JAVACREAM

Training
Consulting
Projectmanagement

Prometheus und Grafana

Vorstellungsrunde

JAVACREAM
Training
Consulting
Projectmanagement

- Name
- Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Aktuelle Problemsituation
- Individuelle Zielsetzung

Ausgangssituation

Bereitstellung von Anwendungen

- Nach bestem Wissen und Gewissen getestet und damit fachlich fehlerfrei
  - Debugging, Tracing, Profiling ist damit bereits “erledigt”
- Die Orchestrierung einer Anwendung wird von “der Cloud” übernommen
  - Skalierung und Failover ist nicht Aufgabe von Prometheus
- Logging schreibt Informationen mit zusätzlichem Context als Freitext
  - Prometheus schreibt und analysiert keine Logs
- Jede Anwendung erzeugt Metriken = Meßwerte = Zahl mit einer Einheit = “eine physikalische Information”
  - Prometheus erfasst diese Metriken historisch und bietet Analysemethoden für diese an

Details zu Metriken

- Zahlenwert
- Einheit
- Ein eindeutiger Name
  - Damit wird ein “Minimal-Kontext” definiert
- Beispiel
  - Diese Metrik beschreibt die Menge der geschriebenen Dateien
    - 128
    - Bytes
    - Geschriebenes Datenvolumen
  - Diese Metrik beschreibt die Menge der geschriebenen Dateien in das Verzeichnis ‘Hugo’
    - 128
    - Bytes
    - Geschriebenes Datenvolumen {dir=’Hugo’}
  - Eine Metrik besteht aus mehreren Zahlenwert-Einheit-Paaren, der Name der Metrik gruppiert damit
    - Geschriebenes Datenvolumen
      - Anzahl der Schreibvorgänge (42)
      - Summe der geschriebenen Daten (4711 Megabyte)

Prometheus Architektur

Bestandteile: Teil 1

- Storage = Datenbank
  - Time Series Database
    - “Datenbank-Kategorie aus dem NoSql-Umfeld”
- Auswertung der Daten wird durch einen Query-Interpreter ermöglicht
  - Die Sprache PromQL ist natürlich ausgerichtet auf die Analyse von Metriken
    - ~~select * from table1 where join~~
    - rate (number_of_requests[1m)
      - Mittelung mit Änderung / Steigung
- Scheduler, der Anhand einer Scraping-Konfiguration die Daten von einem Endpunkt / einer URL abholt
  - Eine Anwendung muss auf Anfrage diese Daten bereitstellen
    - Die Anwendung ist NICHT VERANTWORTLICH, die Daten historisch bereitzustellen

Beispiel

- Der Prometheus-Server stellt bereits Metriken zur Verfügung
  - http://host:port/metrics

Prometheus und Metriken

- Der “Meßwert” in Prometheus ist immer eine Zahl
- Die Einheit ist in Prometheus im Namen der Metrik untergebracht
  - z.B. _seconds, _bytes
  - Best Practice: Grundeinheiten sind zu benutzen, also z.B. Bytes, nicht Kilobytes
- Metrik-Typen
  - Counter
    - diese werden fortlaufend hochgezählt
  - Gauge
    - freier Wert
  - Summary und Histogram gruppert mehrere Metriken (Children) unter einem Metrik-Namen

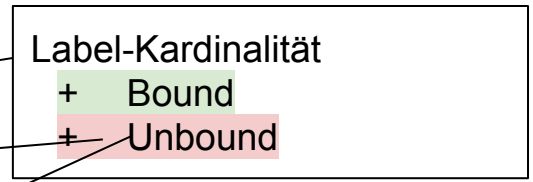
Interpretation von Countern ist stabiler als der Wert von Gauges

