Monitoring with Prometheus (DO-PRO)

Keywords

Monitoring, Prometheus, Metric Collection, Service Discovery, Query, Alerting, Exporter

References

- https://prometheus.io/docs/
- Monitoring with Prometheus - James Turnbull, 2018
- Prometheus: Up & Running - Brian Brazil, 2018

Monitoring

What is monitoring?

- From a technology perspective, **monitoring** is the tools and processes by which you measure and manage your technology systems. But monitoring is much more than that.
- Monitoring provides the translation to business value from metrics generated by your systems and applications. Your monitoring system translates those metrics into a measure of user experience. That measure provides feedback to the business to help ensure it’s delivering what customers want.

Prometheus

The Prometheus Backstory

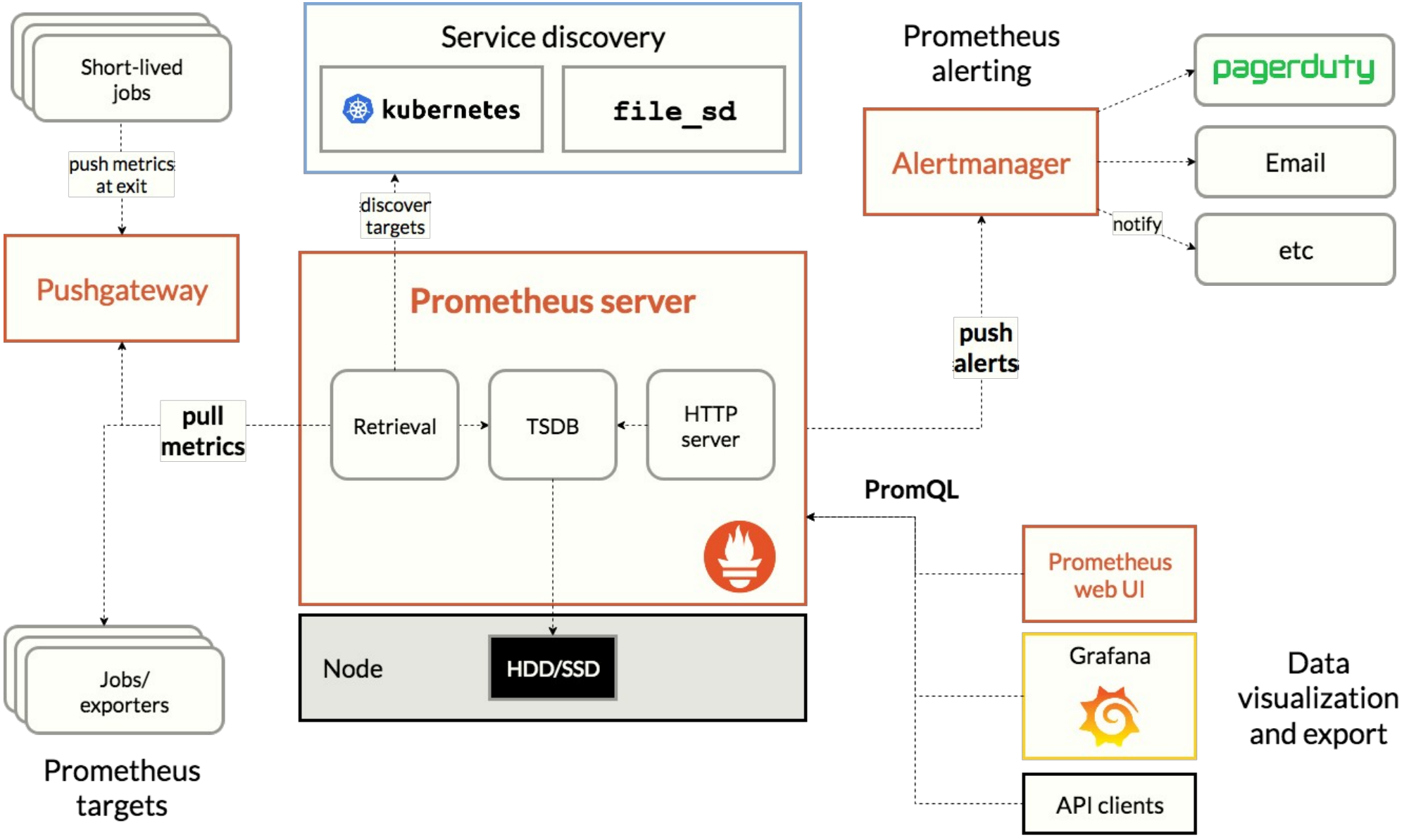
- Prometheus owes its inspiration to Google's Borgmon. It was originally developed by Matt T. Proud, an ex-Google SRE, as a research project.
- Proud joined SoundCloud, he teamed up with another engineer, Julius Volz, to develop Prometheus in earnest. Other developers joined the effort, and it continued development internally at SoundCloud, culminating in a public release in January 2015.
- Prometheus was primarily designed to provide near real-time introspection monitoring of dynamic cloud- and container-based microservices, services, and applications.

The Prometheus

CLOUD NATIVE
COMPUTING FOUNDATION

- Prometheus is an open source, metrics-based monitoring system. Of course, Prometheus is far from the only one of those out there, so what makes it notable?
- Prometheus does one thing and it does it well. It has a simple yet powerful data model and a query language that lets you analyse how your applications and infrastructure are performing. It does not try to solve problems outside of the metrics space, leaving those to other more appropriate tools.
- Prometheus is written in Go, open source, and licensed under the Apache 2.0 license. It is incubated under the Cloud Native Computing Foundation (CNCF).

Prometheus Architecture



Prometheus Architecture (1)

- Prometheus works by scraping or pulling time series data exposed from applications. The time series data is exposed by the applications themselves often via client libraries or via proxies called exporters, as HTTP endpoints. Exporters and client libraries exist for many languages, frameworks, and open-source applications—for example, for web servers like Apache and databases like MySQL.
- Prometheus also has a push gateway you can use to receive small volumes of data—for example, data from targets that can’t be pulled, like transient jobs or targets behind firewalls.

