



# Monitoring with Prometheus at Devskiller

---

Łukasz Szczęsny

[luk@wyb.cz](mailto:luk@wyb.cz), [luk@devskiller.com](mailto:luk@devskiller.com)

Warszawa, 2018-06-04

# \$ whoami

- Infrastructure Lead at Devskiller
- FOSS and Open Hardware lover
- Automation enthusiast
- Co-organizer of the WarLUG (<http://warszawa.linux.org.pl>)



# What is Prometheus?

- An OSS monitoring and alerting toolkit written in Go
- Inspired by Google's Borgmon monitoring system
- Started at SoundCloud in 2012
- Joined Cloud Native Computing Foundation in 2016
- Graphs and dashboards included

# When to use Prometheus?

- It works well for recording any numeric time series
- It's designed for reliability - each Prometheus server is standalone system, there are almost no external dependencies, extremely easy setup
- It's suited well for monitoring infrastructure and distributed systems
- But what about Nagios/check\_mk/Sensu/Graphite/younameit!?

# When not to use Prometheus?

- When 100% accuracy is required, such as for per-request billing
- When you need a long-term archival system
- When you need downsampling

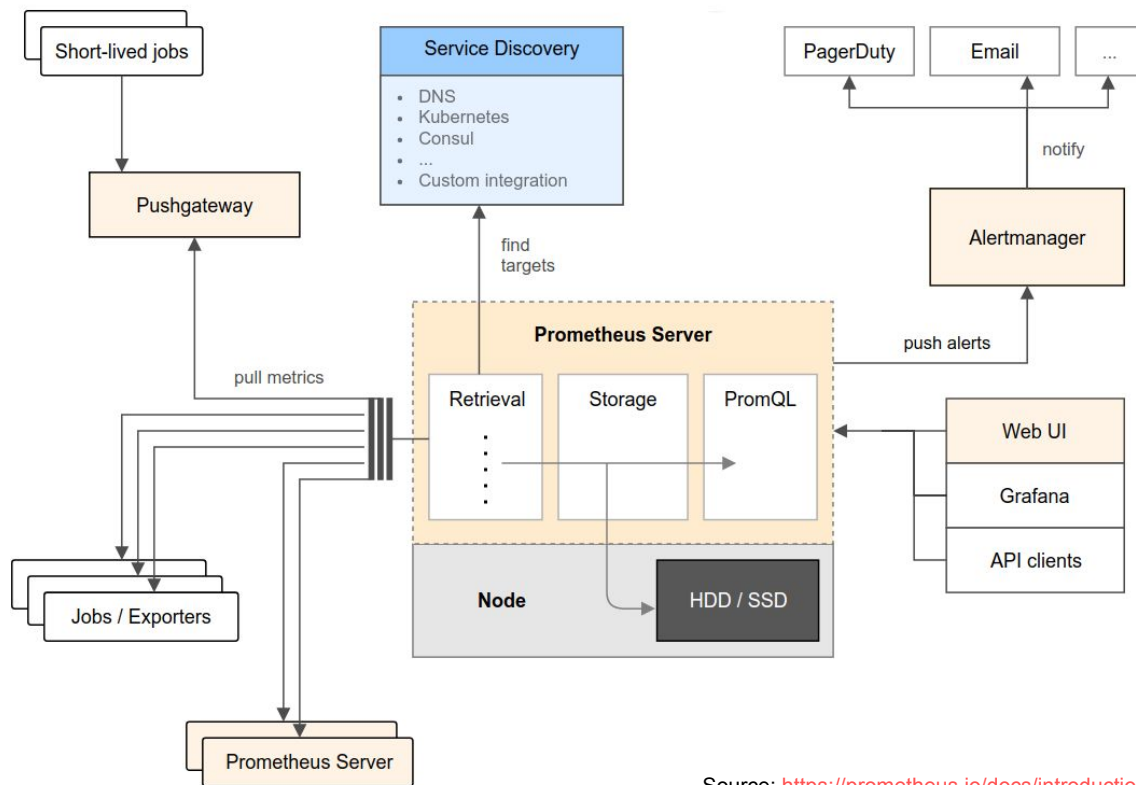
# What we use Prometheus for?