Bestandteile: Teil 2

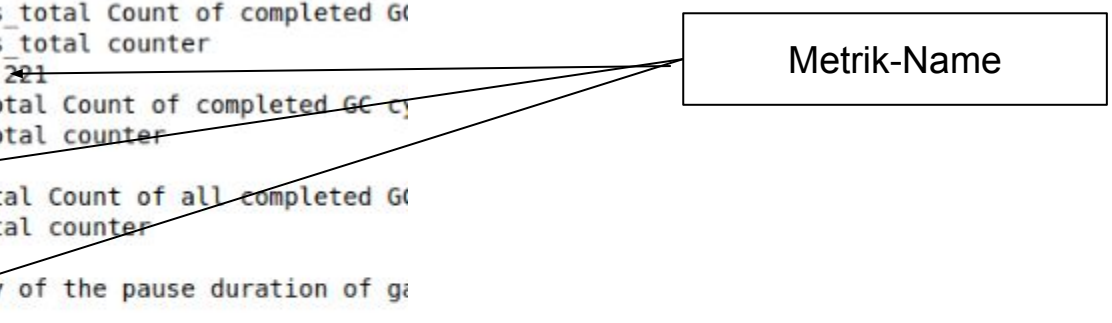
- Web-Anwendung
  - Basis-Administration

Labels in Prometheus

- Jeder Metrik kann ein Dictionary (key-value-Paare) von Labels zugeordnet werden
- Labels werden in PromQL-Abfragen als Kriterien benutzt
- Hinweis
  - Exzessives Nutzen von Labeln ist eine Bad Practice
- Beispiel
  - cpu -> CPU-Auslastung
    - cpu {instance='localhost', job='ubuntu'} -> Die CPU-Auslastung
  - Http-Zugriffe
    - request_total
    - request_total{method='Get'} OK
    - request_total{page=index.html} <- zu fein!
    - request_total{exception_message='NullpointerException@f1'}



```
# HELP go_gc_cycles_automatic_gc_cycles_total Count of completed G(
# TYPE go_gc_cycles_automatic_gc_cycles_total counter
go_gc_cycles_automatic_gc_cycles_total 221
# HELP go_gc_cycles_forced_gc_cycles_total Count of completed GC cy
# TYPE go_gc_cycles_forced_gc_cycles_total counter
go_gc_cycles_forced_gc_cycles_total 0
# HELP go_gc_cycles_total_gc_cycles_total Count of all completed G(
# TYPE go_gc_cycles_total_gc_cycles_total counter
go_gc_cycles_total_gc_cycles_total 221
# HELP go_gc_duration_seconds A summary of the pause duration of ga
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 5.2302e-05
go_gc_duration_seconds{quantile="0.25"} 0.000216142
go_gc_duration_seconds{quantile="0.5"} 0.000367828
go_gc_duration_seconds{quantile="0.75"} 0.000635094
go_gc_duration_seconds{quantile="1"} 0.018348493
go_gc_duration_seconds_sum 0.121415755
go_gc_duration_seconds_count 221
```