Exporter

A) Databases: MySQL, MongoDB & PostgreSQL

B) Hardware: Node & Ubiquiti UniFi

C) Messaging: RabbitMQ & Kafka

D) Storage: Ceph, Gluster & Hadoop

E) HTTP: Apache, HAProxy, Nginx, & Varnish

* https://prometheus.io/docs/instrumenting/exporters/

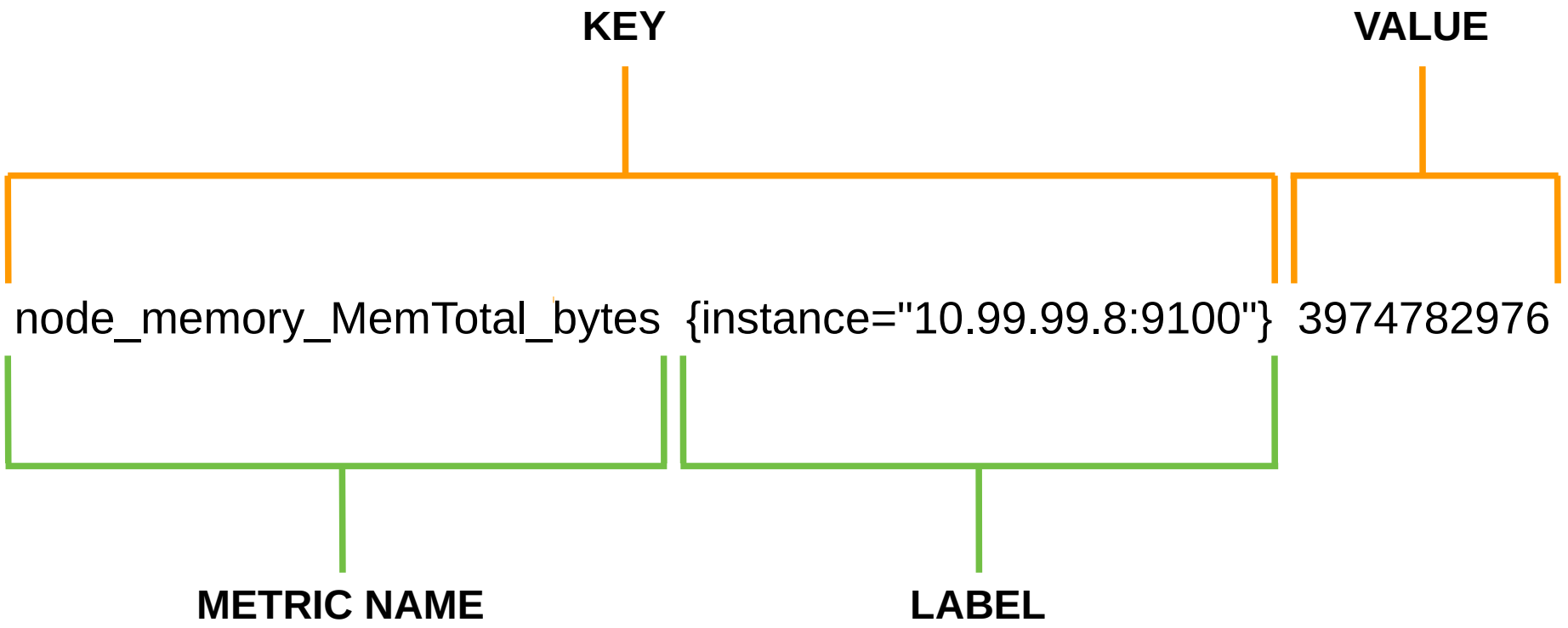
Metric Collection

- Prometheus calls the source of metrics it can scrape endpoints. An endpoint usually corresponds to a single process, host, service, or application. To scrape an endpoint, Prometheus defines configuration called a target. This is the information required to perform the scrape—for example, how to connect to it, what metadata to apply, any authentication required to connect, or other information that defines how the scrape will occur.
- Groups of targets are called jobs. Jobs are usually groups of targets with the same role—for example, a cluster of Apache servers behind a load balancer. That is, they’re effectively a group of like processes.

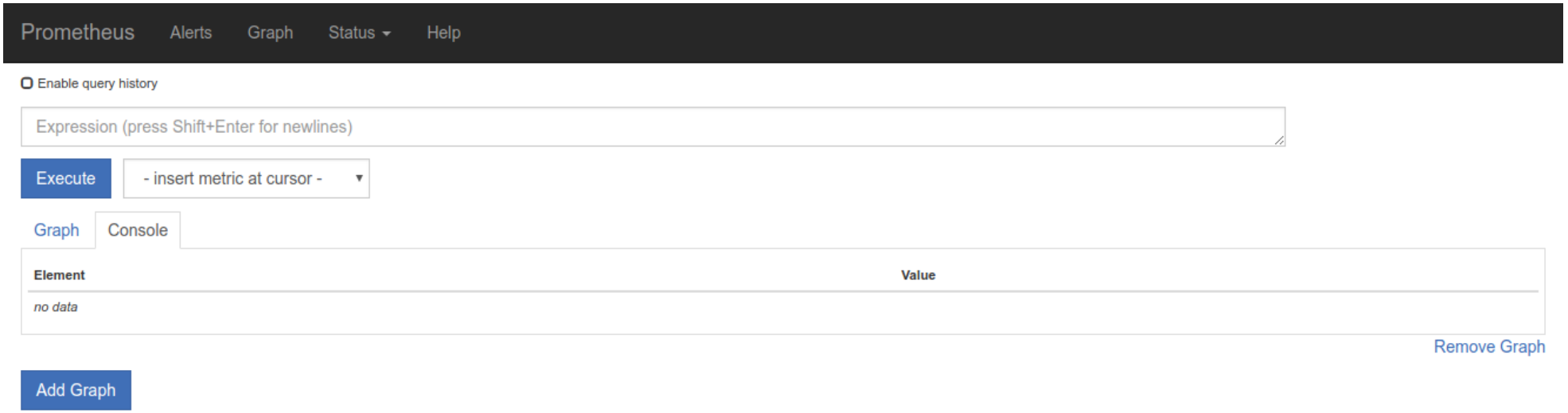
Metric

```
# HELP go_gc_duration_seconds A summary of the GC invocation durations.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.3243e-05
go_gc_duration_seconds{quantile="0.25"} 4.5987e-05
go_gc_duration_seconds{quantile="0.5"} 6.0116e-05
go_gc_duration_seconds{quantile="0.75"} 0.000107072
go_gc_duration_seconds{quantile="1"} 0.005037768
go_gc_duration_seconds_sum 0.225177899
go_gc_duration_seconds_count 1267
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 50
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.10.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.569536e+07
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.2292890696e+10
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.646404e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 8.5329512e+07
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 7.248151698002132e-06
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 2.306048e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 2.569536e+07
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 3.6265984e+07
```

