    ]
  }
]
```

# Beispiel: Mail- / Groupwarestack

## Konfigurationsbeispiele und Walkthrough

- Konfiguration:
  - Alertmanager
  - Prometheus
  - Rules

```
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      -
```

```
global:
  smtp_smarthost:
  smtp_from: 'alertmanager@iku.systems'
route:
  receiver: 'team-X-mails'
  group_by: ['alertname', 'cluster']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 3h
  routes:
  - matchers:
    - group="oxmiddleware"
    receiver: team-X-mails
    routes:
    - matchers:
      - severity="critical"
      receiver: team-X-pager
receivers:
- name: 'team-X-mails'
  email_configs:
  - to: 'team-X@iku.systems'
```

# Alertmanager / Prometheus Rules

- Konfiguration der Benachrichtigungsregeln ***‚Rules‘*** in prometheus:

prometheus.yaml:

```
rule_files:
  - rules/*.yaml
```

```
rules/dovecot-load.yaml
rules/dovecot-sessions.yaml
rules/ox-sessions.yaml
rules/ox-threads.yaml
rules/postfix-mailq.yaml
rules/postfix-submission.yaml
rules/…
```

```
groups:
  - name: dovecot_alerts
    rules:
    - alert: TooManyDovecotSessions
      expr: rate(dovecot_proxy_session_total[1m]) > 19
      for: 5m
      labels:
        severity: critical
      annotations:
        summary: Too Many Dovecot Sessions on host {{ $labels.instance }}
        description: The Number of Dovecot Sessions on host {{ $labels.instance }} has
exceeded 19 for 1 minute.
```

***ACHTUNG:***
***Reload der Prometheus-Konfiguration nötig!***

# Beispiel: Mail- / Groupwarestack

## Konfigurationsbeispiele und Walkthrough

- Grafana

```
---
version: "3.9"

networks:
  grafanet:

services:
  grafana:
    image: grafana/grafana-enterprise
    ports:
      - "3000:3000"
    volumes:
      - /var/lib/grafana:/var/lib/grafana
    healthcheck:
      test: [ "CMD-SHELL", "wget --no-verbose --tries=1 --spider http://l
th || exit 1" ]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - grafanet
```

# Beispiel: Mail- / Groupwarestack

## Konfigurationsbeispiele und Walkthrough

- LOKI

```
---
version: "3.9"

networks:
  grafanet:

services:
  loki:
    image: grafana/loki:2.9.2
    command: "-config.file=/etc/loki/config.yaml"
    ports:
      - 3100:3100
      - 7946
      - 9095
    volumes:
      - ./config/loki-config.yaml:/etc/loki/config.yaml
      - /var/lib/loki:/tmp/loki
    healthcheck:
      test: [ "CMD-SHELL", "wget --no-verbose --tries=1 --spider http://l
 exit 1" ]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - grafanet
```

# Beispiel: Mail- / Groupwarestack

## Konfigurationsbeispiele und Walkthrough

- LOKI

```
auth_enabled: false

server:
  http_listen_port: 3100
  grpc_listen_port: 9096

common:
  instance_addr: 127.0.0.1
  path_prefix: /tmp/loki
  storage:
    filesystem:
      chunks_directory: /tmp/loki/chunks
      rules_directory: /tmp/loki/rules
  replication_factor: 1
  ring:
    kvstore:
      store: inmemory

query_range:
  results_cache:
    cache:
      embedded_cache:
        enabled: true
        max_size_mb: 100
```

```
schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb-shipper
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: 24h

ruler:
  alertmanager_url: http://localhost:9093
```

# Beispiel: Mail- / Groupwarestack

## Konfigurationsbeispiele und Walkthrough - Exporter

- OS:
  prometheus-node-exporter_*.deb
- Postfix:
  prometheus-postfix-exporter_*.deb
- OX: integriert (/metrics)
- Dovecot:

```
service stats {
  inet_listener http {
    port = 9900
  }
}
```

# Grafana LOKI: Konfiguration Promtail

- Konfiguration Promtail

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://1.2.3.4:3100/loki/api/v1/push

scrape_configs:
- job_name: system
  static_configs:
  - targets:
      - localhost
    labels:
      job: varlogs
      __path__: /var/log/*log

  - targets:
      - localhost
    labels:
      job: oxlogs-mw01
      __path__: /var/log/open-xchange/open-xchange.log.0
```

