Introduction to Prometheus

Mehran Salmani

What is Prometheus?

- Monitoring system - checking and measuring state of systems
- Time-series database for storing real-time metrics
- Open-source, built at SoundCloud, using Go language
- Used to monitor nodes (machines), clusters and applications
- A pull-based tool
- Actively scrapes targets to get metrics from
- Monitoring is important to follow the state of your services and take action before trouble happens.
- Prometheus also contains alert manager to alert you based on a condition

Prometheus Metric Types

- Counters: only goes up or gets reseted (e.g number of all requests to service)
- Gauges: goes up or down (e.g. resource utilization or temperature)
- Histogram: take many measurements of a value to later calculate averages or percentiles. Ranges of possible values must be known upfront. (e.g. latency)
- Summary: alternative to histograms. You don’t know what the range of values (bucket boundaries) will be.

Exposition Format

- Text-based
- Line oriented
- Tokens in a line can be separated by any number of blanks/tabs
- Lines starting with # are comments, unless HELP or TYPE comes after
- syntax:

```
metric_name [
  "{" label_name "=" `"` label_value `"` { "," label_name "=" `"` label_value `"` } [ "," ] "}"
] value [ timestamp ]
```

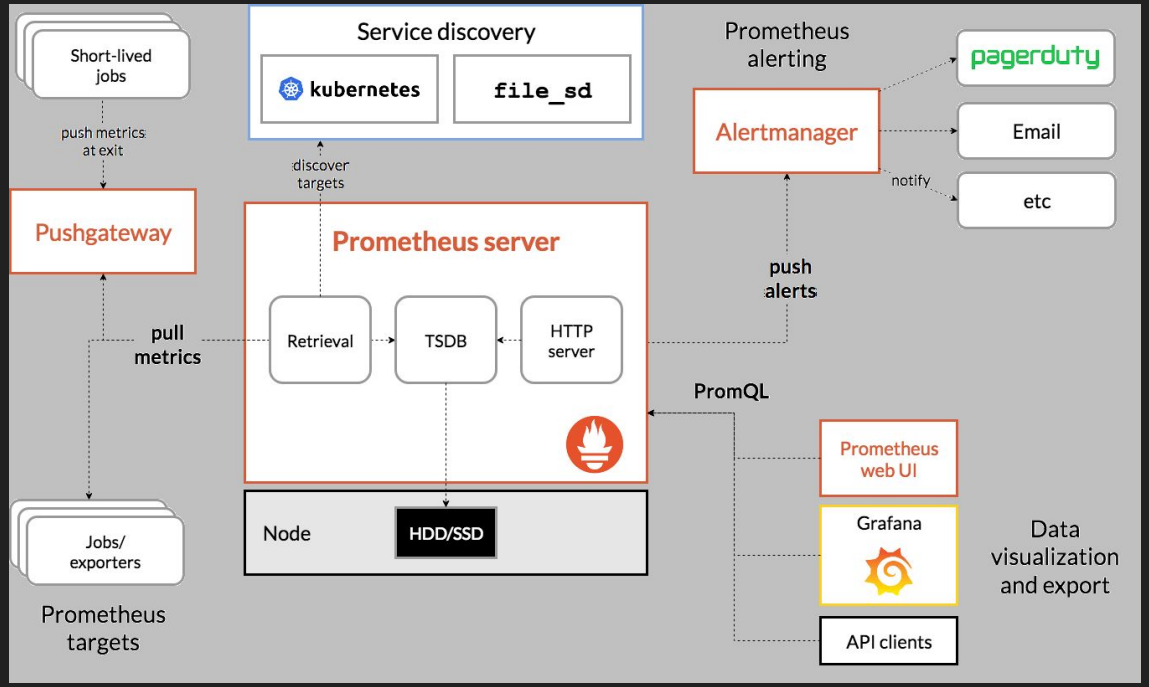
https://prometheus.io/docs/instrumenting/exposition_formats/#text-format-details

Exposition Format: sample

```
# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total counter
http_requests_total{method="post",code="200"} 1027 1395066363000
http_requests_total{method="post",code="400"}    3 1395066363000


# Minimalistic line:
metric_without_timestamp_and_labels 12.47

# A weird metric from before the epoch:
something_weird{problem="division by zero"} +Inf -3982045

# A histogram, which has a pretty complex representation in the text format:
# HELP http_request_duration_seconds A histogram of the request duration.
# TYPE http_request_duration_seconds histogram
http_request_duration_seconds_bucket{le="0.05"} 24054
http_request_duration_seconds_bucket{le="0.1"} 33444
http_request_duration_seconds_bucket{le="0.2"} 100392
http_request_duration_seconds_bucket{le="0.5"} 129389
http_request_duration_seconds_bucket{le="1"} 133988
http_request_duration_seconds_bucket{le="+Inf"} 144320
http_request_duration_seconds_sum 53423
http_request_duration_seconds_count 144320

# Finally a summary, which has a complex representation, too:
# HELP rpc_duration_seconds A summary of the RPC duration in seconds.
# TYPE rpc_duration_seconds summary
rpc_duration_seconds{quantile="0.01"} 3102
rpc_duration_seconds{quantile="0.05"} 3272
rpc_duration_seconds{quantile="0.5"} 4773
rpc_duration_seconds{quantile="0.9"} 9001
rpc_duration_seconds{quantile="0.99"} 76656
rpc_duration_seconds_sum 1.7560473e+07
rpc_duration_seconds_count 2693
```

https://prometheus.io/docs/instrumenting/exposition_formats/#text-format-details

Prometheus Architecture



https://prometheus.io/docs/introduction/overview/

Scrape Targets

- Defined in “scrape_configs” in Prometheus configuration file
- In “scrape_configs”, a list of jobs are defined each including at least 1 target
- Targets that Prometheus scrapes every “scrape_interval”
- Different ways to define targets (service discovery)
  - static_configs
  - file_sd_configs (file-based)
  - dns-based (A, AAAA and SRV records)
  - Cloud-based (AWS, Google Cloud, etc)

Prometheus Rules

- Recording rules: precompute frequent expressions from existing metrics and to save their result as derived time series data
- Alerting rules: define alert condition
- Both defined in the same way, in yaml files
- Rule file path should be written in prometheus configuration file, as value of “rule_files” parameter
- Evaluation interval can be set using “interval” parameter, the default is “evaluation_interval” in prometheus configuration file.

Alerting

- A good alert is:
  - For a good reason
  - At the right time
  - With a right tempo
  - Having useful information
- Alert anti-patterns:
  - Sending too many alerts.
  - Misclassification of alerts (sending to wrong people or with incorrect urgency)
  - Sending alerts that are not useful
  - Alert not having enough context, so not immediately useful

AlertManager

- Two steps
  - Alert rules of Prometheus sending alerts to AlertManager
  - AlertManager manages and sends notification to destinations
- Alert states:
  - Inactive
  - Pending (expression is true and there is a “for” in configuration)
  - Firing (expression continues to be true during “for”)
- We can define templates for notification message

PromQL

- Expression/query language
- A nested functional language
- Expressions that can be used in PromQL:
  - String
  - Scalar
  - Instance vector
  - Range vector
- Supports arithmetic and logical operators, aggregations and builtin functions

Example: sum by (job) (
  rate(http_requests_total[5m])
)

Grafana

- Open-source dashboard
- Used for visualization
- Can integrate with Prometheus, ElasticSearch, Postgresql, etc
- Written in Go programming language
- Different data sources can be mixed in one graph

References

- https://prometheus.io
- Turnbull, James. Monitoring with Prometheus. Turnbull Press, 2018.