Zu den Prometheus-Metriken

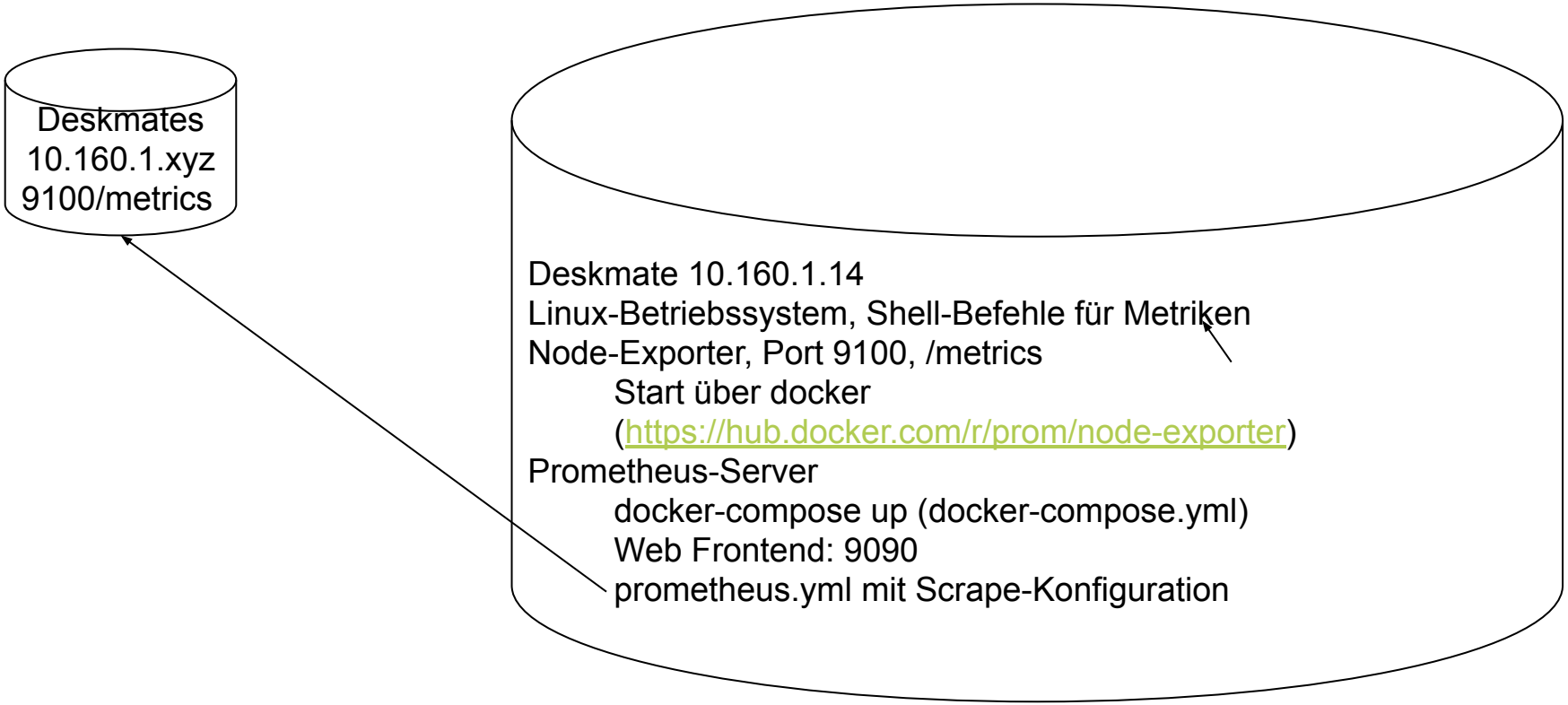
- Format ist proprietär, wurde festgelegt von der Prometheus-Community
  - Kein Bedarf für JSON, XML, …
  - Kein Vorteil, ein Binärformat zu nutzen
- Das Erzeugen der Seite ist
  - Komplett Bestandteil der Anwendung
    - Prometheus-Libraries erzeugen Einträge in diesen Seiten als Einzeiler im Programm
    - Jede Anwendung hat damit einen integrierten Webserver, /metrics
  - Eine Anwendung kann bereits ohne Bezug zu Prometheus Metriken bereitstellen
    - Beispiel: Betriebssystem
    - Lösung: Ein ~~Adapter~~ Exporter kann natürlich jederzeit aus diesen Metriken eine Prometheus-konforme Metrik-Seite erstellen

Beispiel: Node-Exporter

- Node = Host-Maschine mit Linux-Betriebssystem
  - Standalone
  - Port: 9100.
  - Gestartet direkt auf dem zu überwachenden Linux-Host
- Oder: Bestandteil der Applikation
  - z.B. der Java JMX-Exporter
  - Java-Agent, der beim Aufruf eines beliebigen Java-Programms mit angegeben wird
- Oder: Eingebundene Library

Trainings-Umgebung

- Bisher
  - Zentraler Prometheus-Server unter 10.160.1.14
- Nun
  - Jeder Teilnehmende soll einen eigenen Prometheus starten


