Monitoring mit Prometheus

GPN15

Michael Stapelberg
<michael@robustirc.net>

Agenda

- Hintergrund
- Anfang bis Ende-Beispiel (Server in Go)
- Datentypen, Labels, Rules, Alerts
- Third-party-software, Exporter
- Rules (Beispiele)
- Alerting-Philosophie
- Dashboards

Hintergrund

- Borg mit Borgmon, Prometheus ist sehr ähnlich
- Geschrieben von Matt Proud und Julius Volz, hauptsächlich entwickelt bei SoundCloud
- IMO großartige Lösung für Monitoring/Alerting

Ursprüngliches Szenario

- Viele HTTP-Server, z.B. DCS oder RobustIRC
- Zustand des Systems aggregieren und speichern
  → exportierte Counter verfügbar machen
- Ausfälle debuggen, Alerting bei Problemen

Beispiel: HTTP-Server

```
package main

import "fmt"
import "log"
import "net/http"

func main() {
  http.HandleFunc("/search",
    func(w http.ResponseWriter, r *http.Request) {
      fmt.Fprintf(w, "OHAI")
    })
  log.Fatal(http.ListenAndServe(":8080", nil))
}
```

```go
import "github.com/prometheus/client_golang/prometheus"

var queries = prometheus.NewCounter(prometheus.CounterOpts{
    Name: "queries",
    Help: "Queries for /search",
})

func init() {
    prometheus.MustRegister(queries)
}

func main() {
  http.Handle("/metrics", prometheus.Handler())
  http.HandleFunc("/search",
    func(w http.ResponseWriter, r *http.Request) {
      fmt.Fprintf(w, "OHAI")
      queries.Inc()
    })
  log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Demo

- Prometheus starten
- Targets einrichten (localhost:8080/metrics)
- Metrik interaktiv abfragen

Prometheus in a nutshell

- Sammelt Metriken, speichert Timeseries
- Interaktiv: Dashboards und Queries
- Alerting rules → Alertmanager → z.B. Pushover
- Stabil, Standalone (keine Dependencies)

Datentypen

- Counter: nur steigend, z.B. Requests
  Overflows und Restarts werden gehandled
- Gauge: beliebiger Wert, z.B. #Goroutine
- Histogram: aggregierbar, feste Buckets
- Summary: beliebige Quantiles berechnen (keine festen Buckets), aber nicht aggregierbar

Labels

- Aus einer Metrik viele gleichartige machen

```
var queries = prometheus.NewCounterVec(
    prometheus.CounterOpts{
        Name:        "queries",
        Help:        "Queries for /search, by response code",
    },
    []string{"code"},
)

queries.WithLabelValues("200").Inc()
```

Labels (2)

- Fest konfiguriert für bestimmte targets
  → aggregieren z.B. nach staging, production

Rules

- Automatische Queries
  → Vorberechnen (schnellere Dashboards)
  → Modulare rule files

- z.B. persistierte Nachrichten über alle Nodes:
  `job:committed:rate5m_sum = sum(rate(commit_count[5m])) by (job)`

Alerts

```
ALERT CapacityNotNPlusOne
  IF (count(up{job="robustirc"} == 1) < 3)
  FOR 30m
  WITH {
    job="robustirc"
  }
  SUMMARY "Capacity less than n+1"
  DESCRIPTION "Only {{$value}} of 3 nodes are
up. Please replace the faulty nodes."
```

Demo

- Alert einrichten

Third-party software

- node_exporter (disk, memory, …)
- statsd_bridge (liest StatsD, exportiert Prom.)
- Exporter für Redis, MySQL, JVM, …

Rules: Beispiel: Leader-Wechsel

```
job:leader_flaps_stable:sum_deriv10m =
(sum(
  abs(
   deriv(raft_isleader{job="robustirc"}[10m])
  )
 ) by (job)
) *
(count_scalar(
  (time() - process_start_time{job="robustirc"})
  > (5*60)
 ) >= 3)
```

Rules: Beispiel: Verfügbarkeit

```
job_instance:availability:sum_rate =
  sum(
    rate(
seconds_in_state{state=~"Leader|Follower"}[1m])
  ) by (job, instance)
/
  sum(
    rate(seconds_in_state[1m])
  ) by (job, instance)
```

Alerting-Philosophie

- Lese-Tipp: „My Philosophy on Alerting“
- Symptome, nicht Ursachen
- Actionable, nicht automatisierbar

Demo: Dashboards

- console templates
- PromDash
- Grafana

Fragen?

- http://prometheus.io/
- Fragen?
- Bitte gebt mir Feedback zum Vortrag!
  http://goo.gl/forms/HSC1dchPFn

Service Level Agreements: Motivation

- Stabilität eines Dienstes messen
- abwägen zwischen Feature-Launches oder Stabilitätsverbesserungen
- kommuniziert/vereinbart oder auch nicht

SLA (Service Level Agreement)

- Verfügbarkeit pro Monat/Quartal/Jahr?
  99%: 3,65 Tage down/Jahr
  99,9%: 8,76 Stunden down/Jahr
  99,99%: 52,56 Minuten down/Jahr

- Möglichst nahe am Nutzer messen

- Latenz oft wichtig, z.B.
  < 100ms Latenz für 99.9% der Anfragen

SLA: Implementation

- Faustregel:
  1% der erlaubten Fehler in letzten 60m → Alert

- Katastrophales Problem: 1% schnell erreicht
  Anhaltendes Problem: 1% letztendlich erreicht

- Einfach implementiert:
  `rate(queries-over-100ms[60m]) > 1547`