CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Observability

(2) Instrumentierung zur Metrikerhebung

PROF. DR.
NANE KRATZKE

creative commons

CLOUD NATIVE COMPUTING

Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.

KAPITEL 13

Beobachtbare Architekturen

13.1 Konsolidierung von Telemetriedaten

13.2 Instrumentierung von Systemen

- Logging
- Monitoring
- Tracing

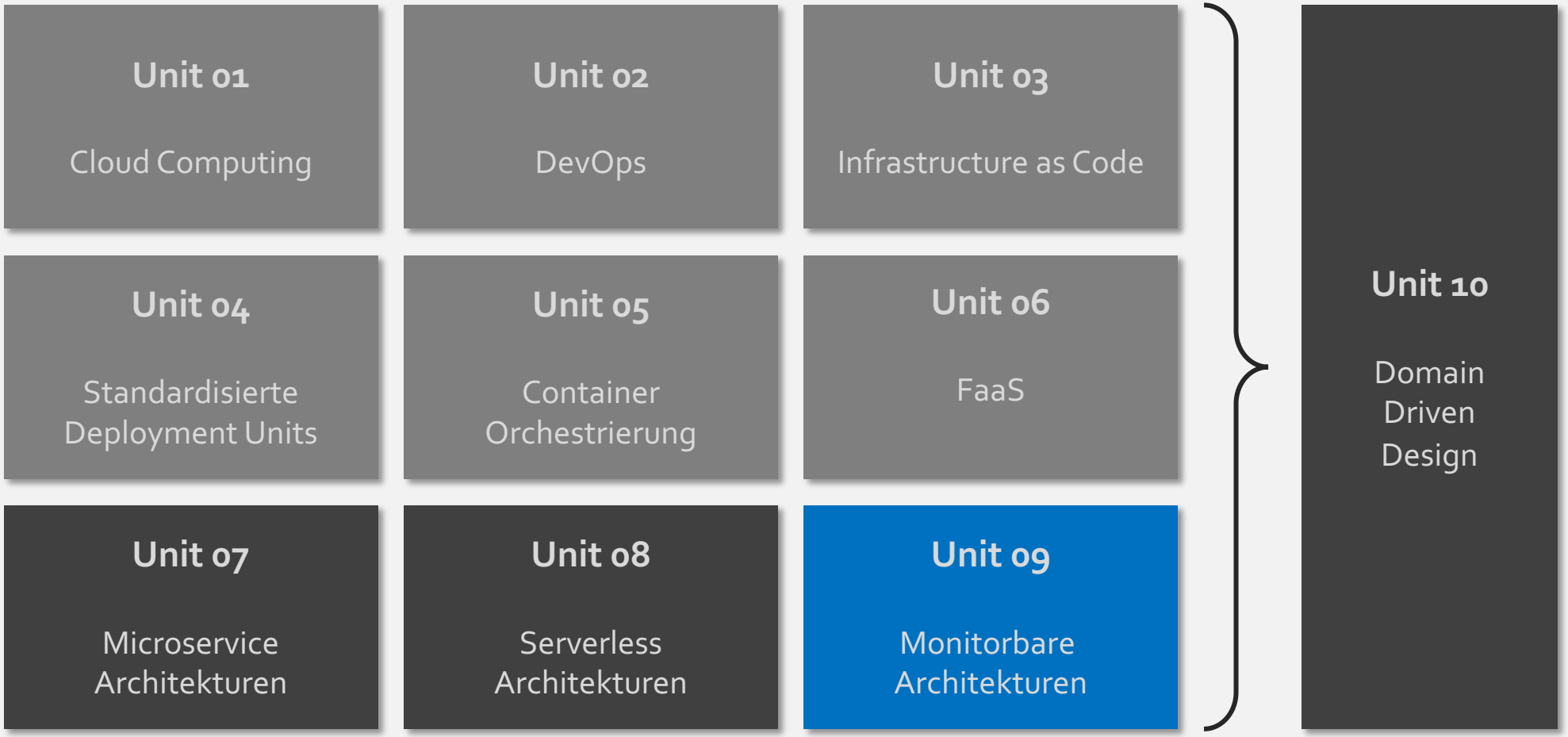
13.3 Automatisierte Instrumentierung

- Service Meshs
- Traffic-Management
- Resilienz
- Sicherheit
- Management und Analyse von Verkehrstopologien