Deskmates
10.160.1.xyz
9100/metrics
Deskmate 10.160.1.14
Linux-Betriebssystem, Shell-Befehle für Metriken
Node-Exporter, Port 9100, /metrics
Start über docker
(https://hub.docker.com/r/prom/node-exporter)
Prometheus-Server
docker-compose up (docker-compose.yml)
Web Frontend: 9090
prometheus.yml mit Scrape-Konfiguration

PromQL

Eine Abfragesprache für ein TSDB

- Eine Abfrage nach einem Metrik-Namen liefert alle passenden Metriken
  - metric_name
  - Analogie zu SQL “select * from metrics_table where name = metric_name”
- Zur Selektion werden die Labels benutzt
  - metric_name{label1=value1}
  - Analogie zu SQL “select * from metrics_table where name = metric_name and label1 = value1”
-

Anwender von Prometheus / Grafana operieren direkt nie auf Instant Vectors

Eine Abfragesprache für ein TSDB

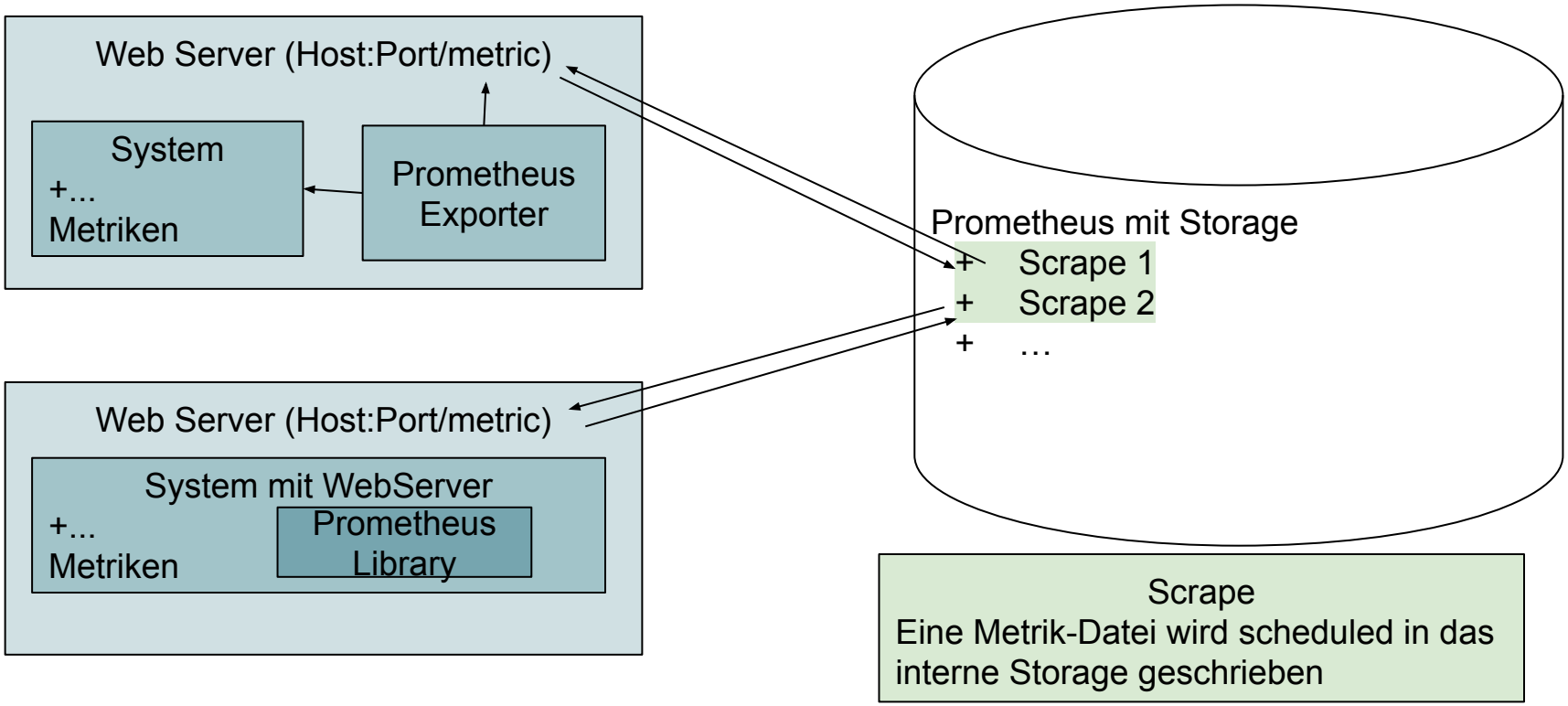
Anwender von Prometheus / Grafana operieren direkt nie auf Instant Vectors