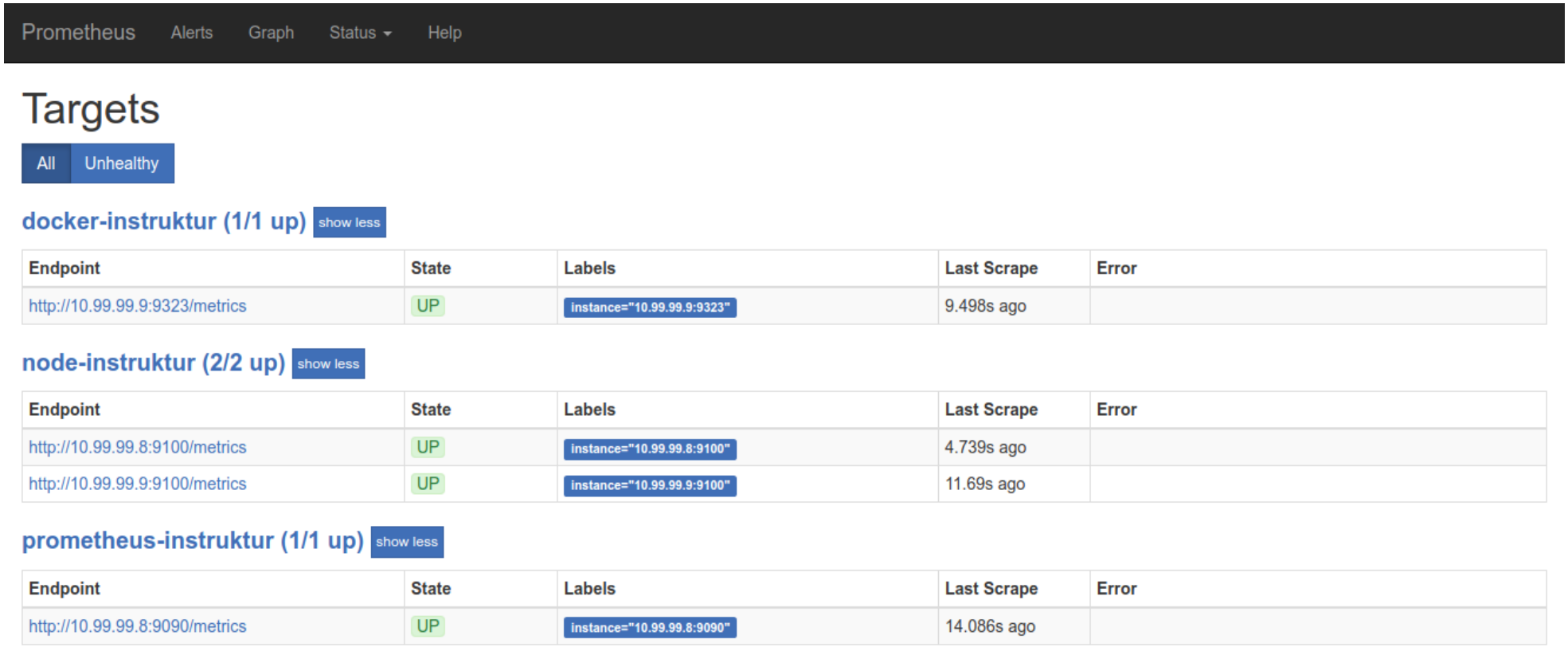
Metric


KEY
VALUE
node_memory_MemTotal_bytes
{instance="10.99.99.8:9100"}
3974782976
METRIC NAME
LABEL

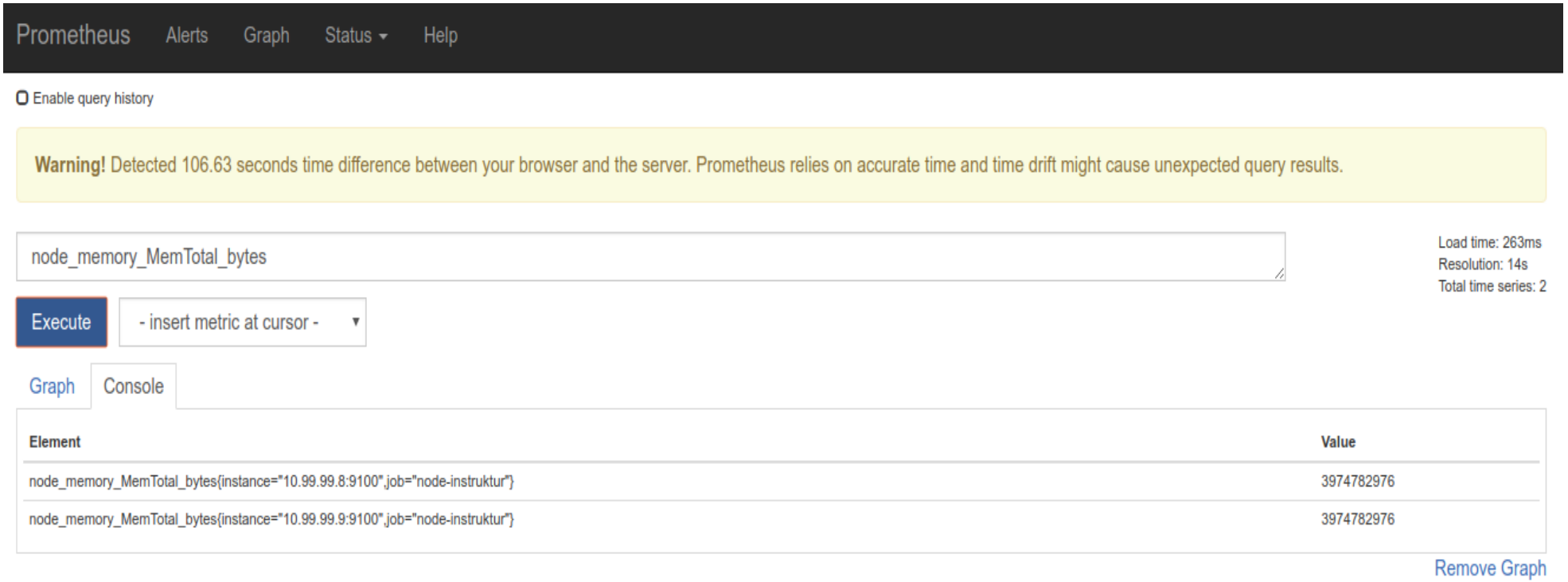
Dashboard


Prometheus
Alerts
Graph
Status
Help
Enable query history
Expression (press Shift+Enter for newlines)
Execute
- insert metric at cursor -
Graph
Console
Element
Value
no data
Remove Graph
Add Graph

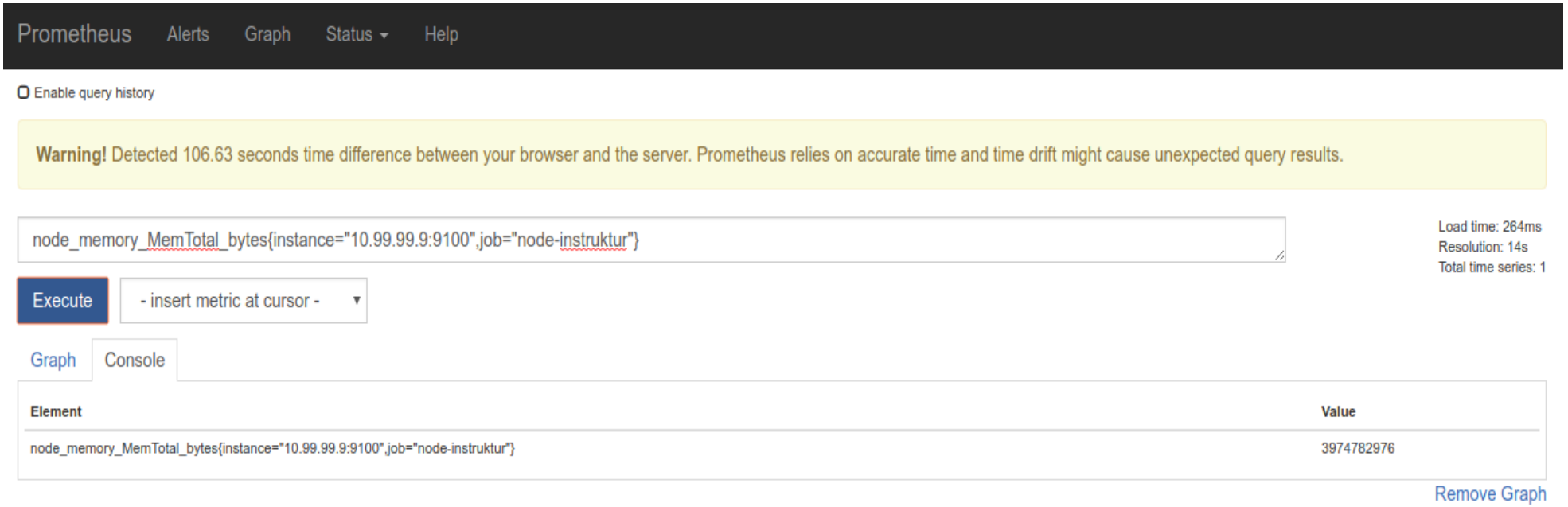
Dashboard (Targets)



