Open Source monitoring using Prometheus

Ed Schouten <ed@kumina.nl>

About the speaker

Ed Schouten <ed@kumina.nl>

- 2003-*: User/developer of Open Source Software.
  - 2008-*: developer at FreeBSD and LLVM.
  - 2015-*: author of CloudABI: easy sandboxing for UNIX.
- 2011-2012: Part-time employed by Kumina while studying.
- 2012-2014: Employed by Google.
  - Software Engineer/Site Reliability Engineer in Munich.
- 2016-*: Full-time employed by Kumina.

Today's set of questions

- What is Prometheus?
- How does Prometheus work?
- How can you store data in Prometheus?
- How can you extract data from Prometheus?
- How does Prometheus do alerting?
- (How does Kumina use Prometheus?)

The history of Prometheus

- ~2003: Google develops Borg: a cluster manager.
  - Existing monitoring systems don't seem to work well with it.
    - Bad support for creating/removes targets to monitor dynamically.
    - Too much focussed on per-target state, as opposed to global state.
  - Google also develops Borgmon: **Borg mon**itoring system.
- 2012: Ex-Googlers at SoundCloud miss Borgmon.
  - Prometheus: a reimplementation of Borgmon.
  - Open Source: Apache 2.0 licensed.
  - Somewhat cleaner than Borgmon.
- (2014: Google creates an Open Source Borg, Kubernetes)

An overview of Prometheus

- Prometheus is broadly applicable.
  - No Nagios-style networks/hosts/services hierarchy.
  - Not intended to keep track of just servers.
    - The temperature of rooms in a house.
    - The number of people walking into a shop.
    - Stock prices and currency exchange rates.
- Prometheus is versatile.
  - Nagios can only do trigger-based alerting.
  - Munin can only do graphing of metrics.
  - Prometheus can use the same dataset to do both.
  - Graphs and alerts can be designed freely on top of the dataset.

Prometheus' data model

- Prometheus is a 2D database.
  - Horizontal axis: time.
    - Has a configurable finite retention period.
  - Vertical axis: unique identifier for every metric.
    - Identifier: a dictionary of string keys and string values.
      `{__name__="http_requests_total",job="nginx",instance="kumina.nl"}`
    - The value of `__name__` can be placed before the `{}`.
      `http_requests_total{job="nginx",instance="kumina.nl"}`
  - In the cells: 64-bits IEEE floating-point values.
  - On-disk format: every metric stored in its own compressed file.
  - PromQL: query/computation language for Prometheus.

Primitive data types

- Gauges (Dutch: ‘meters’):
  - Examples: memory usage, temperature, exchange rate.
  - Can both increase and decrease over time.
- Counters (Dutch: ‘tellers’):
  - Examples: number of requests, customers entering a shop.
  - Can only increase.
  - Decrease indicates a reboot, restart or counter overflow.

Examples of counters

```
http_requests_total{instance="…",job="…",vhost="…",code="2xx"}  12308
http_requests_total{instance="…",job="…",vhost="…",code="3xx"}  5739
http_requests_total{instance="…",job="…",vhost="…",code="4xx"}  141
http_requests_total{instance="…",job="…",vhost="…",code="5xx"}  4
```

Complex data types

- Histograms:
  - A counter for which you want to track a per-event sample.
  - Example: number of HTTP requests and their latency.
  - Constructed by combining multiple primitive metrics:
    - `…_count`: total number of samples.
    - `…_sum`: total sum of all samples.
    - `…_bucket{le="X"}`: number of samples ≤ X.
- Summaries:
  - Similar purpose, but buckets are based on quantiles.
  - For every quantile, it exports the bucket's boundary.
  - Not used as frequently. Will not be discussed any further.

Example of a 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 emails with a combined size of 49,7 MB.

All emails are $1000 < x \leq 1000000$ bytes in size.

