Open Source monitoring met Prometheus

Ed Schouten <ed@kumina.nl>

Over de spreker

Ed Schouten

- 2003-*: Gebruiker/hacker van Open Source Software.
  - 2008-*: ontwikkelaar bij FreeBSD en LLVM.
  - 2014-*: auteur van CloudABI: eenvoudige sandboxing.
- 2011-2012: Tijdens studie werkzaam bij Kumina.
- 2012-2014: Werkzaam bij Google.
  - Als Software Engineer/Site Reliability Engineer in München.
- 2016-*: Full-time werkzaam bij Kumina.

De vragen voor vandaag

- Wat is Prometheus?
- Hoe werkt Prometheus?
- Hoe krijg je data in Prometheus?
- Hoe krijg je data uit Prometheus?
  - Hoe maak je dashboards?
  - Hoe doe je alerting?
- (Hoe gebruiken wij bij Kumina Prometheus?)

De geschiedenis van Prometheus

- ~2003: Google ontwikkelt Borg: een cluster-manager.
  - Bestaande monitoringsystemen lijken hier niet geschikt voor.
    - Slechte ondersteuning voor dynamische targets.
    - Weinig inzicht in globale staat.
  - Borgmon: **Borg**'s **mon**itoringsysteem.
- 2012: Ex-Googlers bij SoundCloud missen Borgmon.
  - Prometheus: een 'herimplementatie' van Borgmon.
  - Open Source: Apache-2.0-licentie.
  - Iets schoner/algemener dan Borgmon.
- (2014: Kubernetes: een Open Source Borg)

Prometheus in vogelvlucht

- Prometheus is algemeen toepasbaar.
  - Geen Nagios-stijl networks-hosts-services-hiërarchie.
  - Niet alleen om servers in de gaten te houden.
    - Temperatuur in je huis.
    - Aandelen op de beurs.
- Prometheus is veelzijdig.
  - Nagios kan alleen maar alerten.
  - Munin kan alleen maar graphen.
  - Prometheus kan dezelfde data gebruiken voor allebei.
  - Alerts en graphs zijn niet voorgedefinieerd.

Prometheus’ datamodel

- Prometheus is een 2D-database.
  - Horizontale as: tijd.
    - Vaak een eindige retentieperiode.
  - Verticale as: unieke identifier voor iedere metric.
    - Identifier: een dictionary van string keys en string values.
      `{__name__="http_requests_total",job="nginx",instance="kumina.nl"}`
    - De waarde van `__name__` mag je ook voor de `{}` zetten.
      `http_requests_total{job="nginx",instance="kumina.nl"}`
  - In de cellen: 64-bits floating-point-waarden.
  - On-disk: iedere metric in een eigen gecomprimeerde file.
  - PromQL: query-taal voor opvragen van data uit de 2D-database.

Primitieve datatypen

- Gauges (meters):
  - Voorbeelden: geheugengebruik, temperatuur, wisselkoers.
  - Kunnen stijgen en dalen.
- Counters (tellers):
  - Voorbeelden: aantal requests, aantal klanten door de voordeur.
  - Kunnen alleen maar stijgen.
  - Resets bij reboots, spanningsverlies, overflows.

Complexe datatypen

- Histograms:
  - Een teller, waarbij je per optreden een waarde wilt bijhouden.
  - Voorbeelden: latency en grootte van HTTP responses.
  - Opgebouwd uit meerdere primitieve waarden:
    - `…_count`: het aantal metingen.
    - `…_sum`: de totale som van alle metingen.
    - `…_bucket{le="X"}`: het aantal metingen ≤ X.
- Summaries:
  - Vergelijkbaar, maar dan buckets op basis van kwantielen.
  - Voor iedere bucket een waarde voor de grens van de bucket.

Voorbeeld van een histogram

```
postfix_queue_message_size_bytes_count{instance="…",job="…"}                613
postfix_queue_message_size_bytes_sum{instance="…",job="…"}                  49681355
postfix_queue_message_size_bytes_bucket{instance="…",job="…",le="1000"}     0
postfix_queue_message_size_bytes_bucket{instance="…",job="…",le="10000"}    1
postfix_queue_message_size_bytes_bucket{instance="…",job="…",le="100000"}   419
postfix_queue_message_size_bytes_bucket{instance="…",job="…",le="1e+06"}    613
postfix_queue_message_size_bytes_bucket{instance="…",job="…",le="1e+07"}    613
postfix_queue_message_size_bytes_bucket{instance="…",job="…",le="1e+08"}    613
postfix_queue_message_size_bytes_bucket{instance="…",job="…",le="1e+09"}    613
postfix_queue_message_size_bytes_bucket{instance="…",job="…",le="+Inf"}     613
```

613 mails met een totale grootte van 49,7 MB in de queue.

