

Die Bibliothek in Zahlen

Metriken einfach selbstgemacht

Uwe Dierolf, KIT-Bibliothek, Karlsruhe

KIT-BIBLIOTHEK



Inhalt

- Motivation
 - „Wissen ist Macht, nichts wissen macht aber auch nichts“ oder
 - Warum machen Metriken Sinn?
- Technologie
 - Statsd
 - Graphite
 - Grafana
- Erfahrungen & Fazit

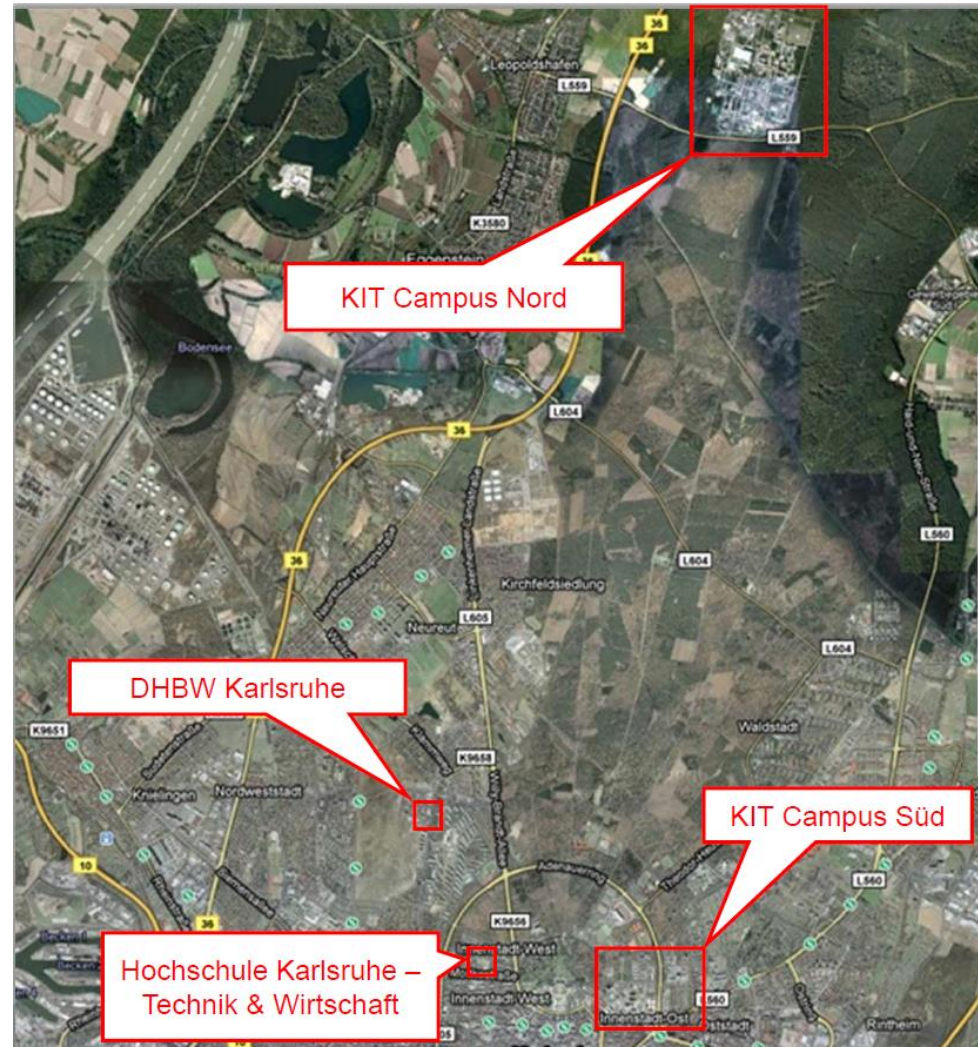
KIT-Bibliothek

- KIT = Karlsruher Institut für Technologie
- Bibliothek ist 24/7 seit April 2006
- 26.000 Studenten
- 10.000 Mitarbeiter