- Infrastructure monitoring
- Application monitoring
- JVM monitoring
- cron (batch) jobs monitoring

# Key concepts

- Multidimensional data model - time series data identified by metric name and key/value pairs
- Flexible query language - PromQL
- Pulling time series via HTTP
- Pushing time series is supported via an intermediary gateway

# Architecture overview



Source: <https://prometheus.io/docs/introduction/overview/#architecture>



# Exporters

- Software exposing existing metrics from 3rd party systems
- Official exporters
  - [node\\_exporter](#)
  - [jmx\\_exporter](#)
  - [blackbox\\_exporter](#)
  - [mysqld\\_exporter](#)
  - [snmp\\_exporter](#)
  - [mtail](#)
  - ...
- Multiple 3rd party exporters

# Targets

```
scrape_configs:  
  - job_name: 'prometheus'  
    static_configs:  
      - targets: ['localhost:9090']  
  
  - job_name: 'alertmanager'  
    static_configs:  
      - targets: ['localhost:9093']  
  
  - job_name: 'node'  
    static_configs:  
      - targets: ['localhost:9100']
```



Prometheus ☰

## Targets

**alertmanager (1/1 up)**

Endpoint	State	Labels	Last Scrape	Error
http://localhost:9093/metrics	UP	Instance="localhost:9093"	424ms ago	

**node (1/1 up)**

Endpoint	State	Labels	Last Scrape	Error
http://localhost:9100/metrics	UP	Instance="localhost:9100"	2.562s ago	

**prometheus (1/1 up)**

Endpoint	State	Labels	Last Scrape	Error
http://localhost:9090/metrics	UP	Instance="localhost:9090"	3.72s ago	

# Jobs and instance

- ***instance*** - a target which is being scraped
- ***job*** - collection of instance with the same purpose

```
job: node
  - instance 1: 1.2.3.4:5670
  - instance 2: 1.2.3.4:5671
  - instance 3: 5.6.7.8:5670
  - instance 4: 5.6.7.8:5671
```

- Service discovery for targets

# Service discovery

- Automagically retrieve scrape targets
- A must in dynamic environments where instances are spawned and killed all the time
- Generic `file_sd_config` can be used e.g. to add targets from Ansible inventory

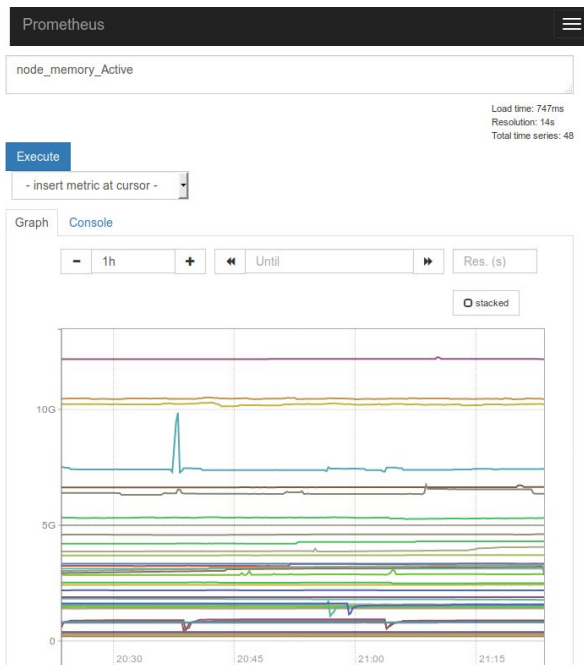
# Direct instrumentation

- Official libraries
  - Go
  - Java or Scala
  - Python
  - Ruby
- Multiple 3rd party libraries

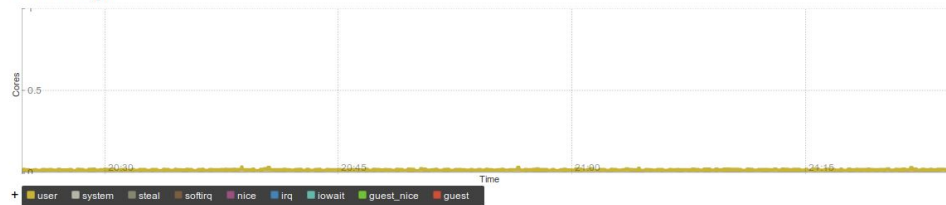
# Graphs and dashboards

- Built-in expression browser
- Console templates
- [Grafana](#)