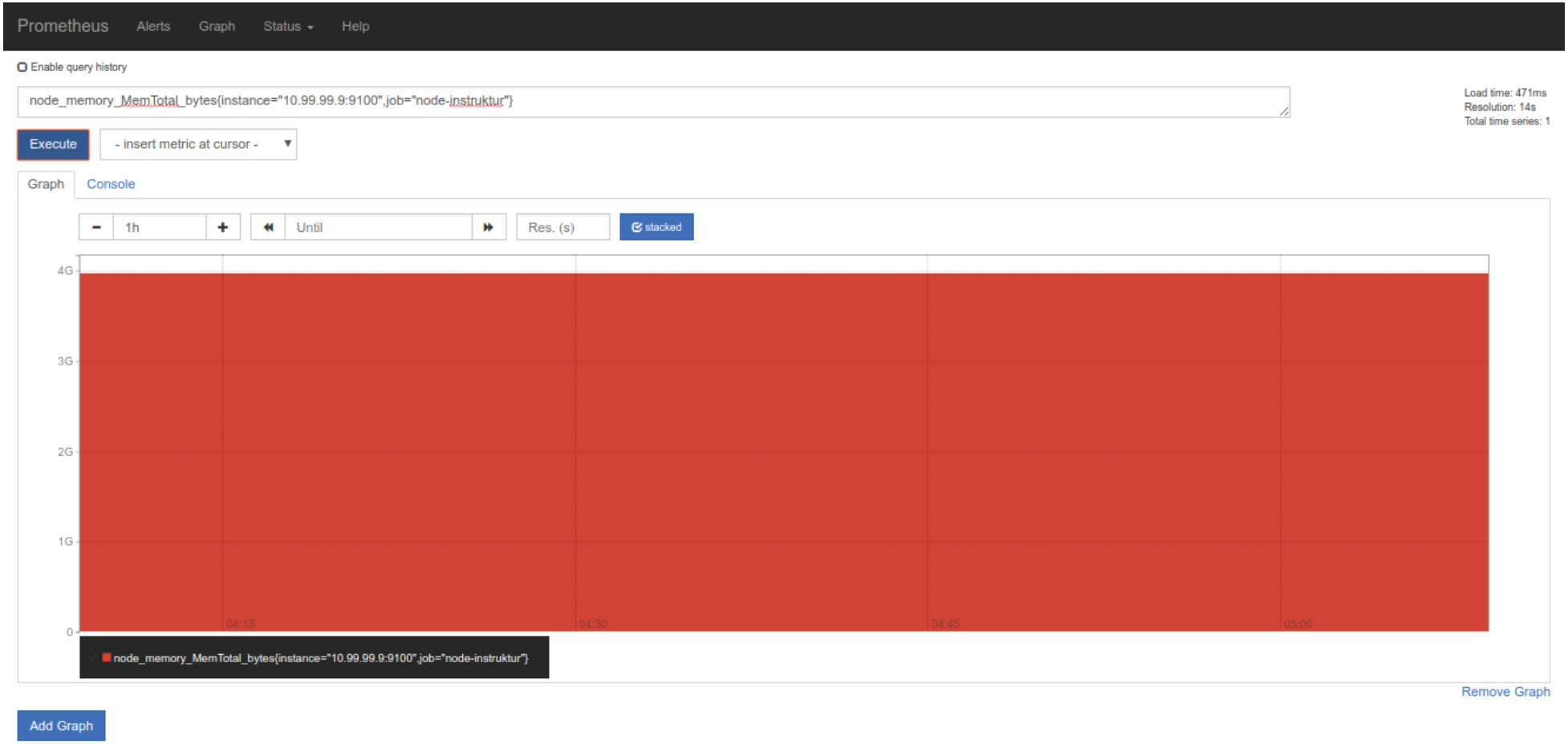
Dashboard (Query 1)



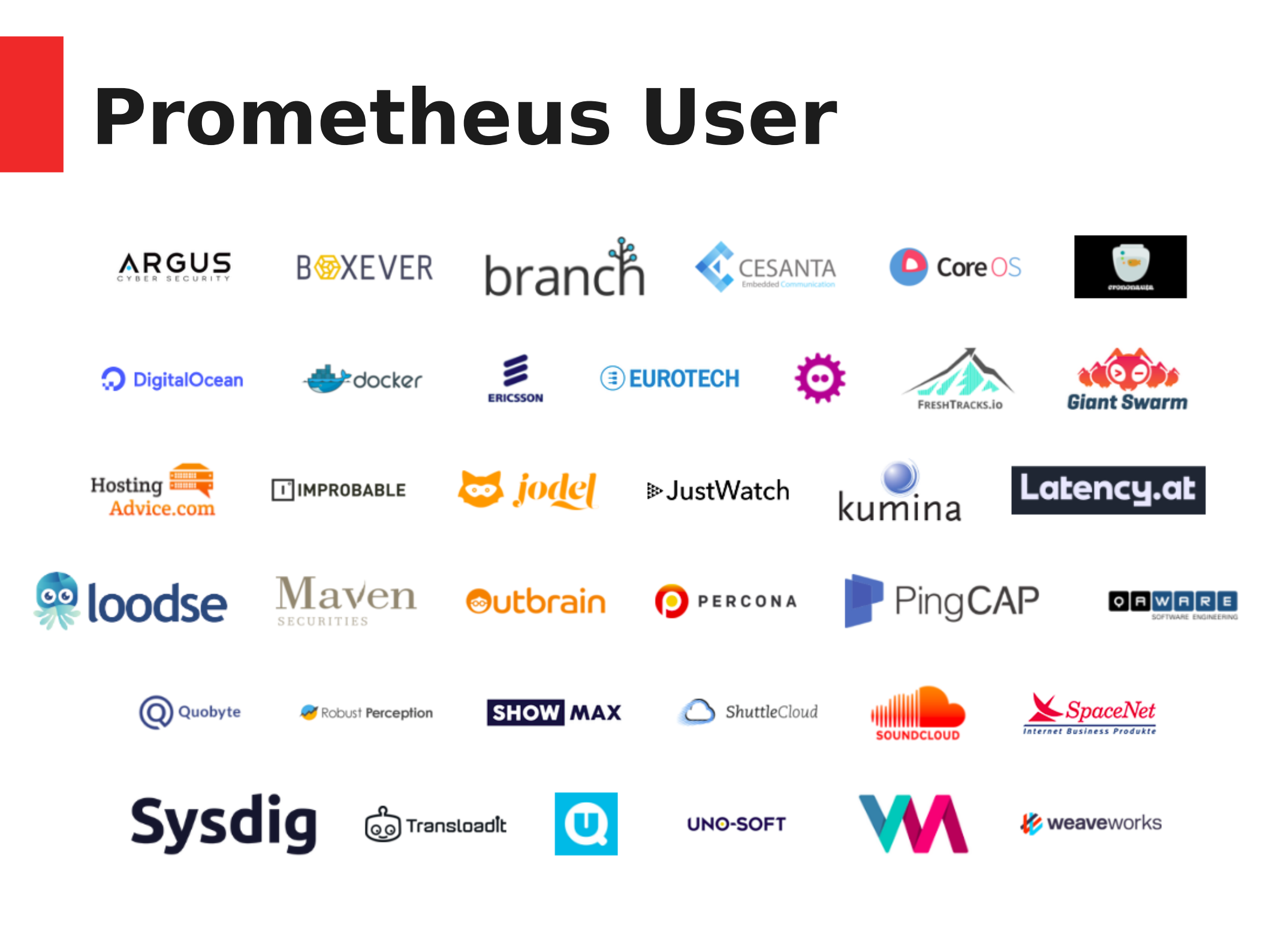
Dashboard (Query 2)