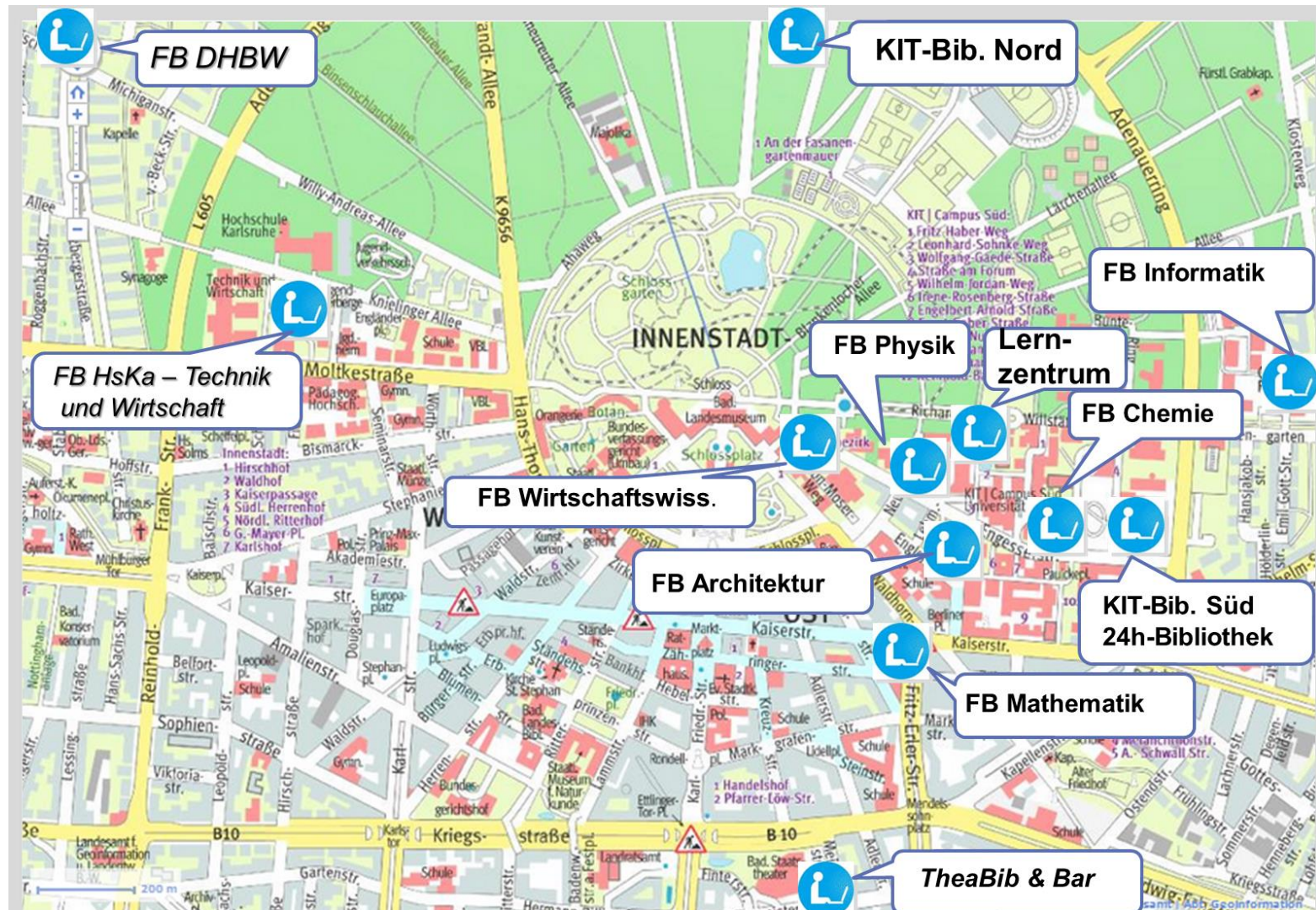


KIT-Bibliothek umfasst 4 Hauptstandorte

- KIT Campus Süd
- KIT Campus Nord
- Hochschule Karlsruhe
- DHBW Karlsruhe



KIT-Bibliothek betreute etliche Lernräume in Karlsruhe



Metrik

- Allgemeine Definition
 - Ein Messverfahren zur Messung quantifizierbarer Einheiten bzw. Größen
- Hier ganz einfach Namen
 - dot-delimited metric names
 - `criteria.sub-criteria.sub-sub-criteria`
 - Bsp `application.server.requests-per-second`
- Nutzen von Metriken
 - Analyse
 - Monitoring
 - rechtzeitige Fehlererkennung (Alerting)
 - nachträgliche Ursachenforschung (Forensik)
 - Kapazitätsplanung

Wie monitore ich richtig?

