PROMETHEUS

MONITORING SISTEMA I SERVISA

Maša Nešić, 16774

MONITORING

ZA POTREBE MIKROSERVISA I INTERNETA STVARI I SERVISA

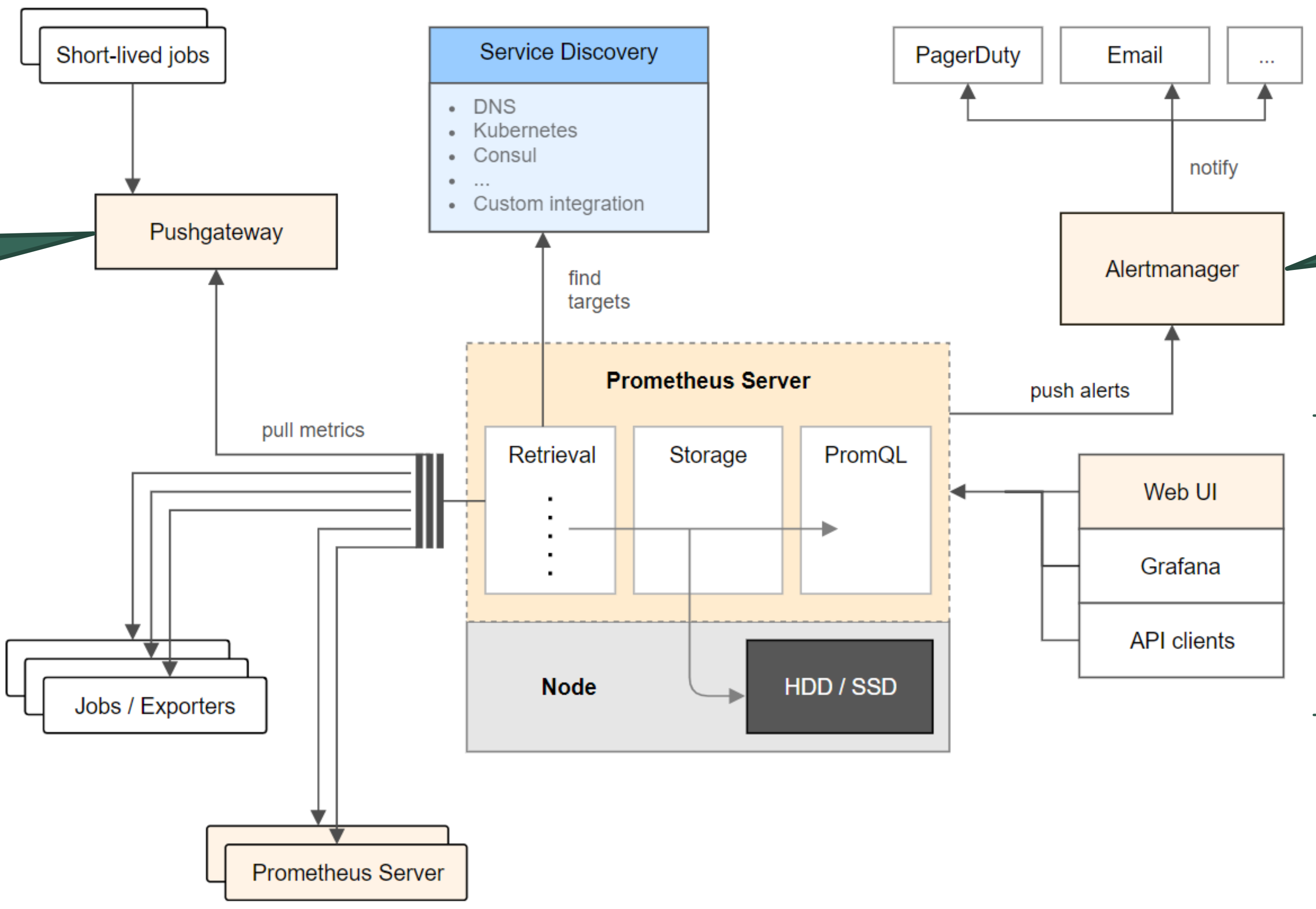
- Šta posmatramo
  - Metriku hardvera uređaja
  - Firmware
  - Kod aplikacije
  - Spoljašnje okruženje
  - Interakciju između uređaja i cloud-a
  - Pomoćne sisteme

- Zašto posmatramo?
  - Detektovanje, debagiranje i rešavanje problema koji se javljaju u sistemima, dok ti sistemi rade
  - Dugoročna analitika i izgradnja modela
  - Optimizacija rada sistema
  - Praćenje promena za potrebe donošenja poslovnih i tehničkih odluka
  - Za potrebe drugih sistema/procesa (QA, automatizacija, bezbednost, ...)

PROMETHEUS

- Sistem za monitoring sistema i servisa, i baza za smeštanje vremenskih serija podataka
- Open-source projekat
- Inicijalno razvijen od strane SoundCloud-a
- Zvanični projekat Cloud Native Computing Fondacije

ARHITEKTURA



Prometheus architecture

Za batch poslove koji se izvršavaju periodično, nisu stalno aktivni

Slanje „uzbuna“ povodom kritičnih događaja

Izvršavanje upita i vizuelizacija

GLAVNE KARAKTERISTIKE

- Multi-dimenzionalni model podataka - vremenska serija je difinisana imenom metrike, a labele (key-value parovi) dodaju dimenzije
- **PromQL** – moćan i fleksibilan jezik za pisanje upita koji može da iskoriti prednosti multi-dimenzionalnosti
- HTTP pull model za prikupljanje metrika
- Podrška za monitoring „batch“ poslova preko posrednika (**Pushgateway**)
- Podrška za vizuelizaciju – kreiranje grafova, tabela i dashboard-ova
- Efikasno skladištenje – lokalna on-disk baza (nezavisno od distribuiranih skladišta, autonomne instance servera), sa mogućnošću integracije sa udaljenim sistemima za skladištenje
- Precizno kreiranje alertova uz pomoć PromQL-a
- Klijentske biblioteke
- Veliki broj raspoloživih eksportera za integraciju third-party podataka sa Prometheus-om

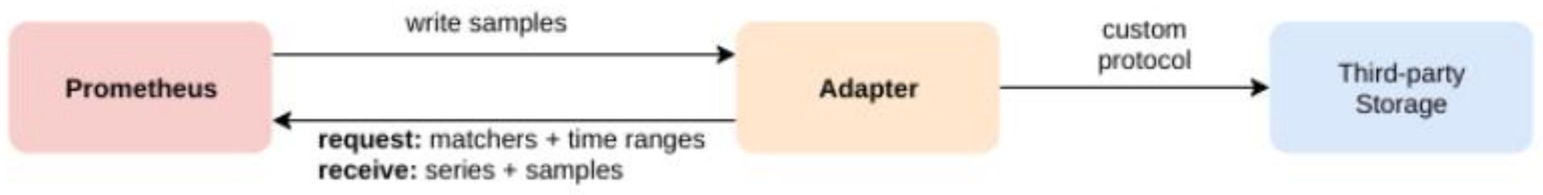
  https://prometheus.io/docs/instrumenting/exporters/

SKLADIŠTENJE

- Uzorci se grupišu u blokove materijla koji je skupljan u protekla 2h
- Svaki blok se sastoji od jednog ili više chunk fajlova koji sadrže vremenske serije podataka prikupljenih u tom periodu, fajl sa metapodacima i indeksni fajl (indeksira nazive metrika i labele za chunk fajlove)
- Kada se neka serija obriše preko API-ja, informacija o tome se čuva u zasebnom fajlu (ne vrši se odmah brisanje i preuređivanje podataka iz chunk fajlova)
- Trenutni blok, u koji se smeštaju podaci se čuva u memoriji i loguje u originalnom formatu u write-ahead fajl (kao siguronosni mehanizam u slučaju otkaza)
- Vremenom se vrši kompakcija 2h blokova u blokove koji sadrže veće količine infomacija - do 10% ukupnog vremena za koje se čuvaju podaci ili 31 dan (šta god je manje)
- Moguće je podešavati koliko dugo se čuvaju podaci – default je 15 dana
- Ukupan potreban prostor = vreme čuvanja podataka (u sekundama) * broj uzoraka koji se prikuplja u sekundi * broj B po uzorku (najčešće 1-2 B)