Dashboard (Query 3)



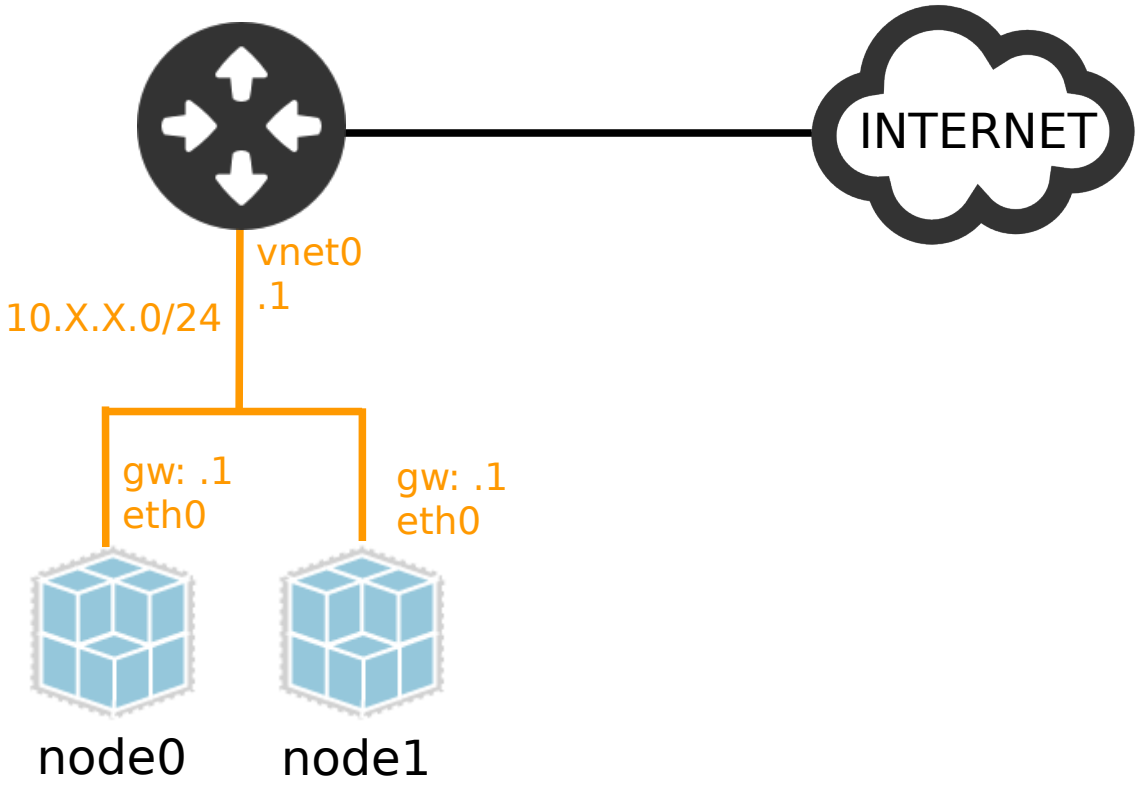
Prometheus User



Lab

Monitoring with Prometheus

Lab 1 Topology



* node0: Prometheus Server & Node Exporter
* node1: Node Exporter

Metric

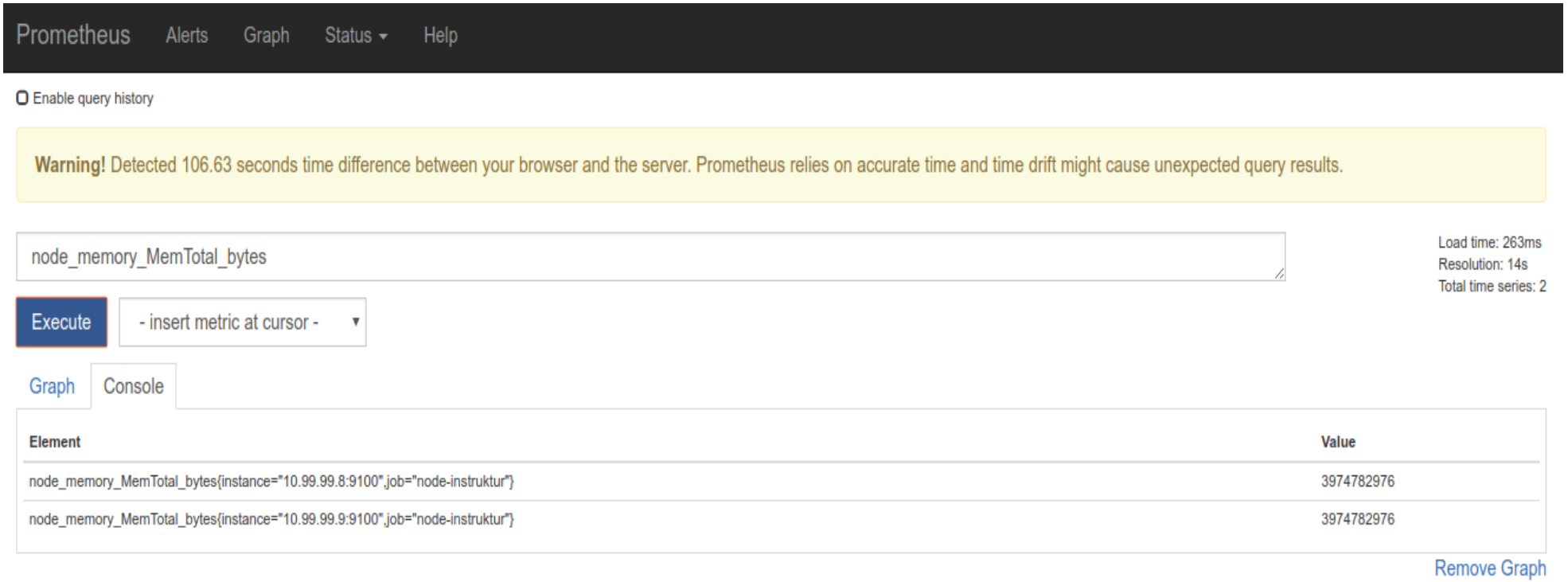
```
# HELP go_gc_duration_seconds A summary of the GC invocation durations.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.3243e-05
go_gc_duration_seconds{quantile="0.25"} 4.5987e-05
go_gc_duration_seconds{quantile="0.5"} 6.0116e-05
go_gc_duration_seconds{quantile="0.75"} 0.000107072
go_gc_duration_seconds{quantile="1"} 0.005037768
go_gc_duration_seconds_sum 0.225177899
go_gc_duration_seconds_count 1267
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 50
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.10.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.569536e+07
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.2292890696e+10
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.646404e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 8.5329512e+07
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 7.248151698002132e-06
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 2.306048e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 2.569536e+07
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 3.6265984e+07
```