- Pull vs. Push
- Pull bzw. Poll
 - Polling agent
 - Typischer Vertreter solcher Lösungen: Munin oder Nagios
- **Push**
 - Metriken werden da, wo sie entstehen an eine zentrale Instanz gesendet
 - Da wo sie entstehen kann auch direkt im Programm bedeuten !
 - StatsD + Graphite

Die Komponenten

■ StatsD

- Nimmt Metrikwerte in Empfang
- Aggregiert diese und gibt sie verzögert an Graphite weiter

■ Graphite

- Verarbeitet und speichert Metrikwerte
- Bietet grafische Anzeige
- Bietet jede Menge Funktionen (Aggregation, Statistik etc.)

■ Grafana

- Visualisierungskomponente
- Grafische Anzeige deutlich besser als bei Graphite
- Einfache Nutzung der Funktionen von Graphite
- Dashboards !

StatsD I

■ Was ist es?

- “StatsD is originally a simple daemon developed and released by Etsy to aggregate and summarize application metrics.”
- Guter Einstieg: <https://github.com/etsy/statsd>

■ Aggregiert mehrere Metrikwerte intern vor Übergabe an Graphite

- Zeitraum währenddessen StatsD sammelt = „flush intervall“
- „flush intervall“ ist normalerweise 10 Sekunden
- Dieser Wert wird oft als „retention“ bezeichnet
- Der Wert von StatsD muss mit dem von Graphite übereinstimmen !

StatsD II

- Leichtgewichtig
 - Basiert auf node.js
- Erwartet Metrikwerte im Textformat
 - Format: `<metricname>:<value>|<type>`
 - Übertragung an StatsD erfolgt per UDP und ist daher non-blocking !
 - Integration in Software einfach möglich

Metriken I - counter

■ Counter

- Bei mehreren Quellen praktisch, um zu zählen wie oft etwas insgesamt passiert
- counter value
 - rate (counter values per second)
- Übergabe-String:
 - `kvk.request:1|c`



Metriken II - timer

■ Timers

- Damit misst man wie lange etwas dauert
- Aber auch wie oft etwas passiert
 - Bsp: Anzahl Kataloge pro KVK-Anfrage
- Neben dem eigentlichen Wert werden alle möglichen Statistikwerte dazu von StatsD ermittelt und an Graphite übertragen
 - Aus einer Metrik werden intern sehr viele !
 - Mean, sum, count, min, max, 90%
 - Diese Werte beziehen sich immer aufs „flush interval“
 - Vorsicht: Plattenplatz !
- Übergabe-String:
 - Bsp: `kvk.catalogsPerRequest:13|ms`



Metriken III - gauge

■ Gauges

- Immer nur der letzte Wert pro flush interval wird übertragen
- Wenn sich der Wert nicht ändert, wird der Wert vom vorigen flush interval geschickt
- Übergabe-String:
 - `system.cpu.whatever:263|g`
 - `system.cpu.whatever:-13|g`
 - `system.cpu.whatever:+23|g`
- Nutzen wir nicht !

Übergabe von Metriken an StatsD

- Für fast alle Programmiersprachen gibt es Bibliotheken

- Wir nutzen bash, PHP und tcl (KVK)

- Übergabe von Daten per bash an Port 8125

```
echo "foo:1|c" | nc -u -w0 statsd-server 8125
```

oder als multi metric packet

```
nc -w 1 -u statsd-server 8125 << EOF  
myMetric.kvk.anfrage:1|c  
myMetric.kvk.catalogsPerRequest:13|ms  
system.cpu.whatever:13|g  
EOF
```

- -u == UDP (non-blocking)