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://1.2.3.4:3100/loki/api/v1/push

scrape_configs:
- job_name: system
  static_configs:
  - targets:
      - localhost
    labels:
      job: varlogs-ucsmta01
      __path__: /var/log/*log
```

```
services:
  promtail:
    image: grafana/promtail:2.9.2
    ports:
      - 9080:9080
    volumes:
      - ./config/promtail-config.yaml:/etc/promtail/config.yaml:ro
      - /var/run/docker.sock:/var/run/docker.sock
      - /var/log:/var/log
    command: -config.file=/etc/promtail/config.yaml
```

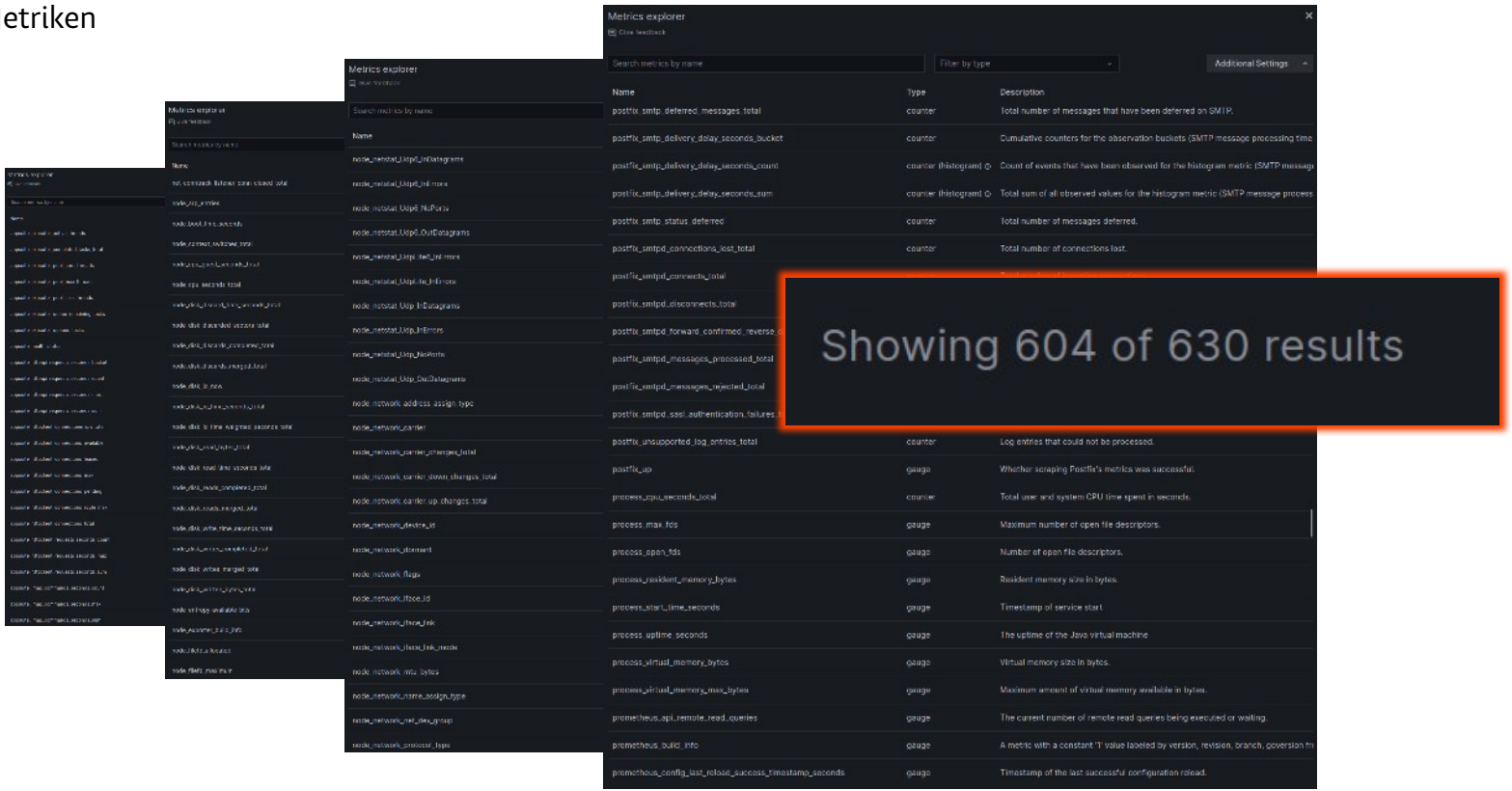
# Beispiel: Mail- / Groupwarestack

- Wie sehen Metriken aus?
- wget http://<host>:9900/metrics