REMOTE SKLADIŠTENJE

- Korišćenjem lokalnog skladišta, svaka instanca je ograničena na sopstvenu pouzdanost i skalabilnost
- Kao rešenje tog problema, Prometheus nudi interfejse za integraciju sa remote sistemima za skladištenje
- Obezbeđuju se upis i čitanje sa remote URL u standardizovanom formatu
- Komunikacija sa adapterom se obavlja u kompresovanom formatu pomoću protokol bafera, preko HTTP
- Predviđa se prelazak na gRPC, kada se obezbedi da sve instance Prometheus ekosistema mogu da rade preko HTTP/2



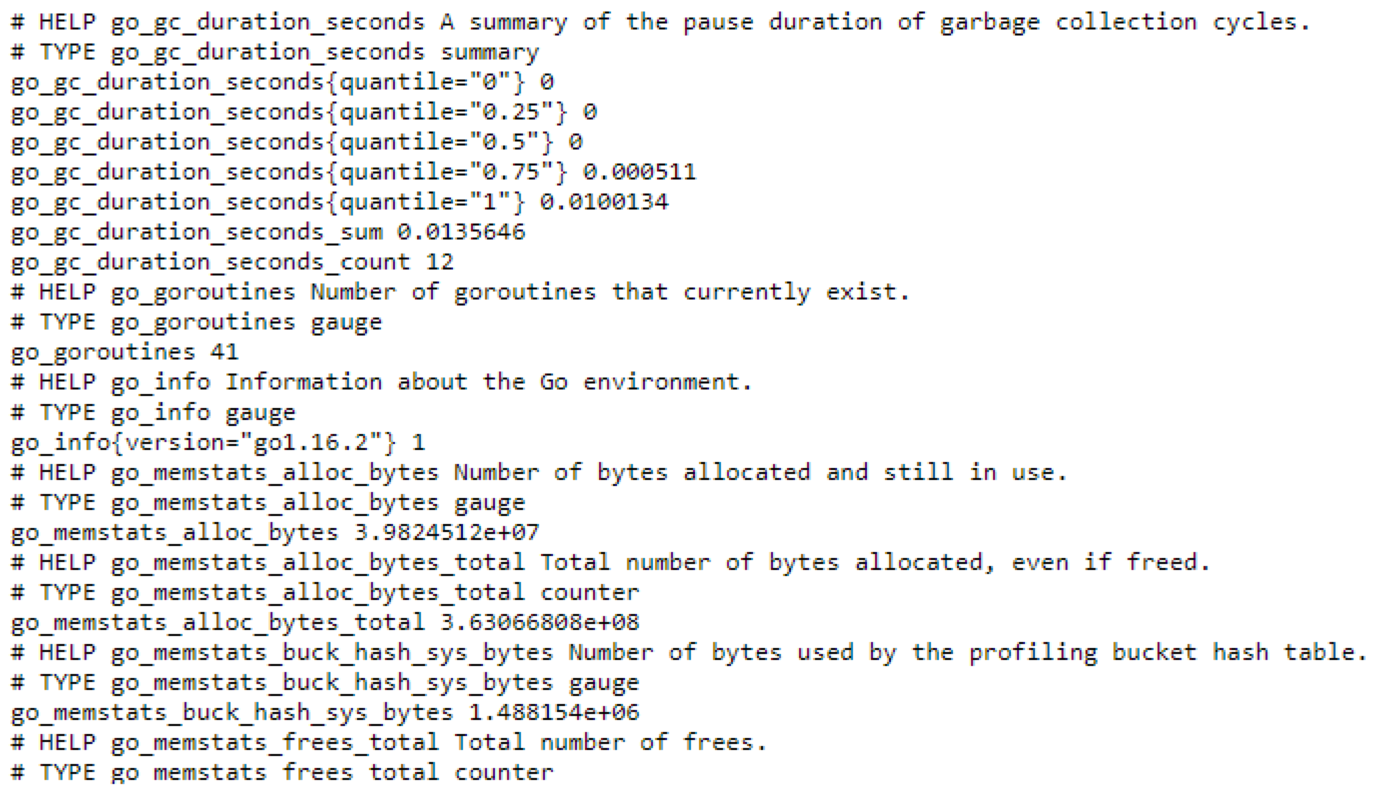
https://prometheus.io/docs/prometheus/latest/storage/

SELF MONITORING

- prometheus.yml

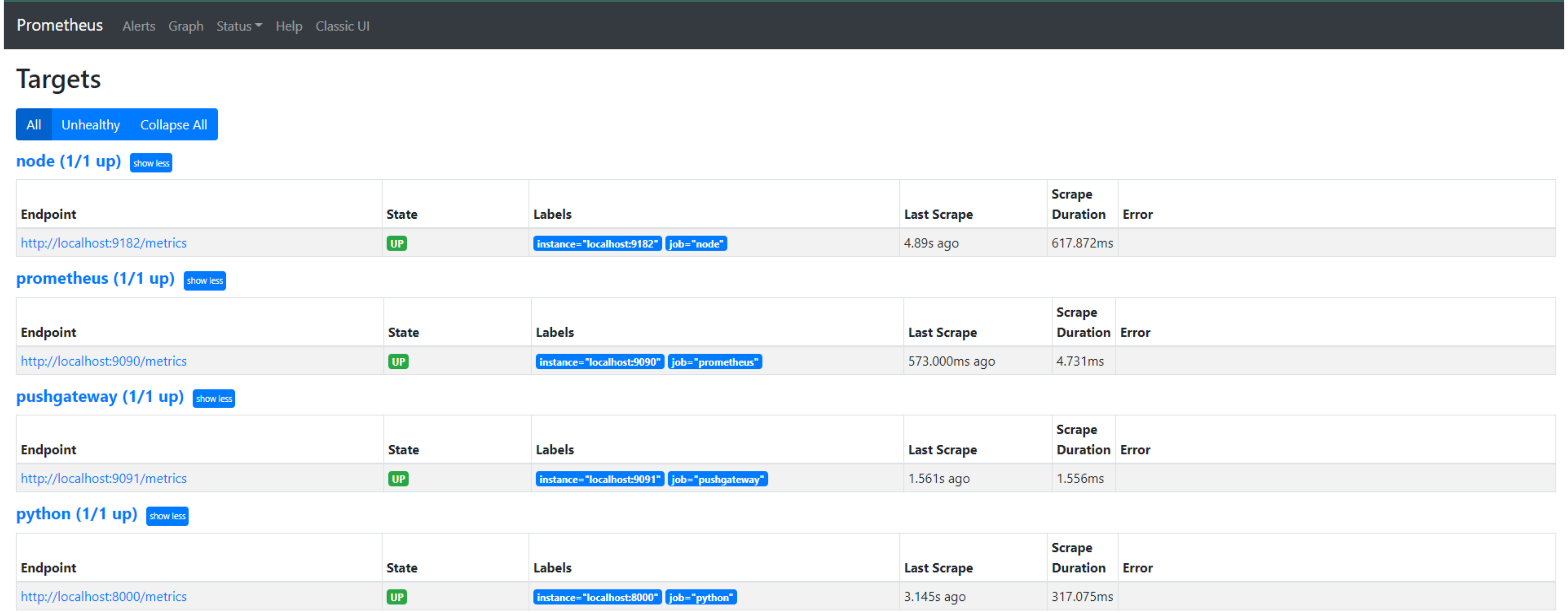
```
global:
  scrape_interval:     15s
  evaluation_interval: 15s
scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    static_configs:
    - targets: ['localhost:9090’]
```

- ./prometheus.exe --config.file=prometheus.yml



```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0.000511
go_gc_duration_seconds{quantile="1"} 0.0100134
go_gc_duration_seconds_sum 0.0135646
go_gc_duration_seconds_count 12
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 41
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.16.2"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 3.9824512e+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 3.63066808e+08
# 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 1.488154e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
```

TARGETS



Targets

node (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9182/metrics	UP	instance="localhost:9182" job="node"	4.89s ago	617.872ms	

prometheus (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	573.000ms ago	4.731ms	