Best practices for designing metrics

- Values are floating-point, so just use base units.
  - Seconds, but not milliseconds.
  - Bytes, but not kilobytes or kibibytes.
- Add the unit as a suffix: `_seconds`, `_bytes`.
- Add a `_total` suffix after counters.
- Everything with one `__name__` should be aggregatable.
  - This doesn't make sense, as you can't add these values together:

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

Examples of 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
```

Output:

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

Examples of 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
```

Examples of 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:

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

Examples of PromQL

Database:

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

Query:

```
sum(http_cache_usage_bytes)
```

Output:

```
{}                                                              1553258
```

Examples of PromQL

Database:

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

Query:

```
sum(http_cache_usage_bytes) by (dc)
```

Output:

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

Examples of 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
```

Examples of 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
```

Computation (simplified):

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

Recording rules

- Prometheus is fast, but not supernaturally fast.
  - It's fine to run complex queries for ad hoc data exploration.
  - Don't build auto-refreshing dashboards with complex queries!
- Solution: add recording rules.
  - Lets Prometheus compute results as it's scraping them.
  - Binds the results of an expression to a new named metric.
  - Dashboards should grab values from the new metric.
- Syntax:

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

Generating alerts

Database:

```
memory_free_bytes{instance="web3.kumina.nl"}          5046272
memory_installed_bytes{instance="web3.kumina.nl"}     2147483648
```

Alert rule:

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

Sending alerts

- Prometheus tries to keep things simple.
  - No proliferation of IRC, email, Slack plugins.
  - No support for acknowledgements, silences, pager rotations.
  - Prometheus just treats alerts as simple booleans.
- Alert Manager is where Prometheus keeps the smartness.
  - Receives alerts from Prometheus through HTTP POST requests.
  - Keeps track of silences: patterns on labels to suppress alerts.
  - Has logic for coalescing similar alerts.
  - Has plugins for contacting IRC, email, Slack, PagerDuty, etc.
  - Alert messages link back to the right Prometheus instance.

Getting data into Prometheus

- Prometheus tries to keep things simple.
  - Always pull based; not push based.
  - Prometheus supports just one protocol: HTTP GET requests.
  - Can scrape targets at configured intervals.
  - Interval is increased if Prometheus is overloaded (‘rushed mode’).
- ‘Client libraries’ available for Go, Python, Java, etc.
  - Allows you to add your own metrics to your software.
  - Adds a tiny HTTP server that exports current metric values.
  - Metrics can be declared as objects in your packages/classes.

Example of code annotations in Go

```
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

- Most programs don’t use the Prometheus client library yet.
- Solution: exporters.
  - Prometheus sends a HTTP GET request to the exporter.
  - Exporter fetches and parses metrics from its target.
  - Exporter returns metrics in Prometheus’ native format.
- Many exporters readily available:
  - ICMP/TCP/UDP health checks: `blackbox_exporter`.
  - Linux/BSD OS-level stats: `node_exporter`.
  - MySQL stats: `mysqld_exporter`.
  - Java JMX stats (Tomcat, Cassandra, etc.): `jmx_exporter`.

Metrics exporters developed by Kumina

Available at https://github.com/kumina:

- `postfix_exporter`: Postfix mail server.
- `dovecot_exporter`: Dovecot POP3/IMAP server.
- `libvirt_exporter`: KVM based virtual machines.
- `unbound_exporter`: Unbound DNS server.
- `openvpn_exporter`: OpenVPN client and server.
- `promacct`: libpcap-based network traffic accounting.
- `birdwatcher`: BGP route statistics for BIRD/Calico.

Additionally written by Kumina:

- DRBD and NFS collectors for the official `node_exporter`.

Federation

- Prometheus can export its data in its own format.
  - Federation: cooperating Prometheus instances.
  - `/federate?match[]=http_requests_total`
- Local Prometheus instances:
  - Scrape processes within a single data center.
  - Can give per-process graphs.
  - Have recording rules for computing per-datacenter stats.
- Global Prometheus instances:
  - Scrape values from recording rules from local instances.
  - Can give per-datacenter graphs.

Links

- Official Prometheus site: http://prometheus.io/
- Prometheus on Twitter: https://twitter.com/PrometheusIO
- Grafana: https://grafana.com/
- Kumina's blog: https://blog.kumina.nl/
- Kumina's GitHub page: https://github.com/kumina