Geen enkele mail is groter dan 1 MB.

Best practices voor metrics

- Het is floating point, dus gebruik gewoon basiseenheden.
  - Seconden, geen milliseconden.
  - Bytes, geen kilobytes of kibibytes.
- Zet de eenheid in een suffix: `_seconds`, `_bytes`.
- Zet achter totalen (zoals counters) een `_total` suffix.
- Maak alles binnen één `__name__` van dezelfde dimensie.
  - Dit moet je dus niet doen:

```
http_requests{type="count"} 12
http_requests{type="response_size"} 158237
http_requests{type="duration"} 8.3843
```

Voorbeelden van PromQL

Database:

```
http_cache_usage_bytes{instance="web1.kumina.nl"}    235572486
http_cache_usage_bytes{instance="web2.kumina.nl"}    348453122
http_requests_total{instance="web1.kumina.nl"}       47584
http_requests_total{instance="web2.kumina.nl"}       57237
```

Query:

```
http_requests_total
```

Output:

```
http_requests_total{instance="web1.kumina.nl"}       47584
http_requests_total{instance="web2.kumina.nl"}       57237
```

Voorbeelden van PromQL

Database:

```
http_cache_usage_bytes{instance="web1.kumina.nl"}    235572486
http_cache_usage_bytes{instance="web2.kumina.nl"}    348453122
http_requests_total{instance="web1.kumina.nl"}       47584
http_requests_total{instance="web2.kumina.nl"}       57237
```

Query:

```
{instance="web1.kumina.nl"}
```

Output:

```
http_cache_usage_bytes{instance="web1.kumina.nl"}    235572486
http_requests_total{instance="web2.kumina.nl"}       57237
```

Voorbeelden van PromQL

Database:

```
http_cache_usage_bytes{instance="web1.kumina.nl"}    235572486
http_cache_usage_bytes{instance="web2.kumina.nl"}    348453122
http_requests_total{instance="web1.kumina.nl"}       47584
http_requests_total{instance="web2.kumina.nl"}       57237
```

Query:

```
http_cache_usage_bytes > 300000000
```

Output:

```
http_cache_usage_bytes{instance="web2.kumina.nl"}    348453122
```

Voorbeelden van PromQL

Database:

```
http_cache_usage_bytes{instance="web1.kumina.nl"}    235572486
http_cache_usage_bytes{instance="web2.kumina.nl"}    348453122
http_requests_total{instance="web1.kumina.nl"}       47584
http_requests_total{instance="web2.kumina.nl"}       57237
```

Query:

```
http_cache_usage_bytes > bool 300000000
```

Output:

```
http_cache_usage_bytes{instance="web1.kumina.nl"}    0
http_cache_usage_bytes{instance="web2.kumina.nl"}    1
```

Voorbeelden van PromQL

Database:

```
http_cache_usage_bytes{instance="web1.kumina.nl"}    235572486
http_cache_usage_bytes{instance="web2.kumina.nl"}    348453122
http_requests_total{instance="web1.kumina.nl"}       47584
http_requests_total{instance="web2.kumina.nl"}       57237
```

Query:

```
sum(http_requests_total)
```

Output:

```
{}                                                   104821
```

Voorbeelden van PromQL

Database:

```
http_requests_total{instance="web1.kumina.nl",dc="AMS"}      47584
http_requests_total{instance="web2.kumina.nl",dc="AMS"}      57237
http_requests_total{instance="web3.kumina.nl",dc="FRA"}      540489
http_requests_total{instance="web4.kumina.nl",dc="FRA"}      907948
```

Query:

```
sum(http_requests_total) by (dc)
```

Output:

```
{dc="AMS"}                                                   104821
{dc="FRA"}                                                   1448437
```

Voorbeelden van PromQL

Query:

```
http_requests_total[5m]
```

Output:

```
http_requests_total{instance="web1.kumina.nl"}  45526411 @1487684536.022
                                                45526731 @1487684596.022
                                                45527043 @1487684656.022
                                                45527351 @1487684716.022
                                                45527661 @1487684776.022
```

Voorbeelden van PromQL

Database:

```
http_requests_total{instance="web1.kumina.nl"}   45526411 @1487684536.022
                                                 …
                                                 45527661 @1487684776.022
```

Query:

```
rate(http_requests_total[5m])
```

Output:

```
{instance="web1.kumina.nl"}                      5.208333333
```

Berekening (versimpeld):

```
(45527661 - 45526411) / (1487684776.022 - 1487684536.022) = 5.208333333
```

rate(sum()) of sum(rate())?