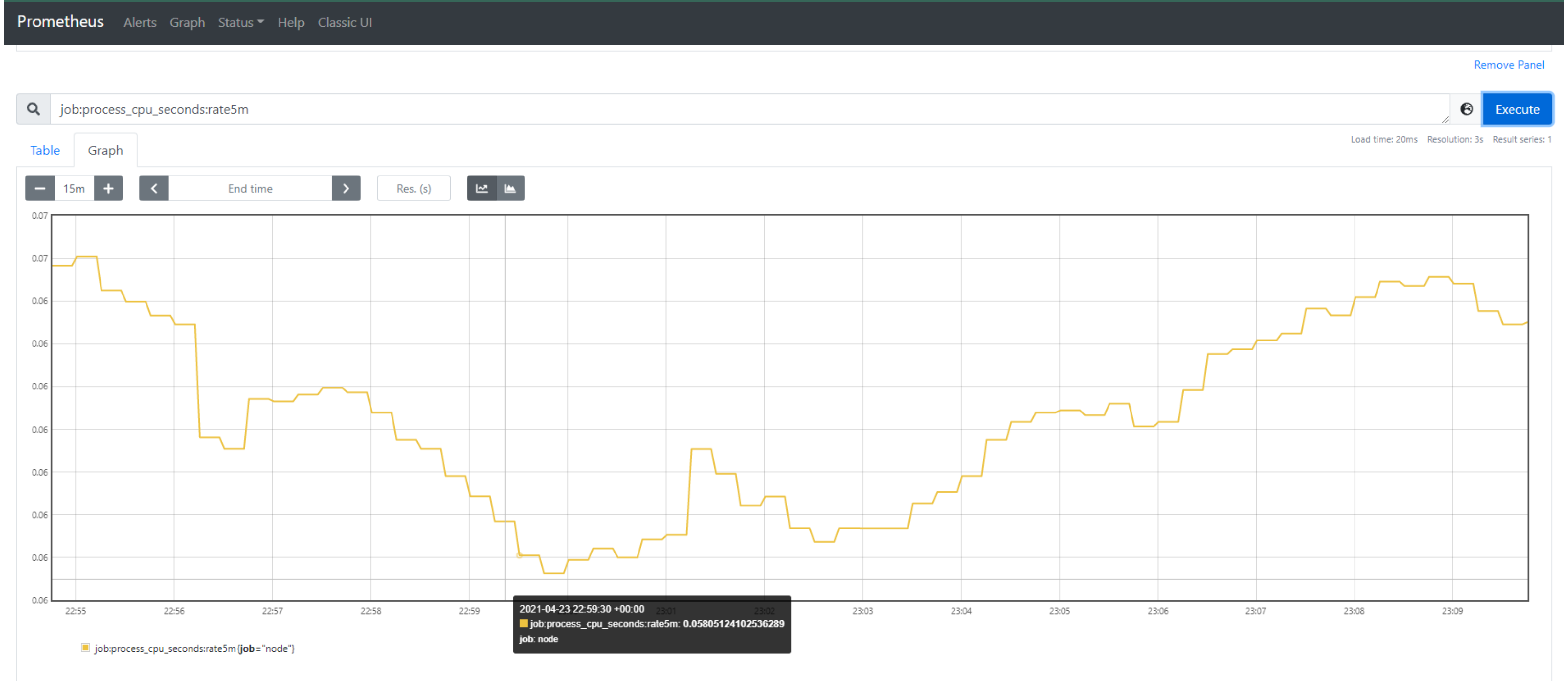
pushgateway (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9091/metrics	UP	instance="localhost:9091" job="pushgateway"	1.561s ago	1.556ms	

python (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:8000/metrics	UP	instance="localhost:8000" job="python"	3.145s ago	317.075ms	

QUERIES



job:process_cpu_seconds:rate5m

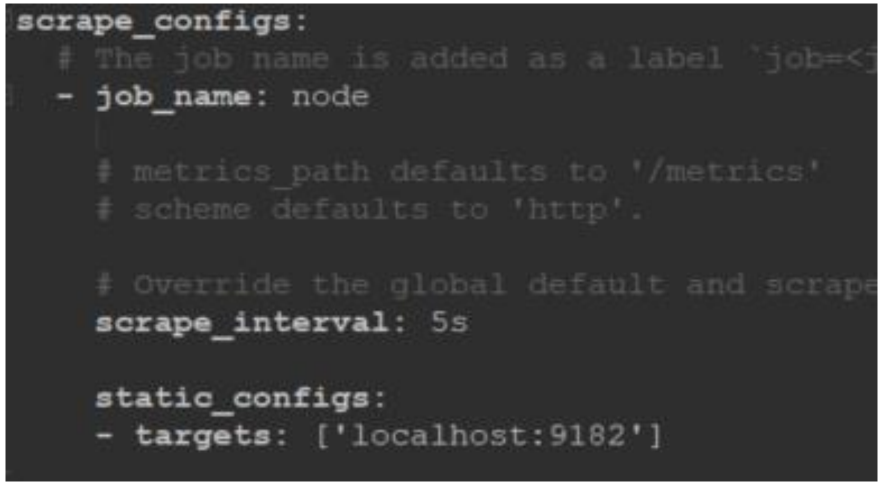
Load time: 20ms Resolution: 3s Result series: 1

2021-04-23 22:59:30 +00:00
job:process_cpu_seconds:rate5m: **0.05805124102536289**
job: node

🟨 job:process_cpu_seconds:rate5m{**job**="node"}

THIRD PARTY MONITORING

- Eksporteri
- Windows exporter
  https://github.com/prometheus-community/windows_exporter
- Metrike vezane za hardver i OS mašine
  - Iskorišćenost CPU, memorije, prostora na disku, upisi i čitanje sa diska, bandwidth mreže, trenutni broj aktivnih niti, ...



```
scrape_configs:
  # The job name is added as a label `job=<j
  - job_name: node

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    # Override the global default and scrape
    scrape_interval: 5s

    static_configs:
    - targets: ['localhost:9182']
```

SERVICE DISCOVERY

- Statički – znamo lokaciju izvora i ona je nepromenjiva

```
scrape_configs:
  - job_name: ‘node’
    static_configs:
    - targets: [‘localhost:9182’]
```

- Iz fajla – za service discovery sisteme koje Prometheus ne podržava out of the box

```
scrape_configs:
  - job_name: file
    file_sd_configs:
      - files:  '*.json'
```

```
[
  {
    "targets": [ "host1:9100", "host2:9100" ],
    "labels": {
      "team": "infra",
      "job": "node"
    }
  },
  {
    "targets": [ "host1:9090" ],
    "labels": {
      "team": "monitoring",
      "job": "prometheus"
    }
  }
]
```

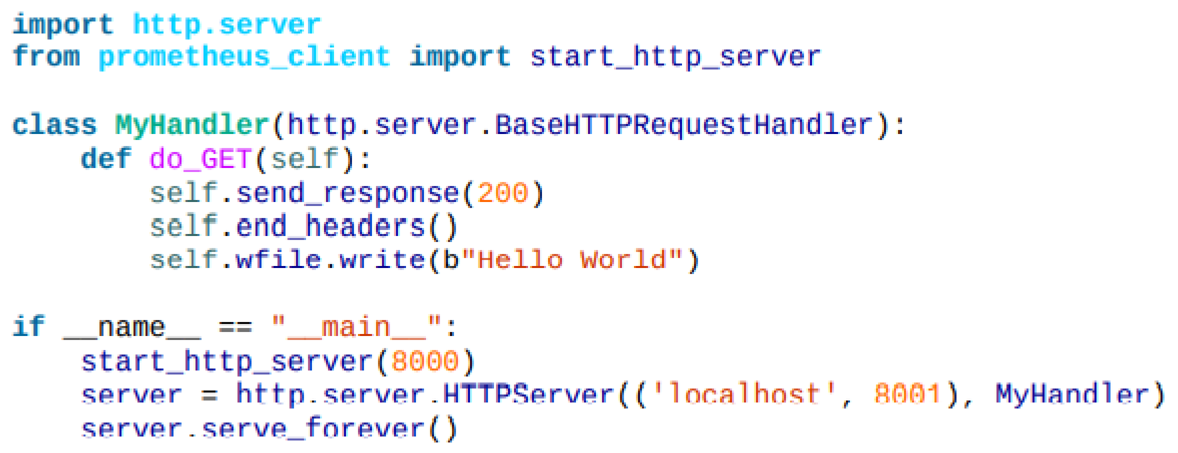
- Out of the box podrška za popularna SD rešenja
  - Azure, Consul, DNS, EC2, OpenStack, File, Kubernetes, Marathon, Nerve, Serverset, and Triton
  - Kubernetes

```
scrape_configs:
 - job_name: 'k8services'
   kubernetes_sd_configs:
    - role: endpoints
   relabel_configs:
    - source_labels:
       - __meta_kubernetes_namespace
       - __meta_kubernetes_service_name
      regex: default;kubernetes
      action: drop
    - source_labels:
       - __meta_kubernetes_namespace
      regex: default
      action: keep
    - source_labels: [__meta_kubernetes_service_name]
      target_label: job
```