- Instant Vector
  - Bereich Messwerte + Zeitstempel
-
- Range Vector
  - Die Menge von Metrik-Werten innerhalb des angegebenen Zeitraums
    - Wert-Timestamp-Paare
  - Angabe des Zeitraums durch [1m]

Prometheus pragmatisch

- In den meisten Fällen sind die Absolutwerte von Metriken eher uninteressant
  - process_cpu_seconds_total
- Aussagekräftig sind nur Mittelwerte
  - gemittelt über einen Bereich von Messwerten
- Häufig ist die Änderung pro Zeitintervall das wichtigste
  - Dafür werden in den meisten Fällen Counter-Metriken benutzt
    - Analyse mit rate(metric[mittelung])
    - Für Counter kann Prometheus einen “reset”, einen “restart” trivial komensieren

Stand von gestern: Architektur



Stand von gestern: PromQL

- Lesender Zugriff auf die im Storage vorhandenen Daten
- Eine Metrik hat einen Namen, Einheit, Wert,
- Eine Time Series ist eine mit Zeitstempel versehene Sequenz von Metrik-Daten
  - Ein einzelner Wert einer Time Series ist der **Wert** der Metrik zu einem bestimmten Zeitpunkt
  - In den meisten Fällen ist die Angabe einer Metrik eine Selektion mit mehreren Treffern
    - Diese Treffer werden ebenfalls in einem Vector = Liste gehalten
      - “Instance Vector”
- Ein Bereich einer Time Series (“Werte von bis”) ist ein **Vektor** (eine Liste) von einzelnen Werten
  - “Range Vector” besteht aus Vektoren von Time Series

ToDo

- Was bedeutet z.B. die Angabe process_cpu_seconds_total? Was ist das Ergebnis?
- Wie kann ich weiter selektieren und damit die Treffermenge einschränken?
- Was macht der offset?
- Wie kann ich einen Range Vector erzeugen?

Aufbereitung der Daten, Mathematik

- node_filesystem_size_bytes
- +, -, *, /

Aufbereitung der Daten, Aggregation

- node_filesystem_size_bytes
- Gruppierung
  - without / by
- Operationen
  - sum
  - avg
  - min, max
  - count
  - stddev, sdtvar
  - topk, bottomk

Sinnvoll für Gages, in der Regel nicht für Counter-Metriken!
+ syntaktisch aber möglich…

Counter-Metriken

- Counter sind bei Metriken sehr beliebt, weil ein reset z.B. durch einen Neustart eines Systems die Metrik-Auswertung nicht beeinflusst
- Beispiel
  - Gauge 2,4,5,9,3,4
  - Counter 2,4,5,9,3,4 -> Prometheus: 2,4,5,9,12,13
- Die Analyse untersucht die Dynamik der Counter, nicht den Absolutwert
  - Grundlage ist immer ein Range Vector
  - Operation
    - rate
    - increase
    - irate
    - resets