```
# HELP process_start_time_seconds Timestamp of service start
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1708789598
# HELP dovecot_build Dovecot build information
# TYPE dovecot_build info
dovecot_build_info{version="2.3.21",revision="955bff2155"} 1
# HELP dovecot_imap_command Total number of all events of this kind
# TYPE dovecot_imap_command counter
dovecot_imap_command_total{cmd_name="CAPABILITY"} 7533436
dovecot_imap_command_total{cmd_name="CAPABILITY",tagged_reply_state="OK"} 7533436
dovecot_imap_command_total{cmd_name="LIST"} 7701185
dovecot_imap_command_total{cmd_name="LIST",tagged_reply_state="OK"} 7701182
dovecot_imap_command_total{cmd_name="LSUB"} 1141074
dovecot_imap_command_total{cmd_name="LSUB",tagged_reply_state="OK"} 1141074
dovecot_imap_command_total{cmd_name="NOOP"} 4349851
dovecot_imap_command_total{cmd_name="NOOP",tagged_reply_state="OK"} 4349851
dovecot_imap_command_total{cmd_name="ENABLE"} 1046767
dovecot_imap_command_total{cmd_name="ENABLE",tagged_reply_state="OK"} 1046767
dovecot_imap_command_total{cmd_name="EXAMINE"} 7059110
dovecot_imap_command_total{cmd_name="EXAMINE",tagged_reply_state="OK"} 7058671
dovecot_imap_command_total{cmd_name="EXAMINE",tagged_reply_state="NO"} 439
dovecot_imap_command_total{cmd_name="SORT"} 3441044
dovecot_imap_command_total{cmd_name="SORT",tagged_reply_state="OK"} 3441033
dovecot_imap_command_total{cmd_name="CLOSE"} 6935365
dovecot_imap_command_total{cmd_name="CLOSE",tagged_reply_state="OK"} 6935365
dovecot_imap_command_total{cmd_name="STATUS"} 25915037
dovecot_imap_command_total{cmd_name="STATUS",tagged_reply_state="OK"} 25914976
dovecot_imap_command_total{cmd_name="STATUS",tagged_reply_state="NO"} 53
dovecot_imap_command_total{cmd_name="ID"} 3060555
dovecot_imap_command_total{cmd_name="ID",tagged_reply_state="OK"} 3060555
…
```

# Beispiel: Mail- / Groupwarestack

- Alle Metriken



# Eigene Metriken

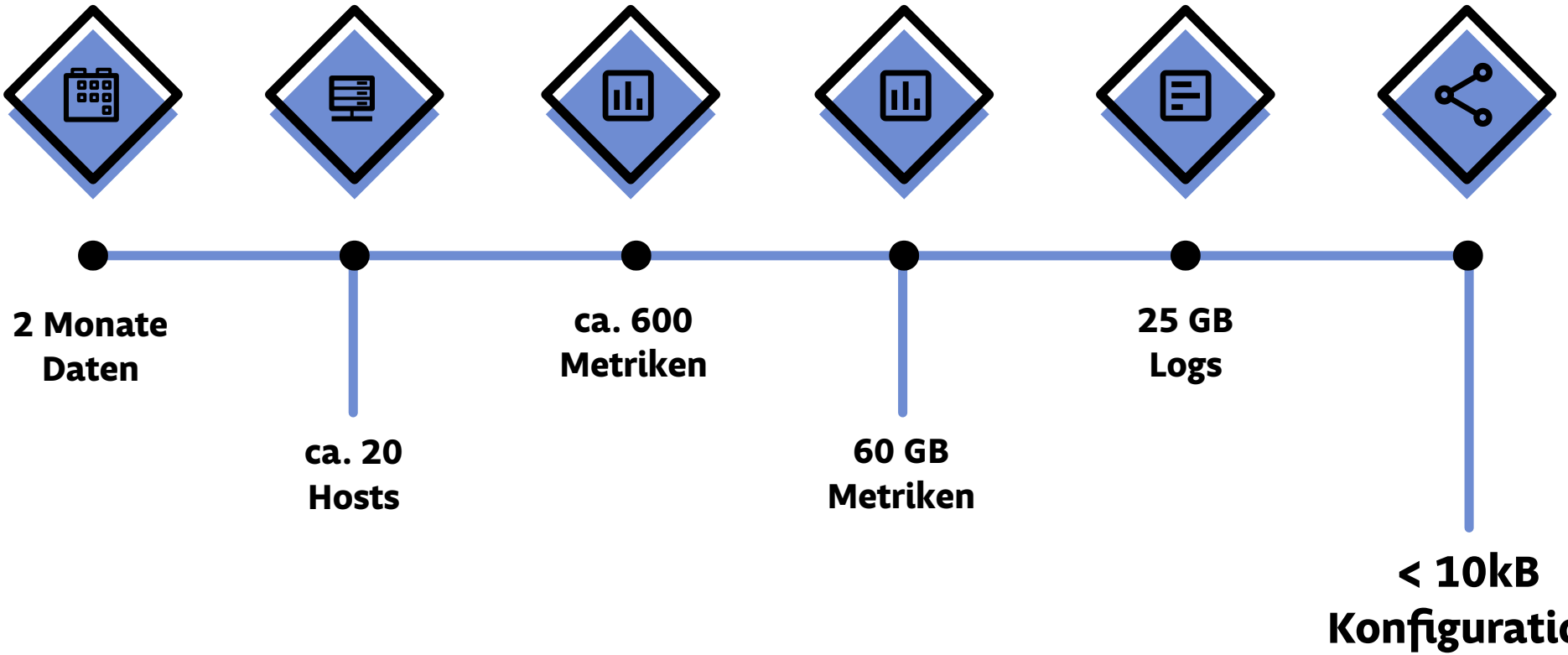
- Anforderungen:
  - Muss nur eine definiertes Textfile unter /metrics liefern
  - Templates/Libraries für verschiedene Programmiersprachen verfügbar.

- Flexibler Einsatz:
  - Klassische Monitoring-Anwendungen
  - Viele weitere Anwendungen denkbar, z.B.: „Nagios → Prometheus Exporter“