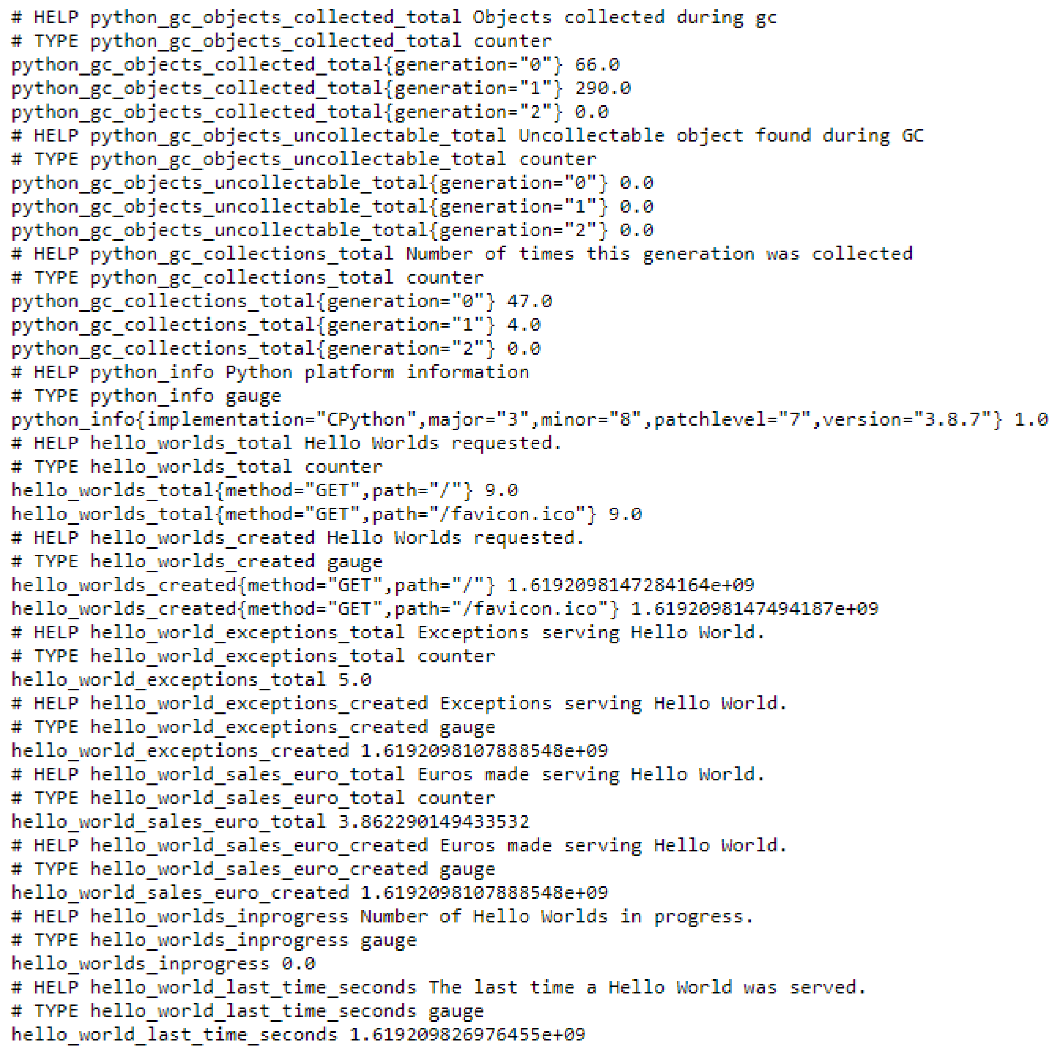
node, endpoints, service, pod, ingress

METRIKE - INSTRUMENTACIJA

- Klijentske biblioteke – Go, Python, Java i Ruby
- Python - prometheus_client



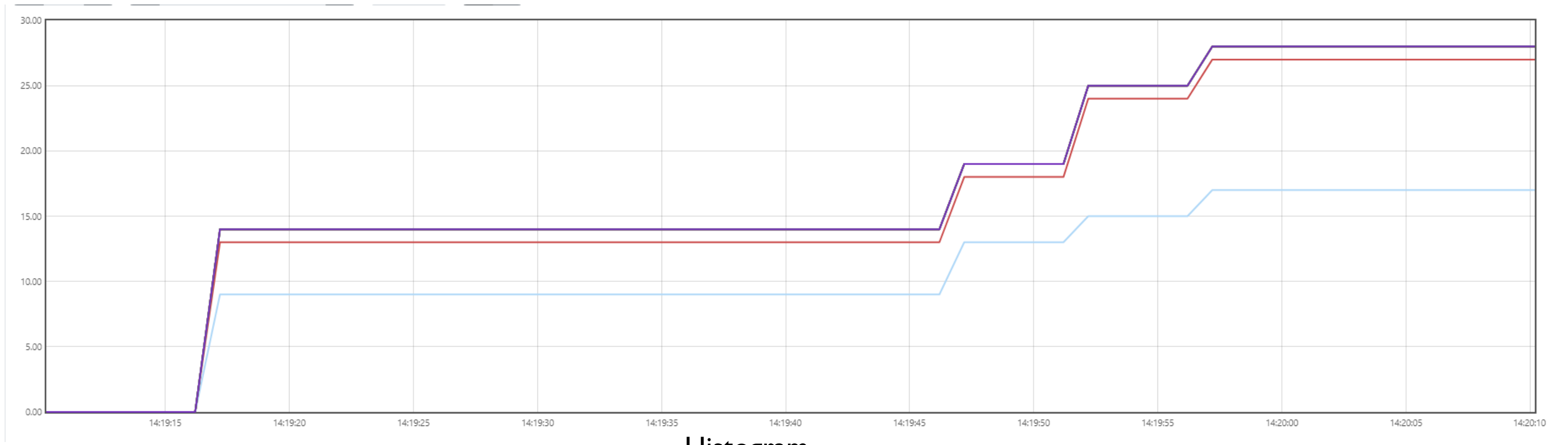
```python
import http.server
from prometheus_client import start_http_server

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")

if __name__ == "__main__":
    start_http_server(8000)
    server = http.server.HTTPServer(('localhost', 8001), MyHandler)
    server.serve_forever()
```



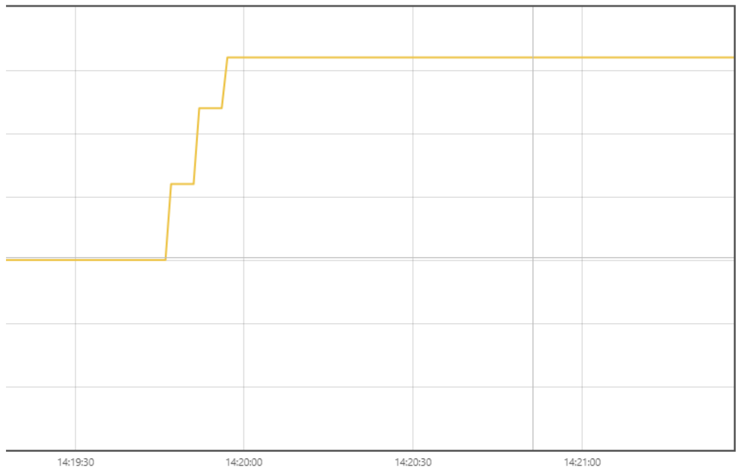
```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 66.0
python_gc_objects_collected_total{generation="1"} 290.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable object found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 47.0
python_gc_collections_total{generation="1"} 4.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="8",patchlevel="7",version="3.8.7"} 1.0
# HELP hello_worlds_total Hello Worlds requested.
# TYPE hello_worlds_total counter
hello_worlds_total{method="GET",path="/"} 9.0
hello_worlds_total{method="GET",path="/favicon.ico"} 9.0
# HELP hello_worlds_created Hello Worlds requested.
# TYPE hello_worlds_created gauge
hello_worlds_created{method="GET",path="/"} 1.6192098147284164e+09
hello_worlds_created{method="GET",path="/favicon.ico"} 1.6192098147494187e+09
# HELP hello_world_exceptions_total Exceptions serving Hello World.
# TYPE hello_world_exceptions_total counter
hello_world_exceptions_total 5.0
# HELP hello_world_exceptions_created Exceptions serving Hello World.
# TYPE hello_world_exceptions_created gauge
hello_world_exceptions_created 1.6192098107888548e+09
# HELP hello_world_sales_euro_total Euros made serving Hello World.
# TYPE hello_world_sales_euro_total counter
hello_world_sales_euro_total 3.862290149433532
# HELP hello_world_sales_euro_created Euros made serving Hello World.
# TYPE hello_world_sales_euro_created gauge
hello_world_sales_euro_created 1.6192098107888548e+09
# HELP hello_worlds_inprogress Number of Hello Worlds in progress.
# TYPE hello_worlds_inprogress gauge
hello_worlds_inprogress 0.0
# HELP hello_world_last_time_seconds The last time a Hello World was served.
# TYPE hello_world_last_time_seconds gauge
hello_world_last_time_seconds 1.619209826976455e+09
```