13.4 Zusammenfassung

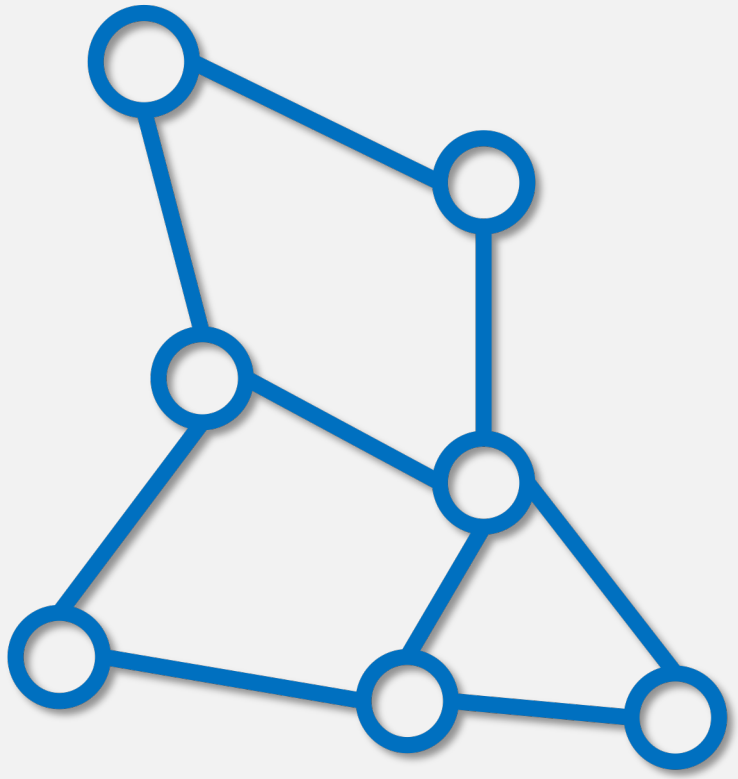
INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



AUSBLICK

- Beobachtbarkeit von Systemen
- Metriken
- Logs
- Tracing



ÜBERWACHUNG VON SERVICE-ARCHITEKTUREN

Logging, Monitoring, Tracing

Die Überwachung von Software erfolgt üblicherweise mittels der Erfassung von drei Arten von Telemetriedaten:

- Metriken im Rahmen eines **Monitorings** liefern **quantitative** Informationen zu Prozessen, die im System ausgeführt werden.
- Mittels **Logging** (Protokollierung) lässt sich **qualitativer** Einblick in anwendungsspezifische Ereignisse gewinnen, die von Prozessen verarbeitet werden.
- **Distributed Tracing** (verteilte Ablaufverfolgung) ermöglicht Einblick in den gesamten Lebenszyklus von Requests entlang von Systemkomponenten (Services), um so bspw. Fehler und Latenzen in der verteilten Verarbeitung zu erkennen.

METRICS

Instrumentierung am Beispiel von Prometheus

```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())
```