Beispiele aus dem KVK-Projekt

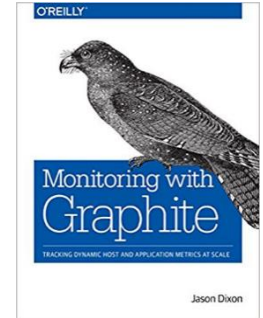
- Aufrufe im KVK-Core
- Counter
 - `kvk_stats::increment "form.digital-media.enabled"`
 - `kvk_stats::increment "form.digital-media.disabled"`
 - `kvk_stats::increment "http.tls.enabled.yes"`
 - `kvk_stats::increment "http.tls.enabled.no"`
 - `kvk_stats::increment "form.mask.$projektname"`
 - `kvk_stats::increment "form.attributes.$projektname.$attributname,,`
- Timer
 - `kvk_stats::timing "catalog.count" $catalogsPerRequest`
 - `kvk_stats::timing "catalog.mask.$projektname.count" $catalogsPerRequest`
 - `kvk_stats::timing "requestTime" $requestTime`

Graphite – das Core-System

- StatsD beliefert Graphite
- Was macht Graphite?
 - Store numeric time-series data
 - Render graphs of this data on demand
 - Aber: Grafana ist das bessere Tool für den Endnutzer
- Wer setzt Graphite ein?
 - Booking.com, GitHub, Electronic Arts (gaming)
 - Pro Sekunde erfassen manche Nutzer bis zu 1 Mio Metriken !!
- Doku
 - <http://graphite.readthedocs.io/>
 - <https://graphiteapp.org/>

Monitoring-Tool Graphite

- Buchtipp „Monitoring with Graphite“
 - Monitoring von Systemen, Diensten und Applikationen
- Time-series database (TSDB)
 - Zeitreihen werden oft in Round-Robin Datenbanken (RRD) gespeichert.
 - Pro Metrik eine Datei fester Größe mit genug Platz für Datenpunkte einer gewissen Zeitspanne
 - Daten werden zyklisch überschrieben
 - Aus Sek -> Min, Min -> Std, Std -> Tag etc. etc.
 - RRD kann keine Daten verarbeiten, wenn die Reihenfolge der Zeitstempel nicht stimmt
 - So entstand Graphite im Jahr 2008
 - **Whisper** ist die TSDB von Graphite



Graphite- Komponenten I - Carbon

■ Carbon daemons

- Carbon ist das storage backend von Graphite
- Besteht aus 3 Komponenten
 - Carbon-cache
 - accepts metrics over various protocols and writes them to disk as efficiently as possible
 - Carbon-relay
 - serves two distinct purposes: replication and sharding
 - Carbon-aggregator
 - buffer metrics over time before reporting them into *whisper*
- Muss man erst genauer kennen bei Projekten in großem Massstab

Graphite- Komponenten II - Whisper

- Whisper = time-series database
 - Pro Metrik eine Datei fixer Größe (Fixed-size database file format)
 - Beispiel: kvk.kataloge-pro-request
 - kvk ist ein Ordnername
 - kataloge-pro-request ist eine Datei
 - „Retention“ Konfiguration
 - Gibt an, wie lange Datapoints gespeichert werden bevor sie aggregiert werden
 - 10s:1w, 60s:1y
 - Nach einer Woche werden 10 Sekunden-Werte in Minuten-Werte umgewandelt und ein Jahr gespeichert
 - Je mehr Datapoints ich speichern möchte, umso größer wird eine Metrik-Datei
 - Praxis: KVK – 100 kB bis 4 MB pro Metrik !
 - Kleinster Wert muss zum „flush interval“ von StatsD passen

Graphite- Komponenten III – web application

■ Graphite Web

- Ist das User-Interface für die direkte Nutzung von Graphite
- Erlaubt die Erzeugung von Graphen (Render API)
 - Bietet dazu jede Menge Funktionen an
 - `scaleToSeconds()`
 - `sumSeries()`, `averageSeries()`, `diffSeries()`
 - etc.
 - Umfasst eine Schnittstelle für die externe Nutzung seitens anderer Analyse-Applikationen
 - Davon sehen wir daher wenig, wenn wir Grafana verwenden
- <http://graphiteapp.org/quick-start-guides/graphing-metrics.html>

Einspielen von Metriken direkt in Graphite

■ Indirekt via StatsD

- Format: `<metricname>:<value>|<type>`

■ Direkter Weg

- Graphite keine Datentypen !
- Format `<metricname> <value> <unix_epoch_time>`
- Counter
 - `echo " kvk.request 1 `date +%s`" | nc graphite-server 2003`
- Timer
 - `echo "mymetric.home.humidity 35.8 `date +%s`" | nc graphite-server 2003`