Visualization

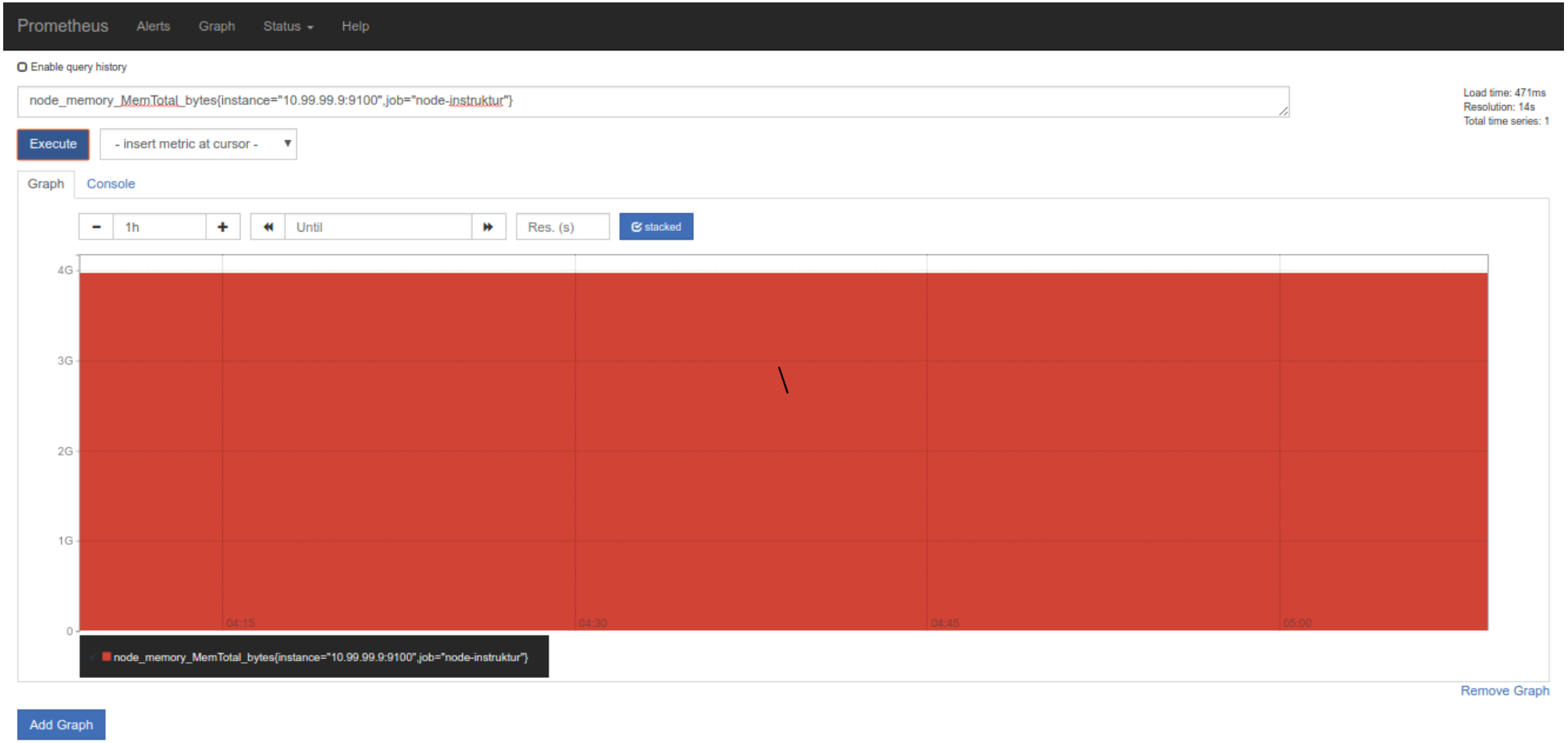
Visualization of Metrics can be made in 3 ways:

1) Expression Browser
2) Console Templates
3) Grafana

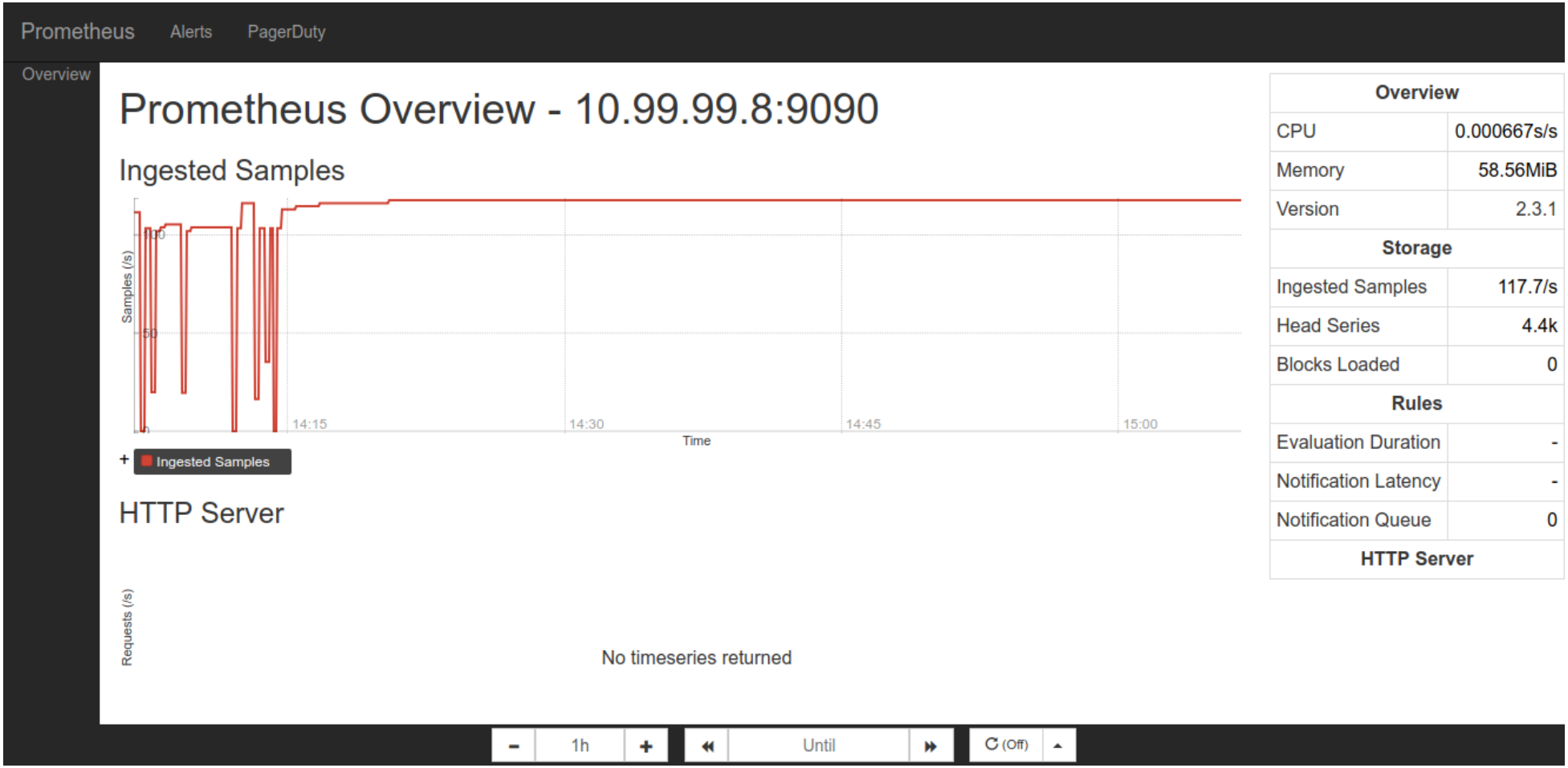
Expression Browser (1)