Instrumentierung einer Methode

Metrik erzeugen

Start des Exporters

```
pip install prometheus-client
```

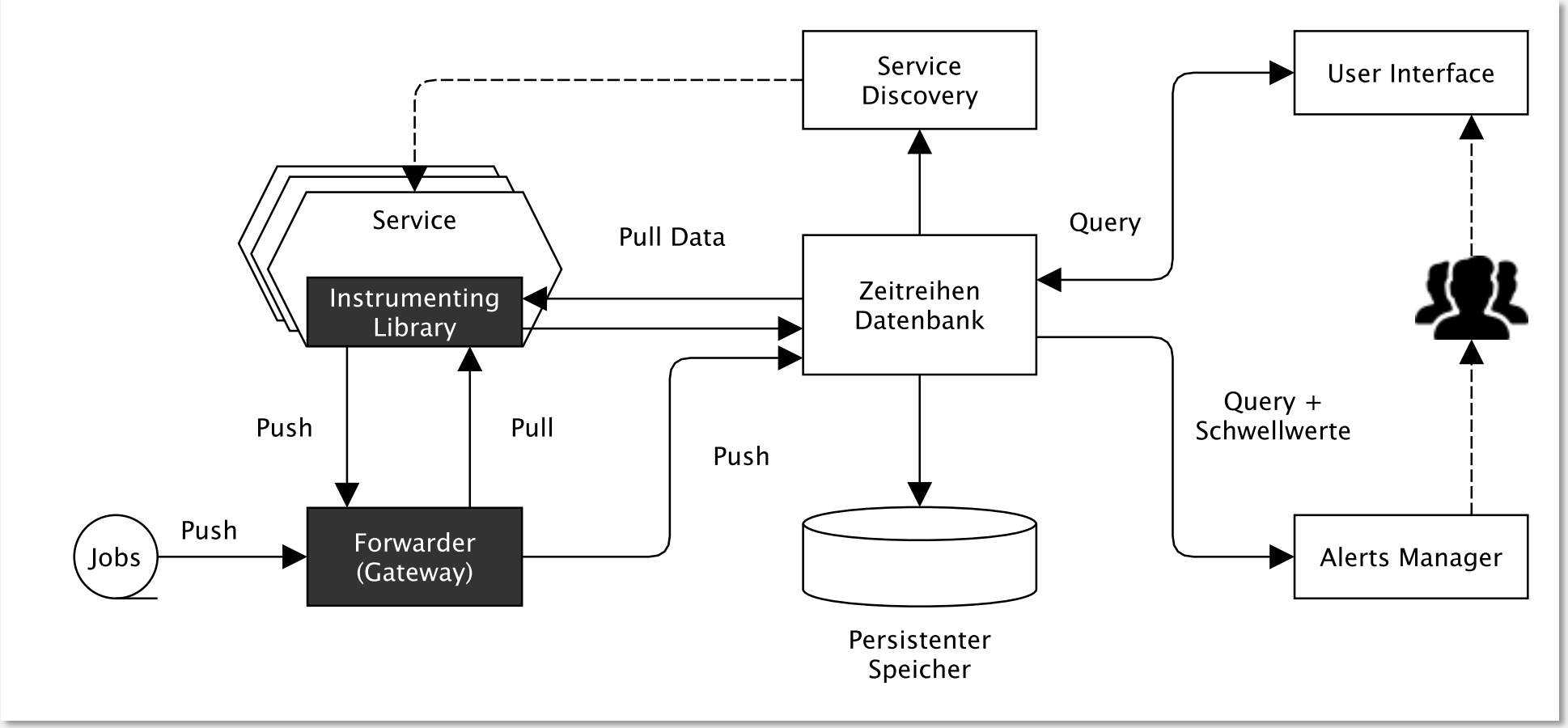
Die aktualisierten Metriken werden dann auf Port 8ooo über HTTP bereitgestellt und können von Prometheus abgefragt werden.

Mittels des folgenden Kommandos können Sie das auch selber in der Konsole nachvollziehen:

```
watch curl -s http://localhost:8000
```

OBSERVABILITY ARCHITEKUR

Telemetriedaten-Konsolidierung



METRICS

Counter am Beispiel von Prometheus

```python
from prometheus_client import Counter

# Counters go up, and reset when the process restarts
c = Counter('my_failures', 'Description of counter')
c.inc()     # Increment by 1
c.inc(1.6)  # Increment by given value

# There are utilities to count exceptions raised

@c.count_exceptions()
def f():
  pass

with c.count_exceptions():
  pass

# Count only one type of exception
with c.count_exceptions(ValueError):
  pass
```

Zähler (Counter)

Ein Zähler ist eine kumulative Metrik, die einen einzelnen monoton ansteigenden Zähler darstellt, dessen Wert nur erhöht oder beim Neustart auf Null zurückgesetzt werden kann. Zähler sind geeignet, um die Anzahl eingegangener Requests, erledigter Aufgaben oder aufgetretenen Fehler zu überwachen.

Instrumentierungsbeispiele, um Zählungen vorzunehmen.

METRICS

Messungen am Beispiel von Prometheus

CLOUD NATIVE COMPUTING

Messung (Gauge)

Unter einer Messung versteht man eine Metrik, die einen einzelnen numerischen Wert darstellt, der willkürlich auf und ab gehen kann. Messung werden normalerweise für Messwerte wie Temperaturen oder die aktuelle Speichernutzung verwendet, aber auch für "Zählungen", die steigen und fallen können. Bspw. die Anzahl gleichzeitiger Requests.

Instrumentierungsbeispiele, um Messungen vorzunehmen.