Nachträglich Logdaten in Graphite einspielen

- Konvertierung von Timestamps in unix_epoch_time Format
- Beispiel: IoT-Messdaten

Format

Datum Uhrzeit Temperatur Luftfeuchtigkeit

uwes.log

2017.08.23 17:28 29.13 35

2017.08.23 17:46 29.13 3

2017.08.23 17:48:13 29.13 3

Aus dem gesamt-Log timestamps machen

```
cat uwes.log | awk '{t=$1 " " $2 ; gsub ( "[.:]", " ",t ); print mktime(t) " " $3}' > uwes-temp.log
```

```
cat uwes.log | awk '{t=$1 " " $2 ; gsub ( "[.:]", " ",t ); print mktime(t) " " $4}' > uwes-humid.log
```

```
cat uwes-temp.log | awk '{print "echo \"test.misc.esp.temp \" $2 \" \" $1 "\"" | nc localhost 2003"}' | head -10
```

```
cat uwes-humid.log | awk '{print "echo \"test.misc.esp.humid \" $2 \" \" $1 "\"" | nc localhost 2003"}' | head -10
```

Graphite-Funktionen allgemein

- <http://graphite.readthedocs.io/en/latest/functions.html>
 - Functions are used to transform, combine, and perform computations on series data.
- Sie machen die Arbeit mit Graphite so mächtig
- Parameter sind meistens eine oder mehrere Serien von Datapoints

Graphite-Funktionen I

- Parameter können auch Wildcards („*“) beinhalten
 - `app.server-*.requests` oder auch `app.server-[1-5].requests`
- Nützliche Funktionen
 - `scale()` und `scaleToSeconds()`
 - Convert network bytes to bits per second
 - `sumSeries(metric-names-with-wildcard)` , analog `diffSeries()`
 - aggregate series
 - `summarize(time-interval)`
 - „1d“ == group results into 24-hours resolution window
 - `averageSeries()` = arithmetisches Mittel
 - `countSeries()`
 - `scale()`, `offset()`

Graphite-Funktionen II

■ Nützliche Funktionen

- `derivative()` - delta zwischen aufeinanderfolg. Datenpunkten
 - Praktisch um zu erkennen, wie sich etwas über die Zeit ändert
- `integral()`
- `sortByMaxima()` , `sortByMinima()` , `sortByName()`
- `limit(numberOfSeries)`
- `mostDeviant(n)`
 - Durchschnitt einer Serie im Vergleich zum Durchschnitt über alle
 - Angezeigt werden dann nur die n größten Abweichler

Graphite-Funktionen III

■ Nützliche Funktionen

- `removeAbovePercentile()` , `removeBelowPercentile ()`,
`removeAboveValue()` , `removeBelowValue()`
 - Elimination von Ausreißern
- `exclude(nameOfSeries-with-wildcards)`
 - Praktisch, um „seltsame“ Metriken zu herauszufiltern
 - Gegenstück ist nicht `include` sondern `grep()`
- `sumSeries()`
- `groupByNode()`
- `movingAverage(time-interval)` – Bsp „1hour“
 - Nützlich fürs „data smoothing“

Graphite-Funktionen IV

■ Nützliche Funktionen

- `timeShift(time-interval)`
 - Vergleich mit relativer Zeit aus der Vergangenheit (Bsp „4weeks“)
- `alias()` , `aliasByNode()` , `substr()`
 - Anpassung von Metrik-Bezeichnern (Labeln)
- `aliasSub(metric, pattern, substitution)`
 - verwendet Python regular expressions, backreference `\1` etc. möglich

■ Chaining bzw. Verketteten von Funktionen

- Anwenden mehrerer Funktionen nacheinander
- `f1(metric).f2().f3()....`

Fallstricke

- Arbeiten mit rate vs. Counter Werten
 - Rate-Werte sind bereits auf „pro Sekunde“ normiert, counter nicht
 - `scaleToSeconds()`
 - Takes one metric or a wildcard seriesList and returns “value per seconds”
- `summarize()` vs. `hitcount()`
 - `hitcount()` erwartet Daten normiert auf “hits per second”
 - `summarize()` erwartet dies nicht
- Am Anfang gut planen, was man möchte
 - Vieles kann nachträglich nur schlecht geändert werden
 - Retention
 - Metrikenamen
 - Sprechende Namen, Granularität