METRIKE - INSTRUMENTACIJA

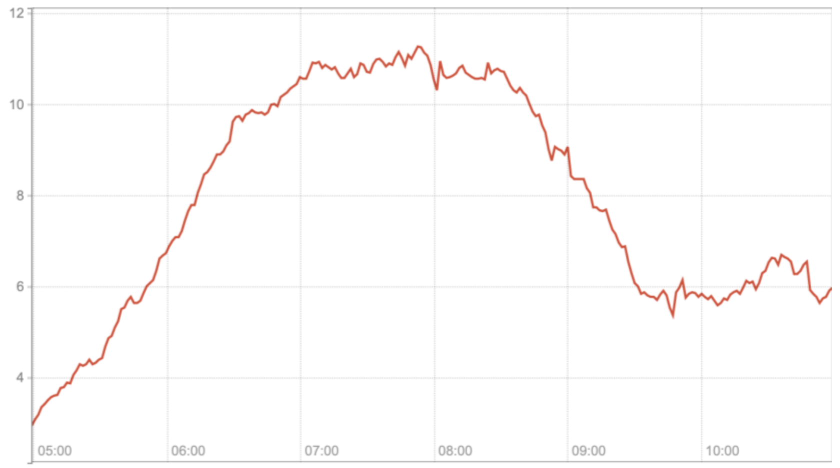
- Counter
- Gauge
- Summary
- Histogram



Histogram



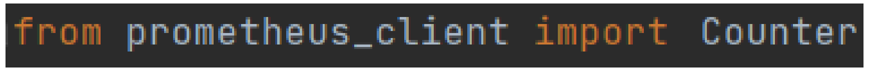
Counter



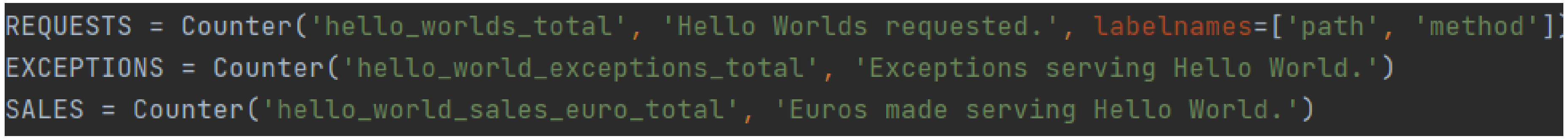
Gauge

COUNTER

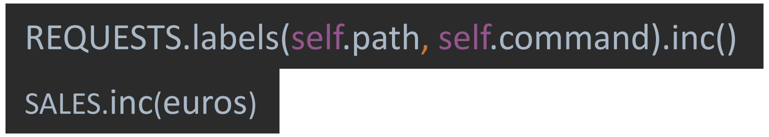
- Najčešće se koriste da bi merili broj izvršenja nekog dela koda



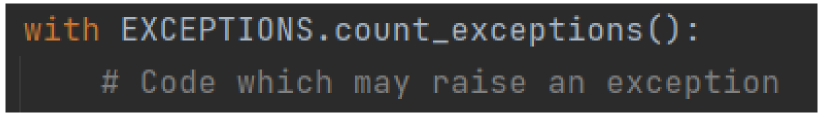
```python
from prometheus_client import  Counter
```



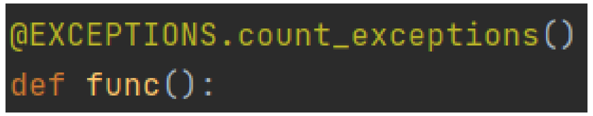
```python
REQUESTS = Counter('hello_worlds_total', 'Hello Worlds requested.', labelnames=['path', 'method']
EXCEPTIONS = Counter('hello_world_exceptions_total', 'Exceptions serving Hello World.')
SALES = Counter('hello_world_sales_euro_total', 'Euros made serving Hello World.')
```



```python
REQUESTS.labels(self.path, self.command).inc()
SALES.inc(euros)
```



```python
with EXCEPTIONS.count_exceptions():
    # Code which may raise an exception
```



```python
@EXCEPTIONS.count_exceptions()
def func():
```

- PromQL

rate(hello_world_exceptions_total[1m])

rate(hello_worlds_total[1m])

rate(hello_world_exceptions_total[1m]) /

sum without(method, path)(rate(hello_worlds_total[1m]))

GAUGE

- Trenutno stanje nekog resursa
- Npr. Broj aktivnih niti, zauzeće keš memorije, poslednji put kada je neki zahtev obrađen, ...
- Inc, dec i set

```python
import time
from prometheus_client import Gauge

INPROGRESS = Gauge('hello_worlds_inprogress',
        'Number of Hello Worlds in progress.')
LAST = Gauge('hello_world_last_time_seconds',
        'The last time a Hello World was served.')

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        INPROGRESS.inc()
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")
        LAST.set(time.time())
        INPROGRESS.dec()
```

$\rightarrow$ `LAST.set_to_current_time()`

time() - hello_world_last_time_seconds

SUMMARY

- Pogodno za praćenje latencije
- Observe()
- Generiše 2 vremenske serije
  - _count – broj poziva observe funkcije
  - _sum – suma vrednosti prosleđenih observe funkciji

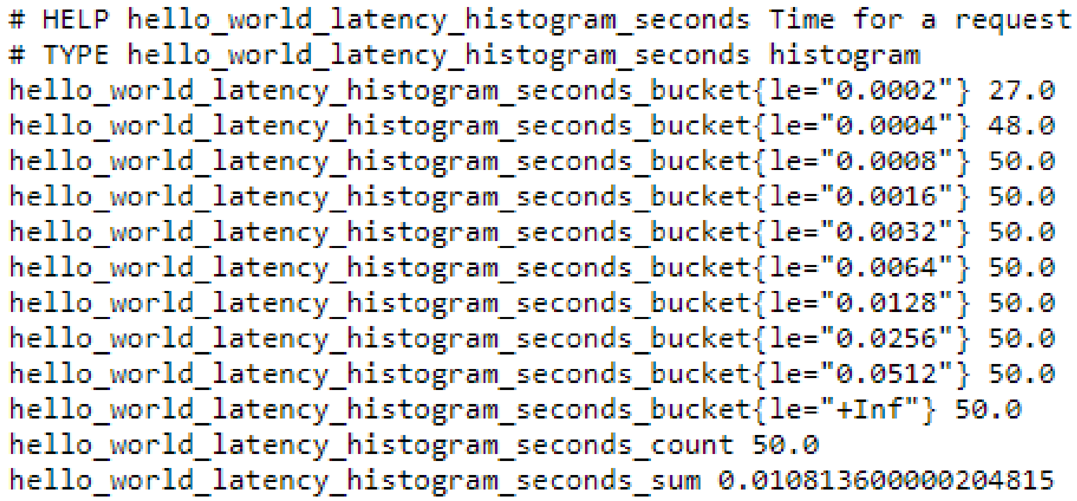
rate(hello_world_latency_seconds_sum[1m]) /
rate(hello_world_latency_seconds_count[1m])

```python
import time
from prometheus_client import Summary

LATENCY = Summary('hello_world_latency_seconds',
        'Time for a request Hello World.')

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        start = time.time()
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")
        LATENCY.observe(time.time() - start)
```

```python
from prometheus_client import Summary

LATENCY = Summary('hello_world_latency_seconds',
        'Time for a request Hello World.')

class MyHandler(http.server.BaseHTTPRequestHandler):
    @LATENCY.time()
    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World")
```