- `delta()`, `rate()` en `irate()` doen reset-detectie.
  - if $X_n \geq X_{n-1}$ then $\Delta X = X_n - X_{n-1}$
    else $\Delta X = X_n$
- Werking reset-detectie verslechtert als je gaat sommeren.
  - 1 metric wordt gereset, terwijl 999 andere stijgen.
  - Gaat fout als stijging van de rest meer is dan de enkele daling.
- Goed: `sum(rate(http_requests_total[5m])) by (dc)`
- Fout: `rate(sum(http_requests_total[5m])) by (dc)`

Recording rules

- Queries uitvoeren is snel, maar niet bovennatuurlijk snel.
  - Interactief met je dataset experimenteren kan gewoon.
  - Bouw geen auto-refresh dashboards met veel moeilijke queries!
- Oplossing: recording rules.
  - Laat Prometheus dingen uitrekenen tijdens het scrapen.
  - Slaat resultaat op onder een nieuwe naam.
  - Dashboards kunnen dan voorberekende data gebruiken.
- Syntax:

```
instance:http_requests:rate5m =
    sum(rate(http_requests_total[5m]))
    by (instance)
```

Alerts genereren

```
ALERT MemoryUsageTooHigh
IF memory_free / memory_total < 0.1
FOR 15m
LABELS { severity = “sms” }
ANNOTATIONS {
  problem = “Free memory on {{ $labels.instance }} is low.”,
  solution = “SSH to {{ $labels.instance }} and kill processes.”,
}
```

Alerts verzenden

- Prometheus probeert dingen simpel te houden.
  - Geen wildgroei aan IRC-, e-mail-, Slack-plugins.
  - Geen ondersteuning voor acknowledgements, silences, rotaties.
  - Alerts zijn effectief gewoon simpele booleans.
- Alert Manager is waar de slimmigheid zit.
  - Ontvangt via HTTP POST alert-data van Prometheus.
  - Houdt silences bij: matches op labels om alerts te onderdrukken.
  - Stuurt alerts door naar IRC, e-mail, Slack, PagerDuty, etc.

Data in Prometheus krijgen

- Prometheus probeert dingen simpel te houden.
  - Altijd pull-based; niet push-based.
  - Prometheus snapt één protocol: HTTP GET-requests.
  - Kan targets scrapen op vaste intervallen.
- Client-libraries beschikbaar voor Go, Python, Java, etc.
  - Kan je gebruiken om zelf metrics toe te voegen aan je software.
  - Voegt een klein HTTP-servertje toe aan je software.
  - Exporteert metrics die je declareert in je code.

Voorbeeld van code-annotaties

```
latency := prometheus.NewHistogram(prometheus.HistogramOpts{
    Name:    "http_request_latency_seconds",
    Help:    "Latency of HTTP requests in seconds.",
    Buckets: []float64{0.1, 0.25, 0.50, 1, 2.5, 5, 10},
})

…

latency.Observe(0.0038);
latency.Observe(1.7453);
latency.Observe(0.0057);
```

Exporters

- Lang niet alle software gebruikt de Prometheus-client.
- Oplossing: exporters.
  - Prometheus doet een GET-request naar een exporter.
  - Exporter doet de daadwerkelijke meting.
  - Exporter serveert resultaat als HTTP-response.
- Veel kant-en-klare exporters beschikbaar.
  - ICMP/TCP/UDP health checks: `blackbox_exporter`.
  - Linux/BSD systeem-stats: `node_exporter`.
  - MySQL-stats: `mysqld_exporter`.
  - Java JMX-stats: `jmx_exporter`.

Door Kumina ontwikkelde exporters

Via https://github.com/kumina te verkrijgen:

- `postfix_exporter`: Postfix outbound queue.
- `unbound_exporter`: Unbound DNS-server.
- `dovecot_exporter`: Dovecot POP3/IMAP-server.
- `promacct`: libpcap-gebaseerde traffic accounting.
- `birdwatcher`: BGP-route stats voor BIRD/Calico.

Daarnaast door ons geschreven:

- DRBD- en NFS-collectors voor de `node_exporter`.

Federation

- Prometheus kan zijn data ook exporteren als metrics.
  - Federation: samenwerkende Prometheus-instances.
  - `/federate?match[]={job="nginx"}`
- Lokale Prometheus-instances.
  - Scrapen processen binnen één datacenter.
  - Geven graphs per proces binnen dat datacenter.
- Globale Prometheus-instances.
  - Scrapen lokale Prometheus-instances.
  - Geven graphs per datacenter.

Links

- Prometheus-site: http://prometheus.io/
- Prometheus op Twitter: https://twitter.com/PrometheusIO
- Kumina's blog: https://blog.kumina.nl/
- Kumina's GitHub-pagina: https://github.com/kumina