Expression Browser (2)



Console Templates



* https://prometheus.io/docs/visualization/consoles/

Grafana

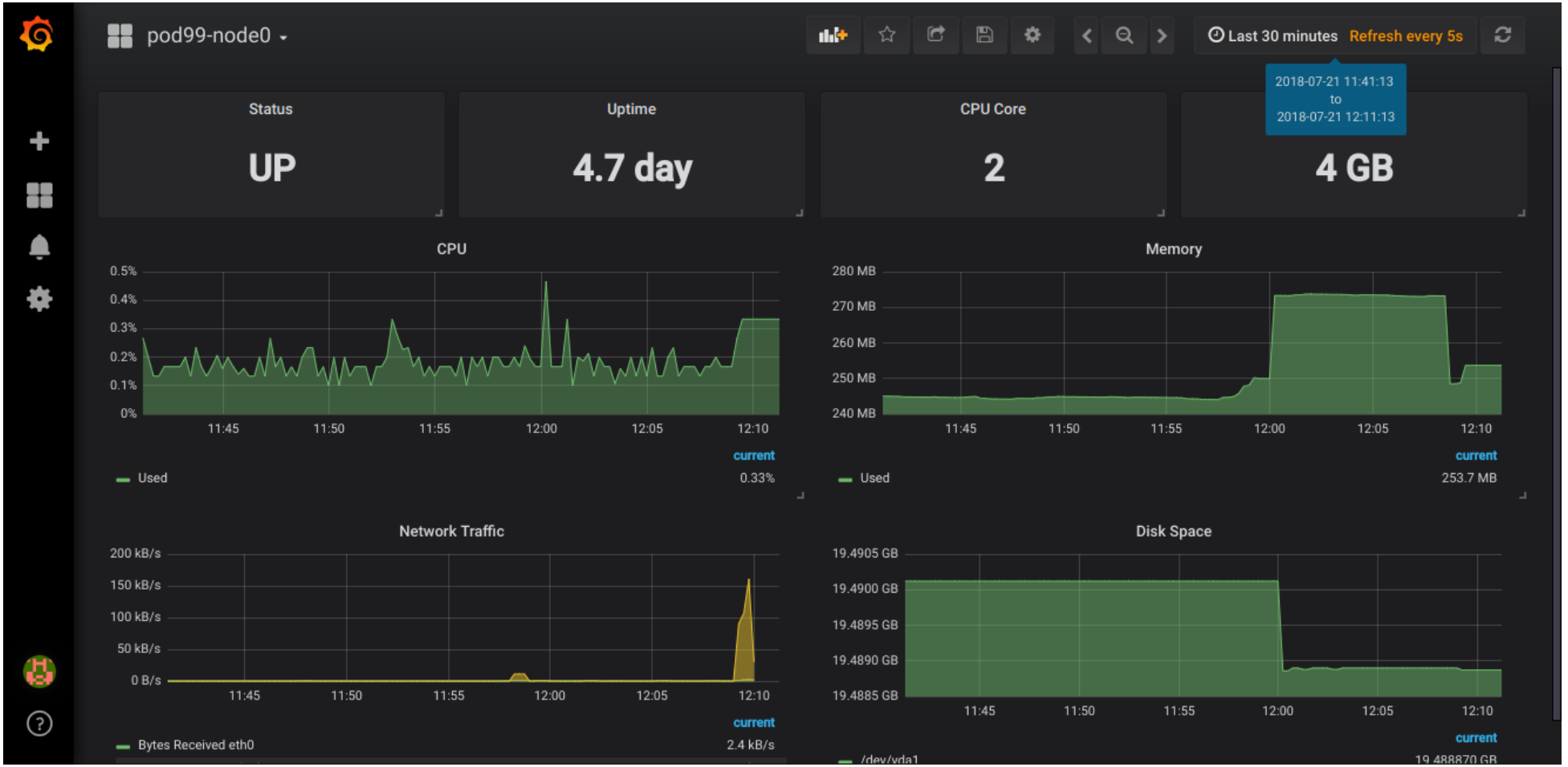
- The open platform for beautiful analytics and monitoring.
- Running on:
  1) Linux
  2) Windows
  3) Mac
  4) Docker
  5) ARM

Grafana

- Data Source:
  1) Elasticsearch
  2) InfluxDB
  3) MySQL
  4) PostgreSQL
  5) Prometheus

* https://grafana.com/plugins?type=datasource

Grafana


pod99-node0
Last 30 minutes
Refresh every 5s
2018-07-21 11:41:13
to
2018-07-21 12:11:13
Status
UP
Uptime
4.7 day
CPU Core
2
4 GB
CPU
0.5%
0.4%
0.3%
0.2%
0.1%
0%
11:45
11:50
11:55
12:00
12:05
12:10
current
Used
0.33%
Memory
280 MB
270 MB
260 MB
250 MB
240 MB
11:45
11:50
11:55
12:00
12:05
12:10
current
Used
253.7 MB
Network Traffic
200 kB/s
150 kB/s
100 kB/s
50 kB/s
0 B/s
11:45
11:50
11:55
12:00
12:05
12:10
current
Bytes Received eth0
2.4 kB/s
Disk Space
19.4905 GB
19.4900 GB
19.4895 GB
19.4890 GB
19.4885 GB
11:45
11:50
11:55
12:00
12:05
12:10
current
/dev/vda1

Alerting

Alerting with Prometheus is separated into two parts:

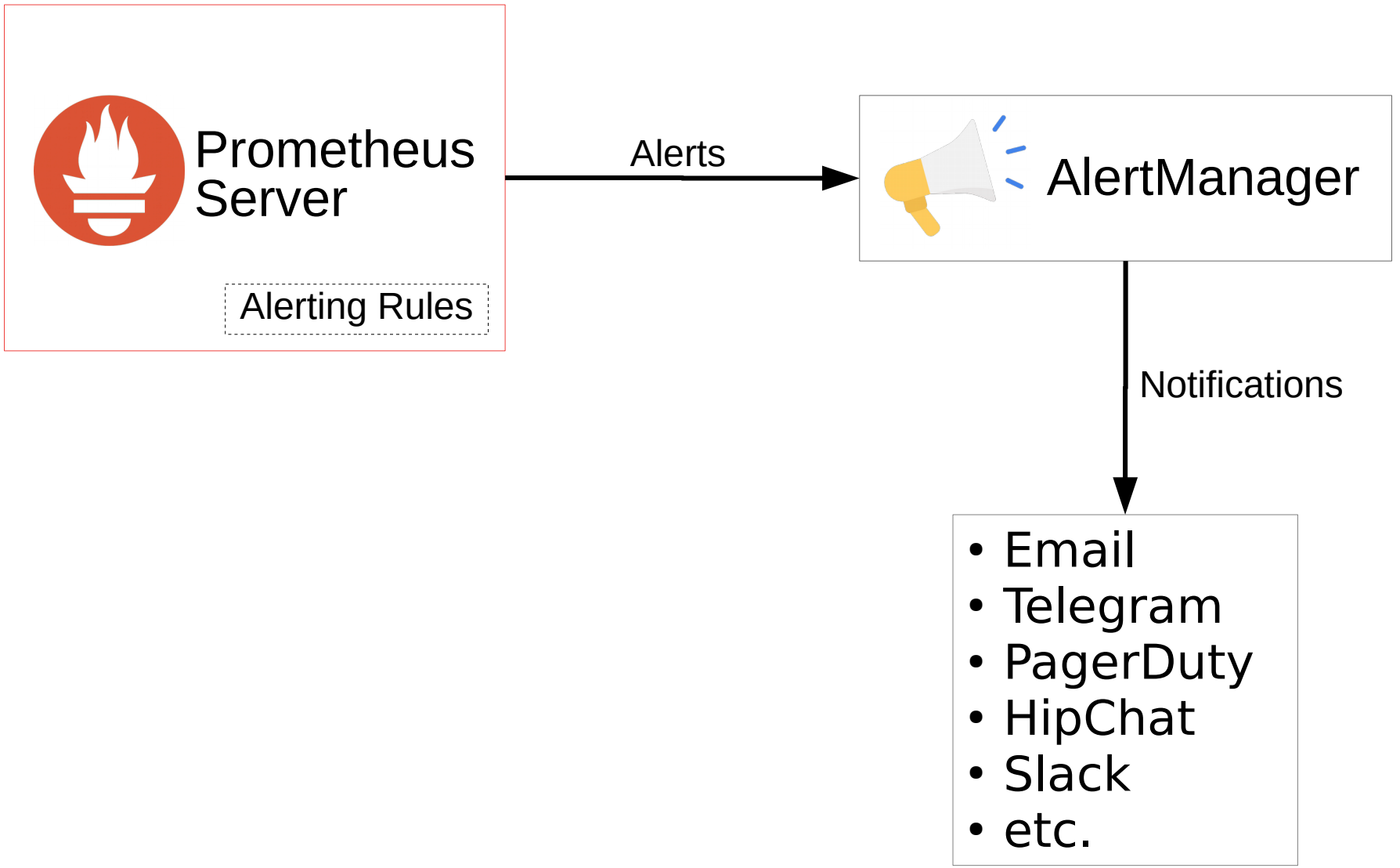
1) Alerting Rules

   Defining conditions in the form of PromQL expressions that are continuously evaluated, and any resulting time series become alerts.

2) AlertManager

   Sending out notifications via email, Telegram PagerDuty, HipChat, Slack, and etc.

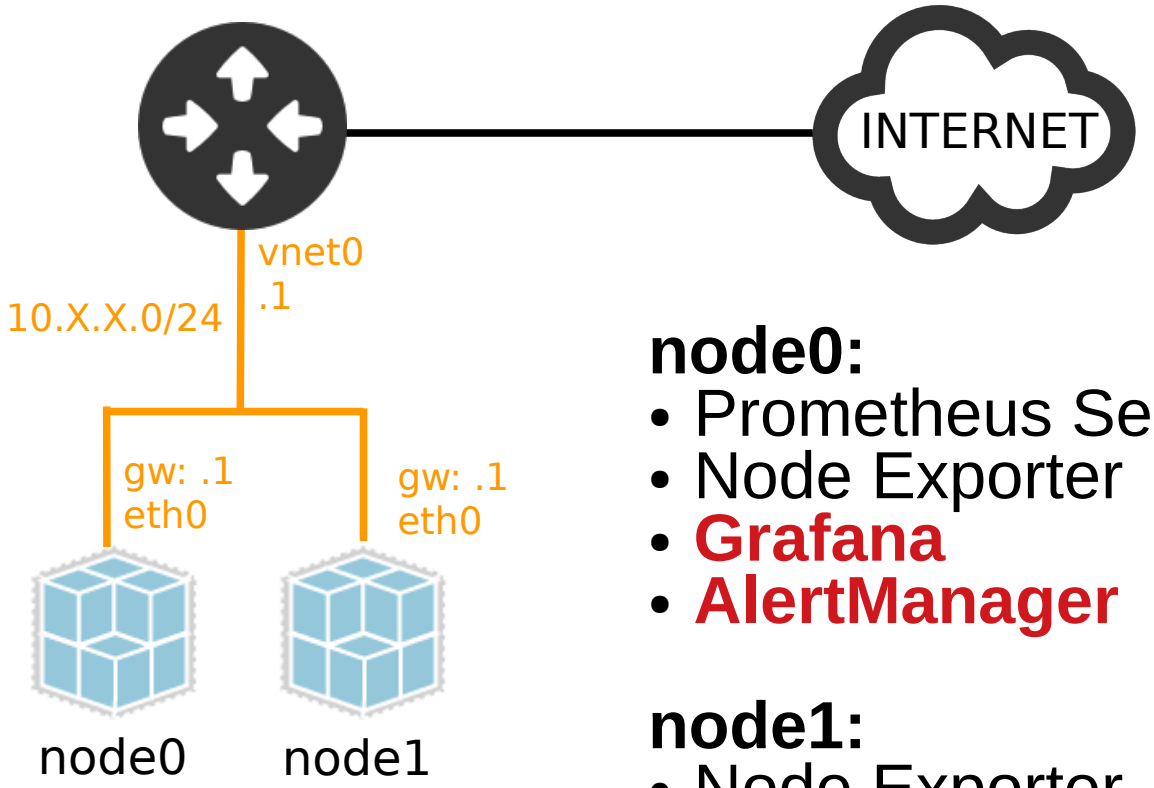
How AlertManager Works



Lab

Monitoring with Prometheus

Lab 2 Topology



node0:
- Prometheus Server
- Node Exporter
- **Grafana**
- **AlertManager**

node1:
- Node Exporter
- Docker Exporter

NolSatu.id