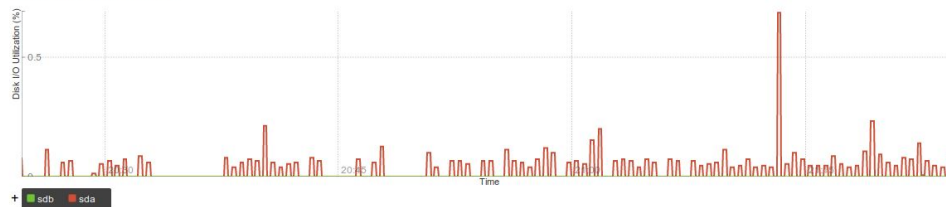
# Graphs and dashboards



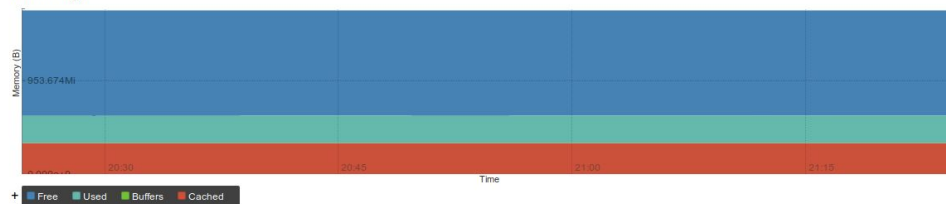
## CPU Usage



## Disk I/O Utilization



## Memory



# Metrics types

- Counter
- Gauge
- Histogram
- Summary



# Data model

```
<metric name>{<label name>=<label value>, ...}
```

# Data model

`<metric name>{<label name>=<label value>, ...}`

# Data model

```
<metric name>{<label name>=<label value>, ...}
```

# Data model

```
node_disk_reads_merged{device="sda"}
```

# Data model

```
node_disk_reads_merged{device="sda"}
```

# Data model

```
node_disk_reads_merged{device="sda"}
```

# Data model

```
node_disk_reads_merged{device="sda", instance="server-01"}
```

# Data model

```
node_disk_reads_merged{device="sda", instance!="server-01"}
```



# Data model

```
node_disk_reads_merged{device="sda", instance=~"server-.*"}
```

# Data model

```
node_disk_reads_merged{device="sda", instance!~"server-.*"}
```

# Data model

```
node_disk_reads_merged{device="sda"}[5m]
```

# Metrics and labels naming

- Metrics should have a suffix describing the unit in plural form, e.g. `node_memory_usage_bytes`
- Single unit - do not mix bytes with seconds!
- Preferably use base units - seconds, bytes, grams etc.
- Accumulating count has ***total*** as a suffix, e.g. `https_requests_total`
- Labels names should be chosen to differentiate the characteristics of the thing which is being measured, e.g. `http_requests_total` differentiate the request method, e.g. `method="GET|POST|PUT|..."`

# Data types

- **Instant vector** - a set of time series containing a single sample for each time series, all sharing the same timestamp
- **Range vector** - a set of time series containing a range of data points over time for each time series
- **Scalar** - a simple numeric floating point value
- **String** - a simple string value; currently unused

# Data types

```
node_arp_entries{device="wlp2s0"}[1m]
```

vs

```
node_arp_entries{device="wlp2s0"}
```

Graph	Console																										
	<table><thead><tr><th>Element</th><th>Value</th></tr></thead><tbody><tr><td>node_arp_entries{device="wlp2s0",instance="localhost:9100",job="node"}</td><td>1 @1528119635.606</td></tr><tr><td></td><td>1 @1528119640.606</td></tr><tr><td></td><td>1 @1528119645.606</td></tr><tr><td></td><td>1 @1528119650.606</td></tr><tr><td></td><td>1 @1528119655.606</td></tr><tr><td></td><td>1 @1528119660.606</td></tr><tr><td></td><td>1 @1528119665.606</td></tr><tr><td></td><td>1 @1528119670.606</td></tr><tr><td></td><td>1 @1528119675.606</td></tr><tr><td></td><td>1 @1528119680.606</td></tr><tr><td></td><td>1 @1528119685.606</td></tr><tr><td></td><td>1 @1528119690.606</td></tr></tbody></table>	Element	Value	node_arp_entries{device="wlp2s0",instance="localhost:9100",job="node"}	1 @1528119635.606		1 @1528119640.606		1 @1528119645.606		1 @1528119650.606		1 @1528119655.606		1 @1528119660.606		1 @1528119665.606		1 @1528119670.606		1 @1528119675.606		1 @1528119680.606		1 @1528119685.606		1 @1528119690.606
Element	Value																										
node_arp_entries{device="wlp2s0",instance="localhost:9100",job="node"}	1 @1528119635.606																										
	1 @1528119640.606																										
	1 @1528119645.606																										
	1 @1528119650.606																										
	1 @1528119655.606																										
	1 @1528119660.606																										
	1 @1528119665.606																										
	1 @1528119670.606																										
	1 @1528119675.606																										
	1 @1528119680.606																										
	1 @1528119685.606																										
	1 @1528119690.606																										