Wechselnden Gauge-Werte

- predict_linear
  - Parameter ist ein Range-Vector
  - Rückgabe ist der Änderung pro Sekunde, für einen Wert in z.B. 1 min: * 60
- delta / idelta
- changes
  - Anzahl der Änderungen des Wertes einer Gauge

Dies und Das

- Sortieren von Werten
  - sort(node_filesystem_size_bytes)
- Ersetzen von Labels
  - label_replace(metric, old_label, new_label)
- label_join

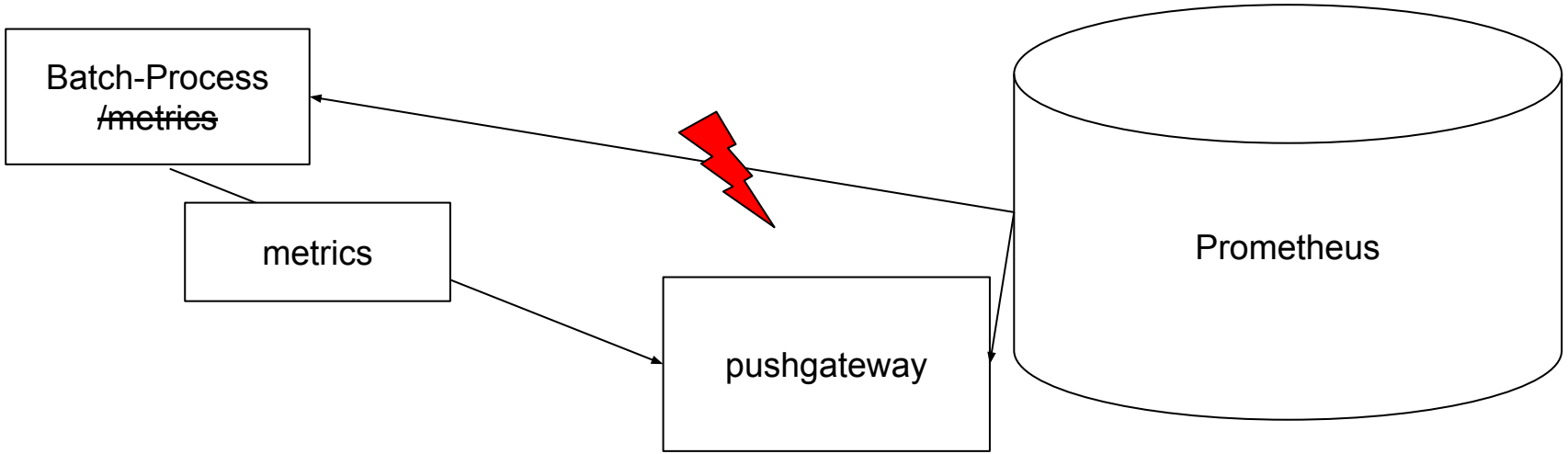
  Fügt zwei Labels zu einem neuen zusammen
- Referenz
    - https://prometheus.io/docs/prometheus/latest/querying/functions/