```python
from prometheus_client import start_http_server, Summary
import random
import time

# Create a metric to track time spent and requests made.
REQUEST_TIME = Summary('request_processing_seconds', 'Time spent processing request')

# Decorate function with metric.
@REQUEST_TIME.time()
def process_request(t):
    """A dummy function that takes some time."""
    time.sleep(t)

if __name__ == '__main__':
    # Start up the server to expose the metrics.
    start_http_server(8000)
    # Generate some requests.
    while True:
        process_request(random.random())
```

Beispiel in python
Quelle: github.com/prometheus

# Erste Ergebnisse



# Grafana LOKI: Konfiguration Promtail

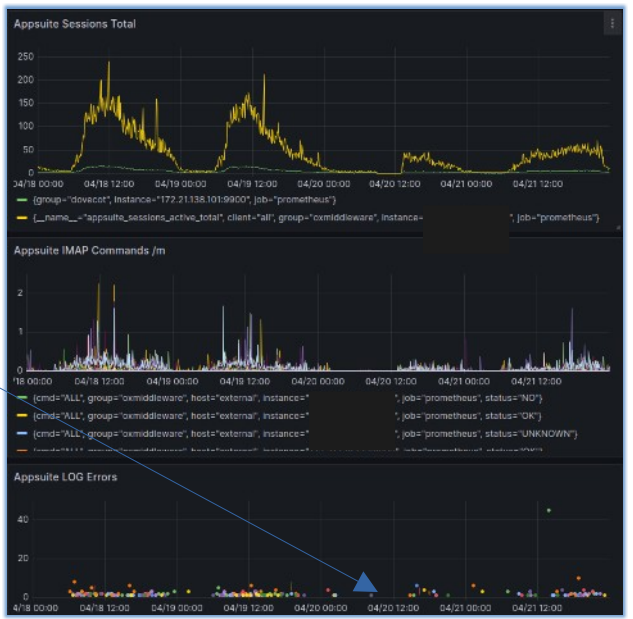
## Beispiele

„Dienstags“-Wartungsfenster:
18 bis 20 Uhr



Vollständiger Ausfall der Internetverbindung

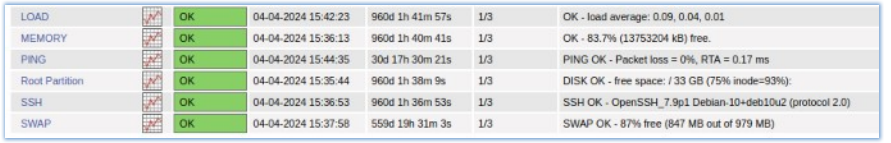
Keine Fehlermeldungen in den Logs



# Ist das nun besser? Monitoring <-> Observability

MONITORING

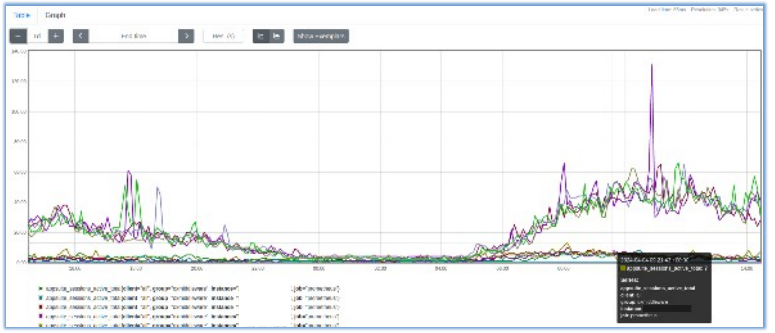
- ✔ Einfach einzurichten / geringe Komplexität
- ✔ Hohe Kompatibilität
- ✔ Umfangreiches Know-How bei vorhandenem Personal



| | | | | | | |
|---|---|---|---|---|---|---|
| LOAD | | OK | 04-04-2024 15:42:23 | 960d 1h 41m 57s | 1/3 | OK - load average: 0.09, 0.04, 0.01 |
| MEMORY | | OK | 04-04-2024 15:36:13 | 960d 1h 40m 41s | 1/3 | OK - 83.7% (13753204 kB) free. |
| PING | | OK | 04-04-2024 15:44:35 | 30d 17h 30m 21s | 1/3 | PING OK - Packet loss = 0%, RTA = 0.17 ms |