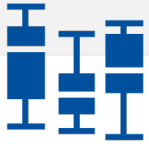
```python
# Gauges can go up and down.

from prometheus_client import Gauge
g = Gauge('my_inprogress_requests', 'Description of gauge')
g.inc()      # Increment by 1
g.dec(10)    # Decrement by given value
g.set(4.2)   # Set to a given value

# There are utilities for common use cases
g.set_to_current_time()   # Set to current unixtime

# Increment when entered, decrement when exited.
@g.track_inprogress()
def f():
  pass

with g.track_inprogress():
  pass

# A Gauge can also take its value from a callback
d = Gauge('data_objects', 'Number of objects')
my_dict = {}
d.set_function(lambda: len(my_dict))
```

METRICS

Histogramme am Beispiel von Prometheus

Histogramm

Ein Histogramm erfasst Beobachtungen (oft Dinge wie Dauern oder Größen) und zählt sie in konfigurierbaren Bereichen. Histogramme stellen die Verteilung von Beobachtungen ausführlicher dar. Häufig reichen auch einfache Quantilen, um Verteilungen einschätzen zu können (=> Summary).



Zusammenfassung (Summary)

Ähnlich wie bei einem Histogramm werden in einer Zusammenfassung Beobachtungen erfasst. Zusammenfassungen, dienen allerdings zur Berechnung konfigurierbarer Quantile über ein gleitendes Zeitfenster und damit für einfache statistische Auswertungen von Beobachtungsverteilungen.

```python
# Histograms track the size and number of events in buckets.
# This allows for aggregatable calculation of quantiles.

from prometheus_client import Histogram

h = Histogram('request_latency_seconds', 'Description of histogram')
h.observe(4.7)    # Observe 4.7 (seconds in this case)

# There are utilities for timing code

@h.time()
def f():
  pass

with h.time():
  pass
```

CLOUD NATIVE COMPUTING

METRICS

Best Practices für die Instrumentierung

- Die Instrumentierung sollte ein integraler Bestandteil Ihres Codes sein.
- In Cloud-nativen Systemen unterscheidet man dabei üblicherweise Online-Systeme und Batch-Systeme.
- Requests sollten einheitlich instrumentiert werden. Dabei ist es empfehlenswert Requests am Ende Ihrer Bearbeitung zu zählen, da dies dann mit den Fehler- und Latenzstatistiken übereinstimmt und in der Regel einfacher zu codieren ist.

Ein Online-System ist ein System, bei dem ein Nutzer oder ein anderes System eine sofortige Reaktion erwartet. Die meisten Datenbank- und Web-Services fallen in diese Kategorie.

Typische **Schlüsselmetriken eines Online-Systems** sind

- Anzahl der beantworteten Requests
- Anzahl aktuell laufender Requests
- Anzahl aufgetretener Fehler
- Latenz

Batch-Systeme zeichnen sich dadurch aus, dass sie nicht kontinuierlich ausgeführt werden, was für die Metrikerhebung meist ein Push Gateway erforderlich macht.

Typische **Schlüsselmetriken eines Batch-Jobs** sind

- der Zeitpunkt, der letzten erfolgreichen Ausführung
- Dauer dar Teilschritte eines Jobs
- Gesamtlaufzeit eines Jobs
- Zeitpunkt des letzten Abschluss eines Jobs (erfolgreich oder fehlgeschlagen)
- Gesamtzahl verarbeiteter Datensätze