HISTOGRAM

- Quantiles and percentiles
- Observe() funkcija, isto kao za Summary
- Kreira _count i _sum vremenske serije (kao Summary) i niz vremenskih serija _bucket, gde je svaka od njih counter događaja koji upadaju u njen opseg



```
# HELP hello_world_latency_histogram_seconds Time for a request Hello World.
# TYPE hello_world_latency_histogram_seconds histogram
hello_world_latency_histogram_seconds_bucket{le="0.0002"} 27.0
hello_world_latency_histogram_seconds_bucket{le="0.0004"} 48.0
hello_world_latency_histogram_seconds_bucket{le="0.0008"} 50.0
hello_world_latency_histogram_seconds_bucket{le="0.0016"} 50.0
hello_world_latency_histogram_seconds_bucket{le="0.0032"} 50.0
hello_world_latency_histogram_seconds_bucket{le="0.0064"} 50.0
hello_world_latency_histogram_seconds_bucket{le="0.0128"} 50.0
hello_world_latency_histogram_seconds_bucket{le="0.0256"} 50.0
hello_world_latency_histogram_seconds_bucket{le="0.0512"} 50.0
hello_world_latency_histogram_seconds_bucket{le="+Inf"} 50.0
hello_world_latency_histogram_seconds_count 50.0
hello_world_latency_histogram_seconds_sum 0.010813600000204815
```

```
LATENCYH = Histogram('hello_world_latency_histogram_seconds’,
                     'Time for a request Hello World.',
                      buckets=[0.0001 * 2**x for x in range(1, 10)])
```

histogram_quantile(0.95, rate(hello_world_latency_histogram_seconds_bucket[1m]))

LABELE

- Daju dimenzionalnosti podacima
- Key/value parovi
- Vrednost labela i ime metrike zajedno jedinstveno identifikuju vremensku seriju podataka

```
http_requests_login_total
http_requests_logout_total
http_requests_adduser_total
http_requests_comment_total
http_requests_view_total
```

vs

```
http_requests_total{path="/login"}
http_requests_total{path="/logout"}
http_requests_total{path="/adduser"}
http_requests_total{path="/comment"}
http_requests_total{path="/view"}
```

- Target labele – identifikuju metu od koje Prometheus pribavlja podatke
  - Arhitektura, id aplikacije, id datacentra, production/development, tim, instanca
  - Prometheus ih dodaje u procesu prikupljanja podataka
- Labele za instrumentaciju – potiču iz procesa instrumentacije
  - Informacije koje su poznate unutar aplikacije/biblioteke – tip HTTP zahteva, baza kojoj se obraća
- PromQL ne pravi razliku između toga kog je tipa labela

LABEL PATTERNS

- Prometheus može da pamti samo 64-bitne float vrednosti, ne i stringove
- Labele su stringovi

Enum

- Stanje resursa – STARTING, RUNNING, STOPING, TERMINATED
- Kodiranje brojevima nije pogodno za izračunavanja

```
# HELP gauge The current state of resources.
# TYPE gauge resource_state
resource_state{resource_state="STARTING",resource="blaa"} 0
resource_state{resource_state="RUNNING",resource="blaa"} 1
resource_state{resource_state="STOPPING",resource="blaa"} 0
resource_state{resource_state="TERMINATED",resource="blaa"} 0
```

avg_over_time(resource_state[1h])

without(resource)(resource_state)

```python
from threading import Lock
from prometheus_client.core import GaugeMetricFamily, REGISTRY

class StateMetric(object):
    def __init__(self):
        self._resource_states = {}
        self._STATES = ["STARTING", "RUNNING", "STOPPING", "TERMINATED",]
        self._mutex = Lock()

    def set_state(self, resource, state):
        with self._mutex:
            self._resource_states[resource] = state

    def collect(self):
        family = GaugeMetricFamily("resource_state", "The current state of resources.",
                                   labels=["resource_state", "resource"])
        with self._mutex:
            for resource, state in self._resource_states.items():
                for s in self._STATES:
                    family.add_metric([s, resource], 1 if s == state else 0)
        yield family
```

LABEL PATTERNS

Info

- Za informacije koje nisu pogodne da budu target labels
- Gauge sa vrednošću 1 i labelama za sve željene informacije

```
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="5",patchlevel="2",
        version="3.5.2"} 1.0
```

```
up
* on (instance, job) group_left(version)
my_python_info
```

```
version_info = {
  "implementation": "CPython",
  "major": "3",
  "minor": "5",
  "patchlevel": "2",
  "version": "3.5.2"
}
INFO = Gauge("my_python_info",
             "Python platform information",
             labelnames=version_info.keys())

INFO.labels(**version_info).set(1)
```

NAZIVI METRIKA

DOBRA PRAKSA

- self-explanatory
- Hijerarhijski model imena – od najopštijeg ka specifičnijim opisima
- Prva reč – domen, namespace
- Osnovne jedinice
- Jedinica na kraju imena metrike
- 1 metrika = 1 značenje, labele određuju kontekst
- Svaka jedinstvena kombinacija vrednosti labela => zasebna vremenska serija
- Nazivi counter-a treba da se završavaju sa _total
- Izbegavati sufikse _count, _sum i _bucket jer se oni već koriste kod histograma i summary

$$\overbrace{\text{windows}}^{\text{namespace}}\text{\_}\underbrace{\text{cpu\_cstate}}_{\text{hijerarhija}}\text{\_}\overbrace{\text{seconds}}^{\text{jedinica}}\text{\_}\underbrace{\text{total}}_{\text{counter}}\overbrace{\{\text{core="0,0",state="c1"}\}}^{\text{labele}}$$

UNIT TESTIRANJE INSTRUMENTACIJE

- Za testiranje nekih kritičnih metrika
- REGISTRY omogućava pristup raspoloživim podacima kroz kod
- get_sample_value funkcija

```python
import unittest
from prometheus_client import Counter, REGISTRY

FOOS = Counter('foos_total', 'The number of foo calls.')

def foo():
    FOOS.inc()

class TestFoo(unittest.TestCase):
    def test_counter_inc(self):
        before = REGISTRY.get_sample_value('foos_total')
        foo()
        after = REGISTRY.get_sample_value('foos_total')
        self.assertEqual(1, after - before)
```

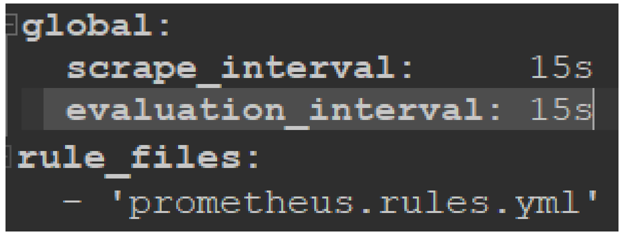
INSTRUMENTACIJA

- Online-serving sistemi – sinhroni interaktivni sistemi
  - Requests
  - Errors
  - Duration – latencija
- Offline-serving sistemi – pipeline za obradu podataka
  - Utilisation – zasićenost servisa
  - Saturation – količina posla na čekanju
  - Errors

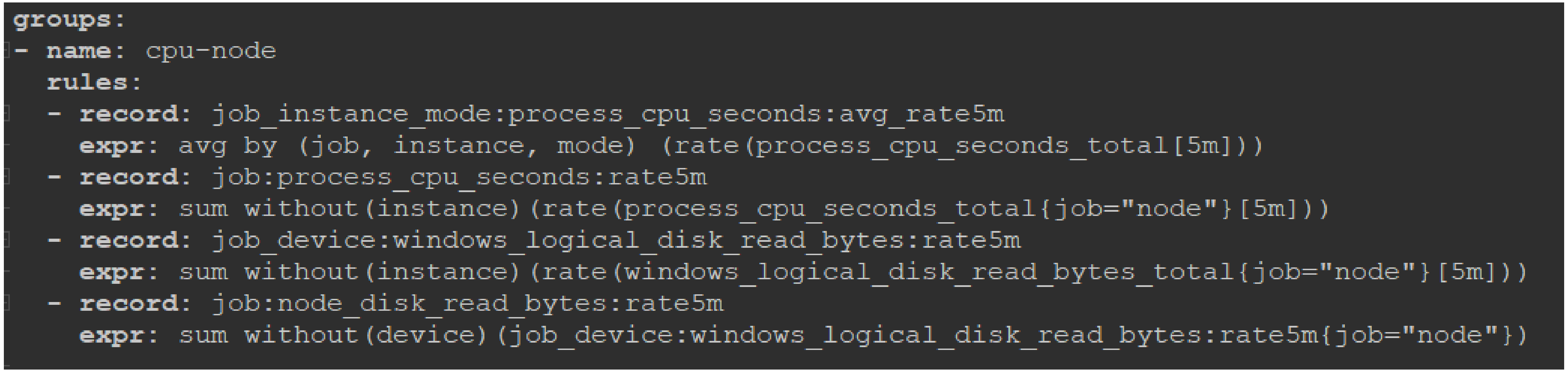
- 10 miliona metrika i 1000 čvorova
- 1 nova metrika za svaku instancu je 0.01% resursa
- Histrogrami uvode visoku kardinalnost
  - Za kardinalnost 100 – 1 nova metrika je 1% resursa
- Rule of thumb – 10 najvećih metrika zauzima polovinu resursa
  - Fokus na njihovoj optimizaciji