| Root Partition | | OK | 04-04-2024 15:35:44 | 960d 1h 38m 9s | 1/3 | DISK OK - free space: / 33 GB (75% inode=93%): |
| SSH | | OK | 04-04-2024 15:36:53 | 960d 1h 36m 53s | 1/3 | SSH OK - OpenSSH_7.9p1 Debian-10+deb10u2 (protocol 2.0) |
| SWAP | | OK | 04-04-2024 15:37:58 | 559d 19h 31m 3s | 1/3 | SWAP OK - 87% free (847 MB out of 979 MB) |

OBSERVABILITY

- ✔ Metriken statt Checks
- ? weniger fertige Pakete (Prometheus-exporter vs. NRPE-checks)



ERGEBNIS

- ? Keine Klare Trennung möglich
- ✔ In der Übergangszeit: Am besten Beides!

# Ist das nun besser? Monitoring <-> Observability

GELÖST

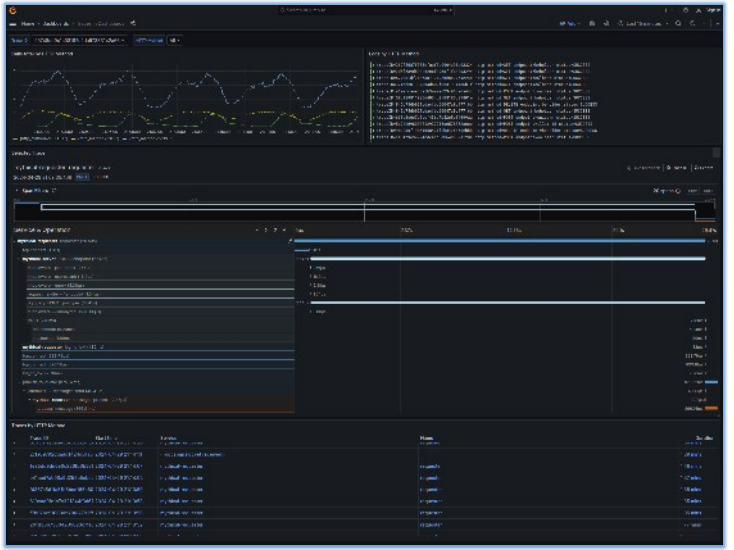
- ✔ Metriken (Prometheus)
- ✔ Abhängigkeiten (Prometheus / Nagios)
- ✔ Statusprüfungen (Prometheus / Nagios)
- ✔ Alerts (Prometheus / Nagios)
- ✔ Dashboards (Prometheus / Grafana / Nagios)
- ✔ Protokolle (Loki)

FEHLT NOCH

- ✗ Traces

# Weitere Möglichkeiten: Traces

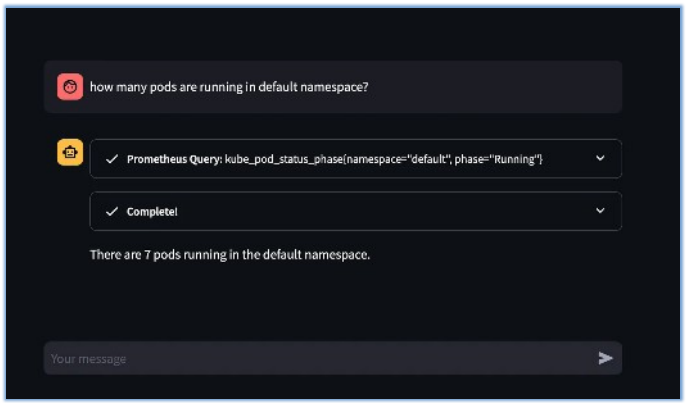
- Muss von den (Micro-)Services unterstützt werden
- HTTP-Header eignen sich sehr gut für Trace-Ids
- Reverse-Proxy ist ein guter Einstiegspunkt



- Installierbar als Container „Grafana Tempo“
- Fertige Beispielumgebung auf der Tempo-Webseite verfügbar
- Visualisierung über Grafana
  - **TraceQL** als Sprache

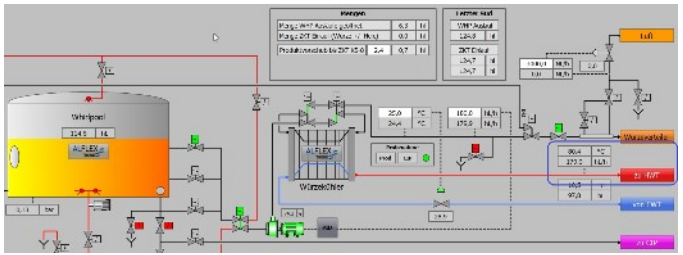
# Weitere Möglichkeiten: Einsatz von KI

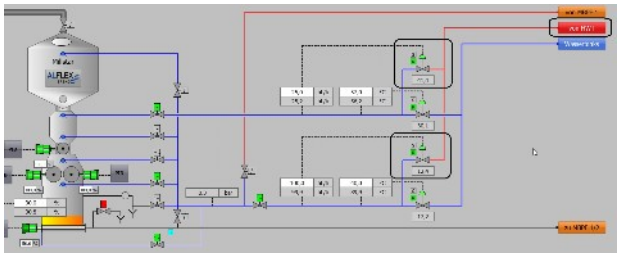