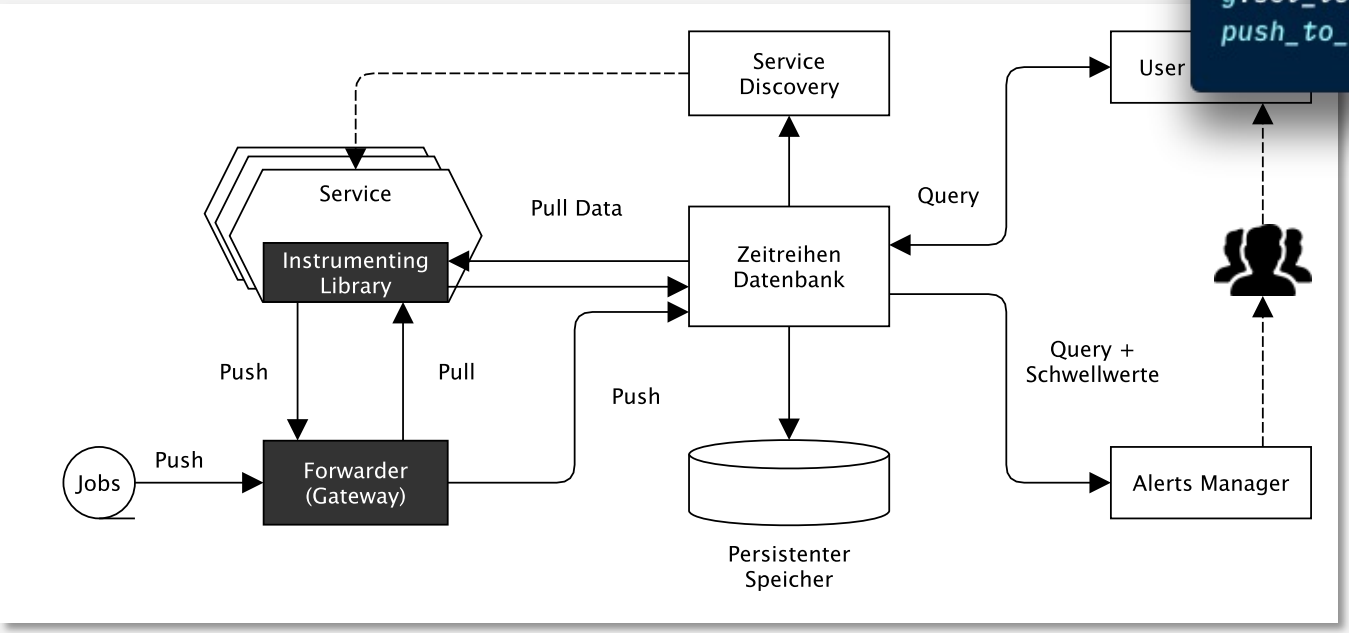
METRICS

Best Practices für die Instrumentierung von Batch-Jobs (Push Gateway)

Da Batch-Jobs keinen Always-On Charakter haben, können diese nicht periodisch von Monitoring-Systemen abgefragt werden.

Über sogenannte Push Gateways können jedoch auch kurzlebige Prozesse und Batch-Jobs, Metriken einem Monitoring-Systemen zugänglich machen.

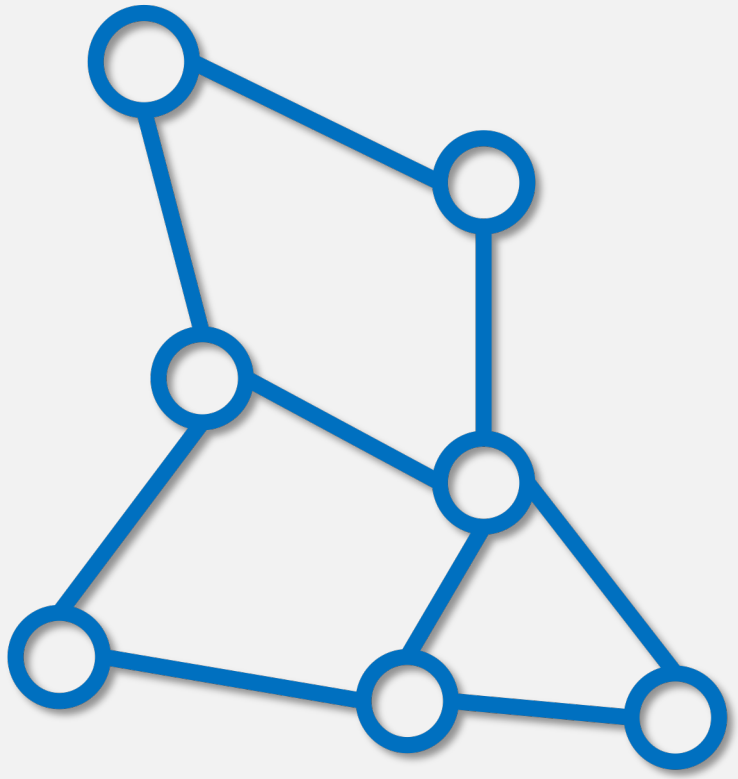
```python
# The Pushgateway allows ephemeral and batch jobs to expose
# their metrics to Prometheus

from prometheus_client import Gauge, push_to_gateway

g = Gauge('job_last_success_unixtime',
          'Last time a batch job successfully finished',
          )
g.set_to_current_time()
push_to_gateway('localhost:9091', job='batchA')
```



Instrumentierung eines Batch-Jobs am Beispiel von Python und Prometheus.

AUSBLICK

- Beobachtbarkeit von Systemen
- Metriken
- Logs
- Tracing



KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de