PRAVILA

- PromQL izrazi koji se rodovno evaluiraju
- Agregirani rezultati za kasnije korišćenje – ubrzava rad dashboard-ova
- Mala periodična obrada vs obrada velike količine podataka u trenutku izvršenja upita

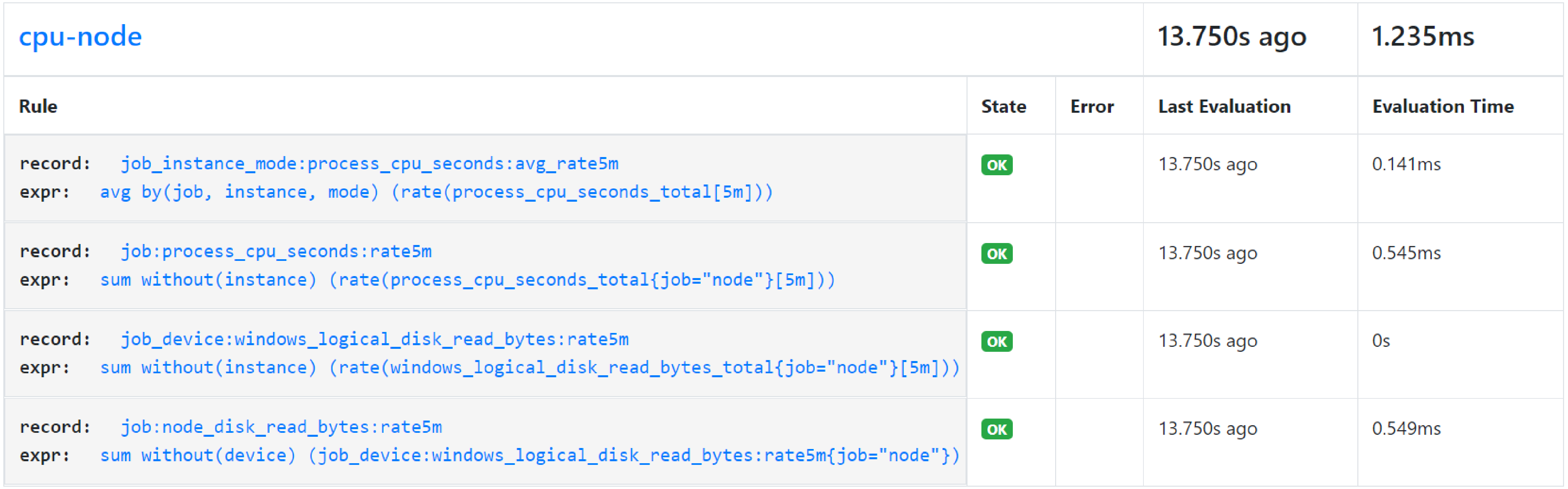


```yaml
global:
  scrape_interval:     15s
  evaluation_interval: 15s
rule_files:
  - 'prometheus.rules.yml'
```



```yaml
groups:
- name: cpu-node
  rules:
  - record: job_instance_mode:process_cpu_seconds:avg_rate5m
    expr: avg by (job, instance, mode) (rate(process_cpu_seconds_total[5m]))
  - record: job:process_cpu_seconds:rate5m
    expr: sum without(instance)(rate(process_cpu_seconds_total{job="node"}[5m]))
  - record: job_device:windows_logical_disk_read_bytes:rate5m
    expr: sum without(instance)(rate(windows_logical_disk_read_bytes_total{job="node"}[5m]))
  - record: job:node_disk_read_bytes:rate5m
    expr: sum without(device)(job_device:windows_logical_disk_read_bytes:rate5m{job="node"})
```

PRAVILA



cpu-node			13.750s ago	1.235ms
Rule	**State**	**Error**	**Last Evaluation**	**Evaluation Time**

`record: job_instance_mode:process_cpu_seconds:avg_rate5m`
`expr: avg by(job, instance, mode) (rate(process_cpu_seconds_total[5m]))`
	OK		13.750s ago	0.141ms

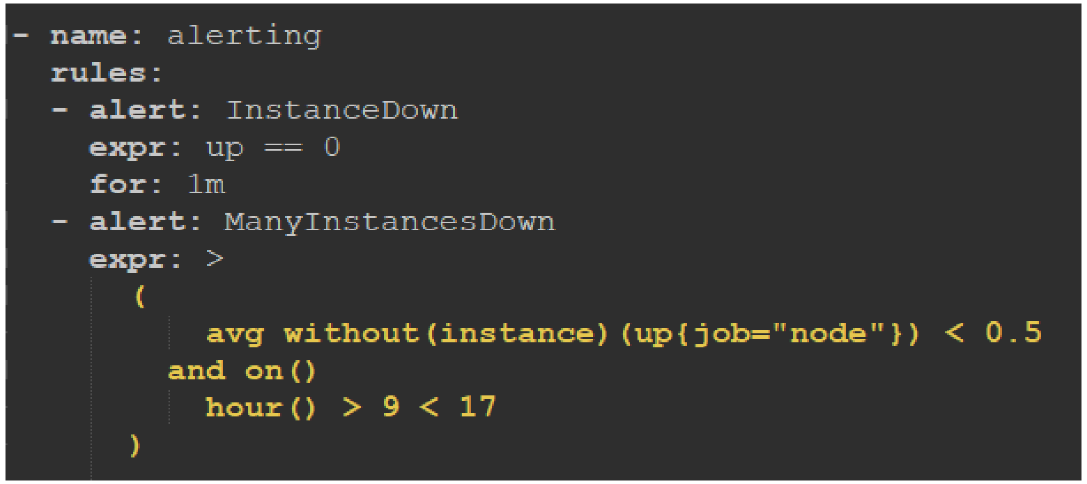
`record: job:process_cpu_seconds:rate5m`
`expr: sum without(instance) (rate(process_cpu_seconds_total{job="node"}[5m]))`
	OK		13.750s ago	0.545ms

`record: job_device:windows_logical_disk_read_bytes:rate5m`
`expr: sum without(instance) (rate(windows_logical_disk_read_bytes_total{job="node"}[5m]))`
	OK		13.750s ago	0s

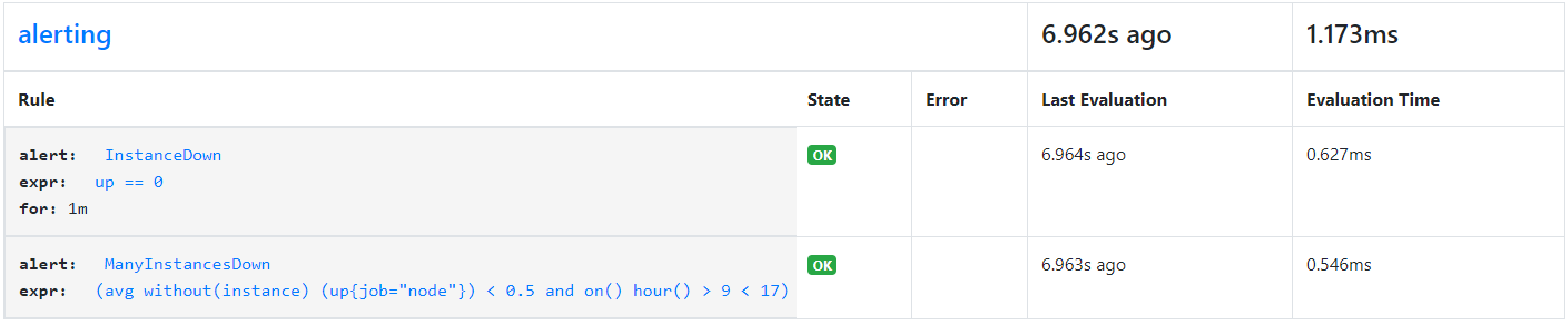
`record: job:node_disk_read_bytes:rate5m`
`expr: sum without(device) (job_device:windows_logical_disk_read_bytes:rate5m{job="node"})`
	OK		13.750s ago	0.549ms