- Kommerzielle Angebote - unübersichtlich
- Interessante Ansätze:
  - Generelle Hilfe bei spezifischen Sprachen, z.B. PromQL
  - AI-Agent, der interaktiv Zustände/Metriken anfragt
  - AI-Agent, der selbständig eingreift oder alarmiert

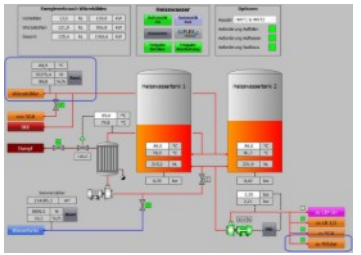



Quelle: https://blog.devops.dev

# Bier oder E-Mails? Für Grafana das Gleiche ;-)







IKU
SYSTEMS & SERVICES

A. Niederländer

# SLAC 2024

## MONITORING:
## PROMETHEUS / LOKI / GRAFANA

Differenzierte Metriken statt einfacher Checks

# DANKE!

# Quellen

LINKS

- Was versteht man unter Observability?:
  https://www.splunk.com/de_de/data-insider/what-is-observability.html
  https://www.ibm.com/de-de/resources/automate/observability-3-steps-to-start
- Alertmanager:
- https://github.com/prometheus/alertmanager?tab=readme-ov-file
- https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/
- Zentrales Log-Management mit Grafana-LOKI
- https://ubuntu.com/blog/tag/grafana-loki
- Eigene Metriken
- https://prometheus.github.io/client_python/getting-started/three-step-demo/
- Fertige Beispielumgebung Grafana Tempo: https://grafana.com/docs/tempo/latest/setup/linux/
- Visualisierung über Grafana: https://grafana.com/docs/grafana/latest/panels-visualizations/visualizations/traces/
- AI Agent:
  https://blog.devops.dev/transforming-monitoring-with-ai-building-a-langchain-agent-for-prometheus-7045eed214e2

BILDER

- https://grafana.com/grafana/dashboards/13699-tado-dashboard-celsius/