Prometheus-Ökosystem

Alert-Manager

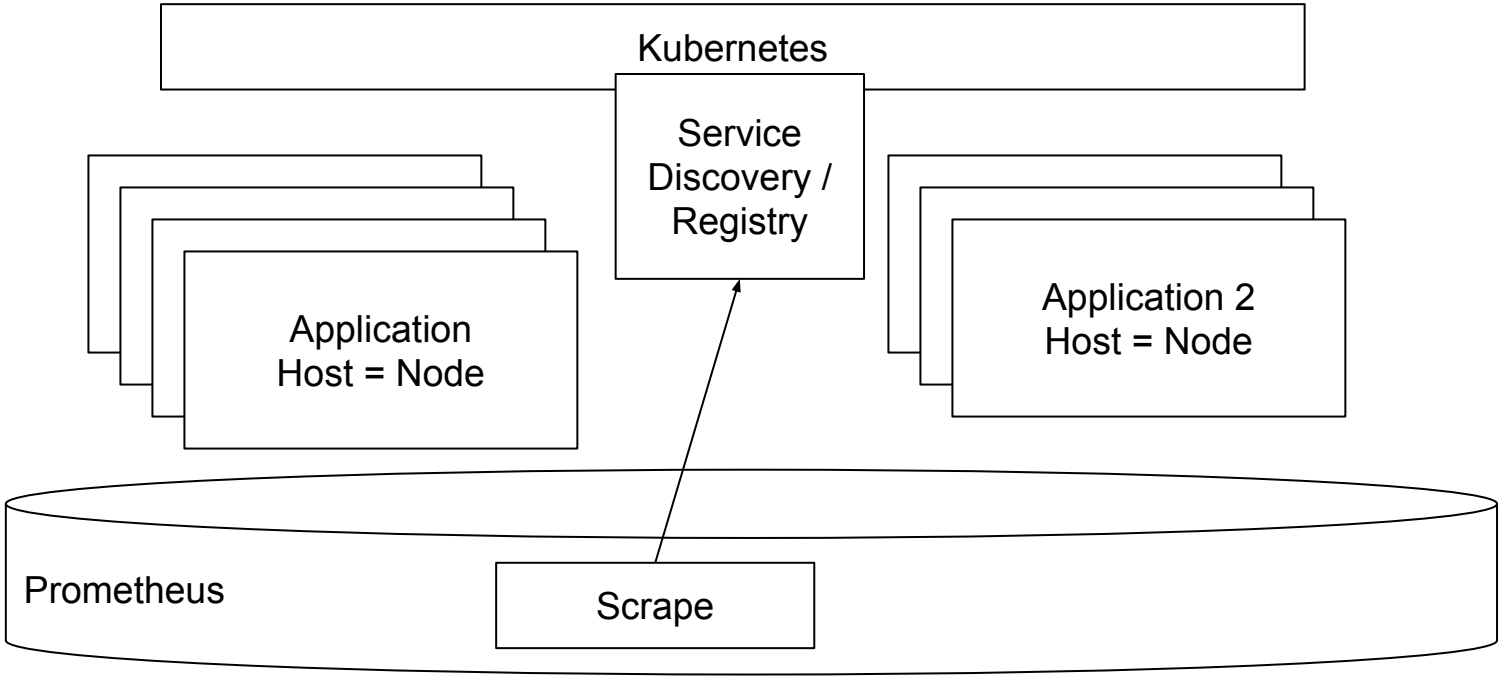
- Separates Prometheus-Produkt
  - Nutzbar übrigens auch für mehrere Prometheus-Instanzen
  -
  - rules.yml
    - Enthält “rules”
      - z.B. kann eine rule zu bestimmten Metriken Labels hinzufügen
      - Metriken beim Scrape erzeugen
        - durch Angabe eines PromQL-Ausdrucks
    - Ein Alert ist ebenfalls eine Regel
      - - alert AlertName
      - expr job:up:avg{job=’node’} < 0.8

Pushgateway



Prometheus und Orchestrierung / Kubernetes

- “Prometheus und die Cloud”
-



Prometheus und Java-Applikationen

Allgemein

- Die Java Virtual Machine ist nicht Prometheus-kompatibel
- Die JVM erfasst permanent bereits einen sehr reichhaltigen Satz von Metrik-Informationen
  - Durchschleifen von Betriebssystem-Metriken
  - Java-typische Metriken
    - Speicherorganisation
    - Garbage Collection
- Zugriff auf die Metriken der JVM erfolgt mit der Java Management Extension, JMX
  - dafür existieren eigene Client-Applikationen

Exporter

- Jede Java-Anwendung kann die JMX-Metriken via JMX Exporter als Javaagent Prometheus-kompatibel anbieten
- Prometheus-Java-Libraries können in die Applikation übernommen werden
  - Beispiel
    - Spring Boot Application
    - 10.160.1.14:8080/swagger-ui.html