Graph	Console				
	<table><thead><tr><th>Element</th><th>Value</th></tr></thead><tbody><tr><td>node_arp_entries{device="wlp2s0",instance="localhost:9100",job="node"}</td><td>1</td></tr></tbody></table>	Element	Value	node_arp_entries{device="wlp2s0",instance="localhost:9100",job="node"}	1
Element	Value				
node_arp_entries{device="wlp2s0",instance="localhost:9100",job="node"}	1				

# PromQL - operators

+	==	and
-	!=	or
*	>	unless
/	<	
%	>=	
^	<=	

## PromQL - aggregation operators

sum

min

max

avg

stddev

without / by

stdvar

count

count\_values

bottomk

topk

quantile



# PromQL - grouping

without () / by ()

# PromQL - matching

ignoring () / on ()

group\_left () / group\_right ()

# PromQL - matching

```
// input
method_code:http_errors:rate5m{method="get", code="500"} 24
method_code:http_errors:rate5m{method="get", code="404"} 30
method_code:http_errors:rate5m{method="put", code="501"} 3
method_code:http_errors:rate5m{method="post", code="500"} 6
method_code:http_errors:rate5m{method="post", code="404"} 21

method:http_requests:rate5m{method="get"} 600
method:http_requests:rate5m{method="del"} 34
method:http_requests:rate5m{method="post"} 120

// query
method_code:http_errors:rate5m{code="500"} / ignoring(code) method:http_requests:rate5m

// output
{method="get"} 0.04 // 24 / 600
{method="post"} 0.05 // 6 / 120
```

# PromQL - matching

```
// input
method_code:http_errors:rate5m{method="get", code="500"} 24
method_code:http_errors:rate5m{method="get", code="404"} 30
method_code:http_errors:rate5m{method="put", code="501"} 3
method_code:http_errors:rate5m{method="post", code="500"} 6
method_code:http_errors:rate5m{method="post", code="404"} 21

method:http_requests:rate5m{method="get"} 600
method:http_requests:rate5m{method="del"} 34
method:http_requests:rate5m{method="post"} 120

// query
method_code:http_errors:rate5m / ignoring(code) group_left method:http_requests:rate5m

// output
{method="get", code="500"} 0.04 // 24 / 600
{method="get", code="404"} 0.05 // 30 / 600
{method="post", code="500"} 0.05 // 6 / 120
{method="post", code="404"} 0.175 // 21 / 120
```

# PromQL - functions

abs()  
absent()  
ceil()  
changes()  
clamp\_max()  
clamp\_min()  
day\_of\_month()  
day\_of\_week()  
days\_in\_month()  
delta()

deriv()  
exp()  
floor()  
histogram\_quantile()  
holt\_winters()  
hour()  
idelta()  
increase()  
irate()  
label\_join()

label\_replace()  
ln()  
log2()  
log10()  
minute()  
month()  
predict\_linear()  
rate()  
resets()  
round()

scalar()  
sort()  
sort\_desc()  
sqrt()  
time()  
timestamp()  
vector()  
year()  
<aggr>\_over\_time()