Grafana

■ <https://grafana.com/>

- “The open platform for beautiful analytics and monitoring”
- “The leading open source software for time series analytics”
- “Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored.
Create, explore, and share dashboards with your team and foster (“fördern”) a data driven culture.”
- Unterstützt jede Menge Datenquellen
 - **Graphite**
 - Elasticsearch (Grafana ist ein “Kibana”-Fork)
 - Datenbanken wie PostgreSQL etc.
- **General purpose dashboard**
- Playground <http://play.grafana.org/>
 - Let’s play around !! Hilfe zu shortcuts “h”

Cool Features

■ Doku

- <http://docs.grafana.org/>

■ Templating

- “.. allows for more interactive and dynamic dashboards.
Instead of hard-coding things like server, application and sensor name in your metric queries you can use variables in their place.”

■ Annotations / Events

- „.. provide a way to mark points on the graph with rich events”
- Bsp: Zeitpunkt, wenn ein neues Release deploy wird

■ 2 Edit-Modi

- <http://docs.grafana.org/features/datasources/graphite/#metric-editor>

■ Nested queries

- #A, #B verweist auf vorige Anfragen Bsp `averageSeries(#A, #B)`

Missing Features

- Man verliert leicht den Überblick
 - Dashboards können nur anhand des Namens gesucht werden
 - Suche nach folgenden Merkmalen ist nicht möglich
 - Überschriften von Graphen
 - In Formeln verwendeten Metriknamen oder Funktionen

Quellen im Internet

- Youtube Videos zu Grafana
 - Grafana Screencasts Episode 1 - Building Graphite Queries
 - <https://www.youtube.com/watch?v=mgcJPREl3CU&t=11s>

Ideen für Bibliotheken I

- Abschaffung von Strichlisten oder Excel-Sheets zur Zählung von Ereignissen
 - Mögliche Anwendungsfälle
 - Alarmer an den Gates
 - Beschwerden von Benutzern
 - Ausfall von Geräten
 - Angebot eines simplen Strichlisten-Dashboards als Webseite

<https://.../sendMetric/index.php?metricType=measure&metricName=uwe.temperatur.wohnzimmer&metricValue=24.5>
<https://.../sendMetric/index.php?metricType=measure&metricName=uwe.luftfeuchtigkeit.wohnzimmer&metricValue=35.6>
<https://.../sendMetric/index.php?metricType=count&metricName=uwe.strichliste.kaugummi>
<https://.../sendMetric/index.php?metricType=count&metricName=benutzung.strichliste.alarm>

Ideen für Bibliotheken II

- Unterstützung beim Changemanagement
 - Events wie „deployed“ oder „updated“ bzw. „changed“ erfassen
 - Tagging
 - Korrelation zwischen diesen Ereignissen und Performance möglich
- Bibliothek in Zahlen hilft bei
 - Erkennung von Schieflagen
 - Erkennung von Veränderungen
 - Einschätzung von Nutzerverhalten
 - Ersetzt Bauchgefühl („das passiert ganz oft“ !?)

Aufbau der Metrik-Umgebung

■ Installation der drei Komponenten

- auf dediziertem Server
- als drei Docker Container
 - KVK
 - Verwendete Images von <https://hub.docker.com>
 - dockerana/statsd
 - sitespeedio/graphite:1.0.2-2-b
 - Alternative: graphiteapp/graphite-statsd
 - grafana/grafana:latest
 - Es gibt besser gepflegte Images
 - Evtl. dockerfile selbst erstellen



Fazit

- Einarbeitung lohnt sich
- Auswertungen ohne IT möglich
 - Die IT liefert „nur“ die Metrikdaten
 - Endnutzer werten selber aus
- Installationsaufwand dank Docker moderat
 - Aber Vorsicht: neue Technologie erfordert Einarbeitung

Vielen Dank für Ihre Aufmerksamkeit

Fragen ?

uwe.dierolf@kit.edu