# PromQL samples

```
http_requests_total{job="apiserver", handler="/api/comments"}
```

# PromQL samples

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

# PromQL samples

```
sum(rate(http_requests_total[5m])) by (job)
```



# PromQL samples

```
topk(3, sum(rate(instance_cpu_time_ns[5m])) by (app, proc))
```

# PromQL samples

```
count(node_uname_info) by (release)
```

# PromQL samples

```
sum by(kubernetes_pod_name)
(container_memory_usage_bytes{kubernetes_namespace="kube-system"})
```

# PromQL samples

```
rate(demo_api_request_duration_seconds_count{status="500",job="demo"}[5m]) * 50
  > on(job, instance, method, path)
  rate(demo_api_request_duration_seconds_count{status="200",job="demo"}[5m])
```

# PromQL samples

```
histogram_quantile(0.9, rate(demo_api_request_duration_seconds_bucket{job="demo"}[5m])) > 0.05  
and  
rate(demo_api_request_duration_seconds_count{job="demo"}[5m]) > 1
```

# PromQL samples

```
sum(rate(http_requests_total{service="foo", code=~"5.."}[2m])) /  
  sum(rate(http_requests_total{service="foo"}[2m])) * 100 > 0
```

# PromQL samples

```
predict_linear(node_filesystem_free{job="node"}[1h], 4 * 3600) < 0
```

# Rules

- Recording rules
  - precomputing frequently needed or computationally expensive expressions
  - save their result as a new set of time series
- Alerting rules
  - defining alert conditions based on PromQL expressions



# Rules

groups:

- name: example

  - rules:

    - record: job:http\_inprogress\_requests:sum

      - expr: sum(http\_inprogress\_requests) by (job)

    - alert: HighErrorRate

      - expr: job:request\_latency\_seconds:mean5m{job="myjob"} > 0.5

      - for: 10m

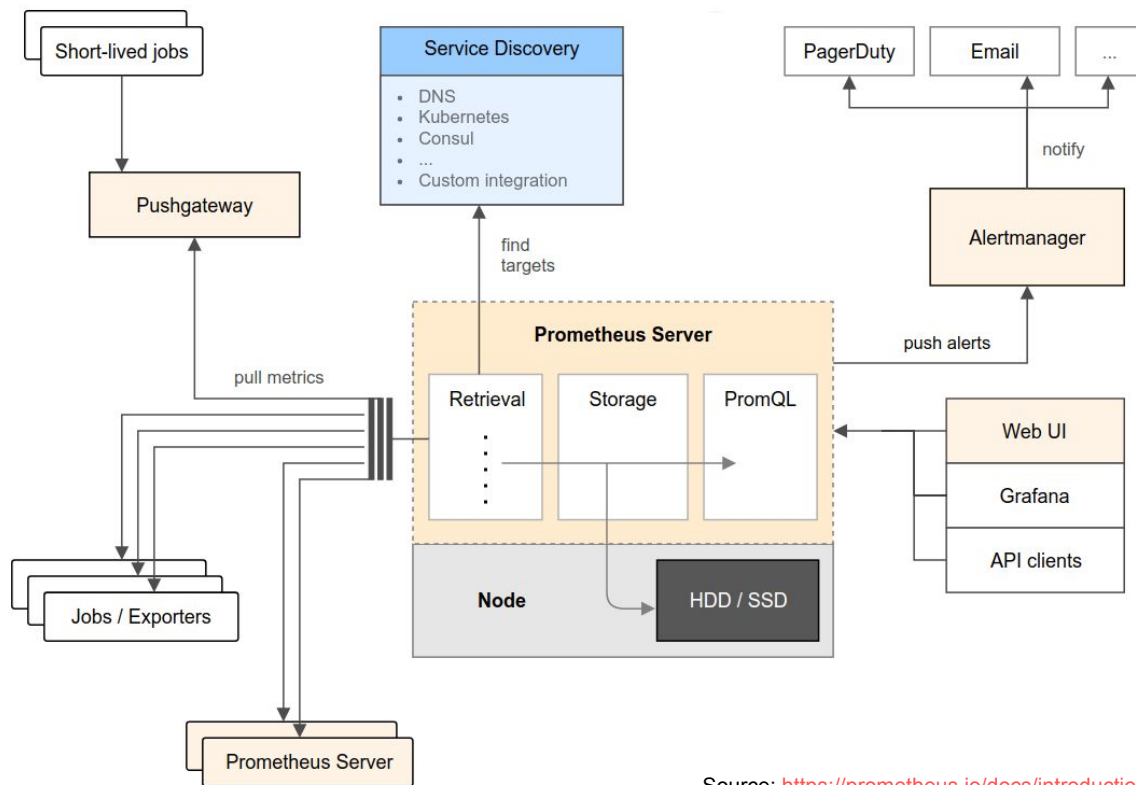
      - labels:

        - severity: page

      - annotations:

        - summary: High request latency

# Architecture overview



Source: <https://prometheus.io/docs/introduction/overview/#architecture>

# Alertmanager

- Receives alerts from Prometheus
- Takes care of deduplicating, grouping, and routing them
- Alerts inhibition
- Silencing alerts

# Alerting rules

```
groups:  
- name: example  
  rules:  
- alert: InstanceDown  
  expr: up == 0  
  for: 5m  
  labels:  
    severity: page  
  annotations:  
    summary: "Instance {{ $labels.instance }} down"  
    description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 5 minutes."  
    dashboard: "https://grafana.example.com/dashboard/db/job?var-job={{ $labels.instance }}"  
    runbook: "https://wiki.example.com/runbook/IntanceDown"
```

# Alerting rules

```
groups:  
- name: example  
  rules:  
- alert: InstanceDown  
  expr: up == 0  
  for: 5m  
  labels:  
    severity: page  
  annotations:  
    summary: "Instance {{ $labels.instance }} down"  
    description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 5 minutes."  
    dashboard: "https://grafana.example.com/dashboard/db/job?var-job={{ $labels.instance }}"  
    runbook: "https://wiki.example.com/runbook/IntanceDown"
```

# Alertmanager - grouping

route:

```
# The labels by which incoming alerts are grouped together. For example,  
# multiple alerts coming in for cluster=A and alertname=LatencyHigh would  
# be batched into a single group.  
group_by: ['alertname', 'cluster', 'service']
```

# Alertmanager - routing #1

```
route:
  # A default receiver
  receiver: team-X-mails
  # The child route trees.
  routes:
    # This routes performs a regular expression match on alert labels to
    # catch alerts that are related to a list of services.
    - match_re:
        service: ^(foo1|foo2|baz)$
        receiver: team-X-mails
        # The service has a sub-route for critical alerts, any alerts
        # that do not match, i.e. severity != critical, fall-back to the
        # parent node and are sent to 'team-X-mails'
        routes:
          - match:
              severity: critical
              receiver: team-X-pager
```

# Alertmanager - routing #2

receivers:

- name: 'team-X-mails'  
email\_configs:
  - to: 'team-X+alerts@example.org'
  
- name: 'team-X-pager'  
email\_configs:
  - to: 'team-X+alerts-critical@example.org'

pagerduty\_configs:

  - service\_key: <team-X-key>



# Alertmanager - inhibit rules

```
inhibit_rules:  
- source_match:  
    severity: 'critical'  
  target_match:  
    severity: 'warning'  
  # Apply inhibition if the alertname is the same.  
  equal: ['alertname', 'cluster', 'service']
```

# Operating Prometheus

- High Availability
  - Alertmanager mesh configuration HA
  - For Prometheus itself just run two instances with the same configuration! Alertmanager will de-duplicate alerts
- Scalability
  - Divide workload by jobs
  - Instance per AZ, DC, etc.
  - Federation
- Capacity planning

# Meta-monitoring

- Create an always firing alert (*expr: vector(1)*) in Prometheus
- Get an account on cron monitoring service
- Create a webhook in e.g. PD or Alertmanager to call cron monitoring service API - this alert should not page people!
- Voilà!

# On-call



Questions



# Thank you!

- [@wybczu](#)
- [luk@wyb.cz](mailto:luk@wyb.cz)
- IRC: wybczu@freenode
- <https://keybase.io/wybczu>
- GPG key ID: 0x2EB8C2A248AD1541  
Key fingerprint = 5BEA 0D72 A27F ABB6 2F23 FF2A 2EB8 C2A2 48AD 1541

# Resources

1. <https://prometheus.io/docs/introduction/overview/>
2. <https://github.com/prometheus>
3. <https://promcon.io/>
4. <https://www.robustperception.io/blog/>
5. <https://groups.google.com/forum/#!forum/prometheus-users>
6. Long-term storage:
  - a. <https://www.influxdata.com/blog/prometheus-influxdb-thoughts/>
  - b. <https://github.com/improbable-eng/thanos>