ALERTS

- Specijalna pravila
- Pišu se u istom fajlu kao i pravila
- Umesto record, alert

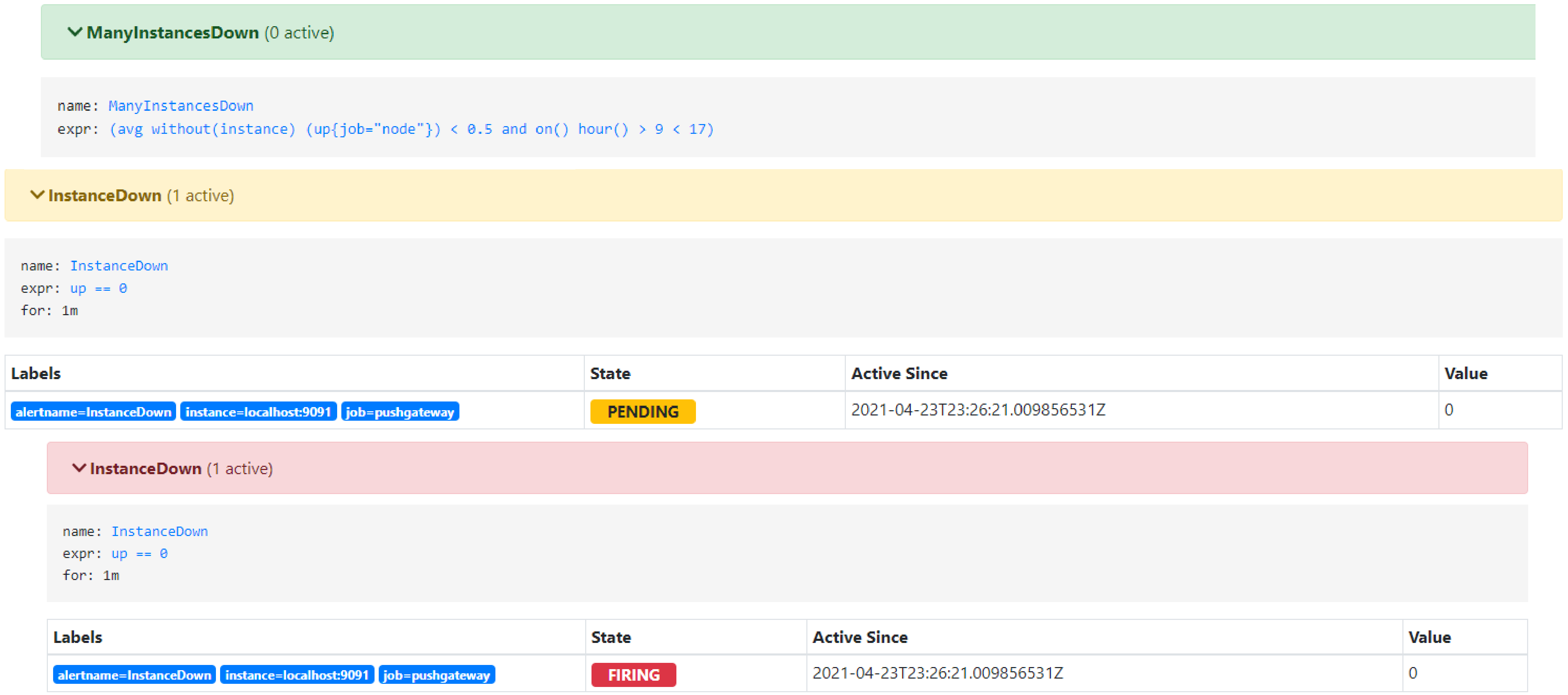


```
- name: alerting
  rules:
  - alert: InstanceDown
    expr: up == 0
    for: 1m
  - alert: ManyInstancesDown
    expr: >
      (
        avg without(instance)(up{job="node"}) < 0.5
      and on()
        hour() > 9 < 17
      )
```



alerting			6.962s ago	1.173ms
Rule	**State**	**Error**	**Last Evaluation**	**Evaluation Time**
alert: InstanceDown expr: up == 0 for: 1m	OK		6.964s ago	0.627ms
alert: ManyInstancesDown expr: (avg without(instance) (up{job="node"}) < 0.5 and on() hour() > 9 < 17)	OK		6.963s ago	0.546ms

ALERTS



ManyInstancesDown (0 active)

```
name: ManyInstancesDown
expr: (avg without(instance) (up{job="node"}) < 0.5 and on() hour() > 9 < 17)
```

InstanceDown (1 active)

```
name: InstanceDown
expr: up == 0
for: 1m
```

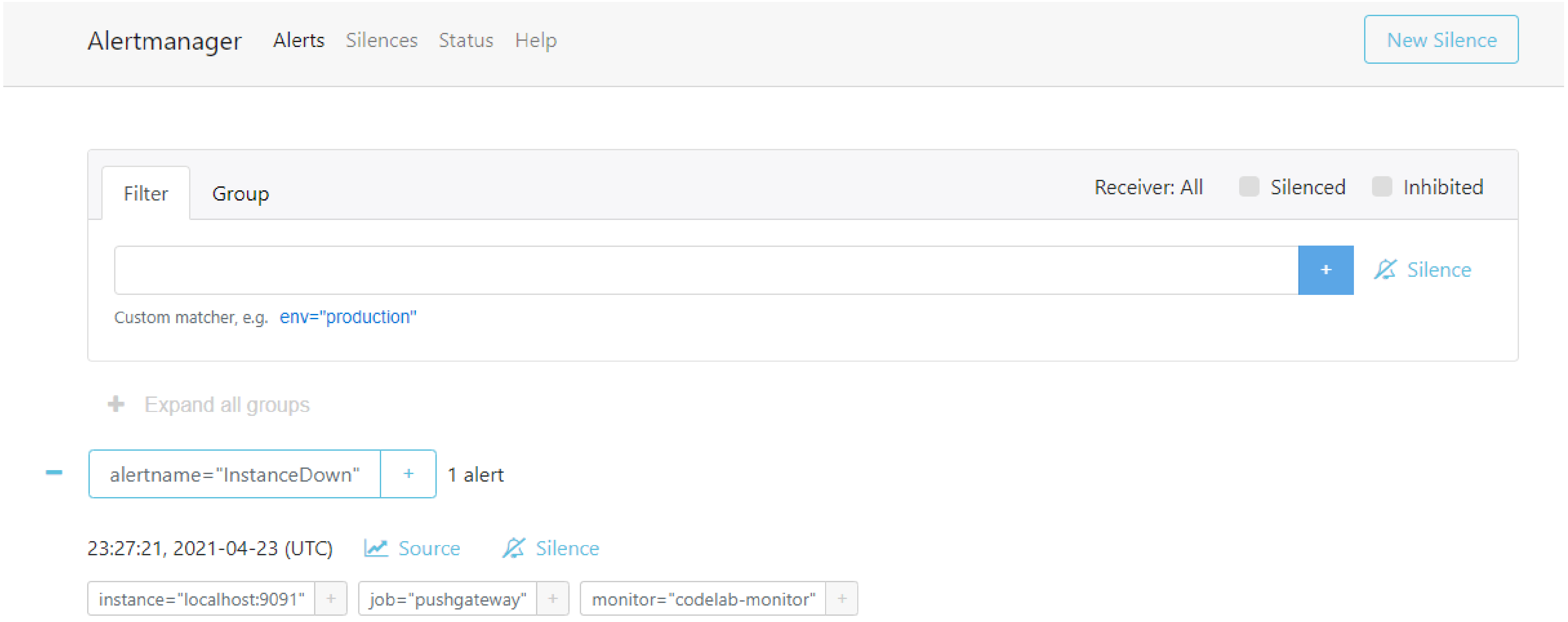
Labels	State	Active Since	Value
alertname=InstanceDown instance=localhost:9091 job=pushgateway	PENDING	2021-04-23T23:26:21.009856531Z	0

InstanceDown (1 active)

```
name: InstanceDown
expr: up == 0
for: 1m
```

Labels	State	Active Since	Value
alertname=InstanceDown instance=localhost:9091 job=pushgateway	FIRING	2021-04-23T23:26:21.009856531Z	0

ALERTMANAGER



Custom matcher, e.g. env="production"

alertname="InstanceDown" + 1 alert

23:27:21, 2021-04-23 (UTC)

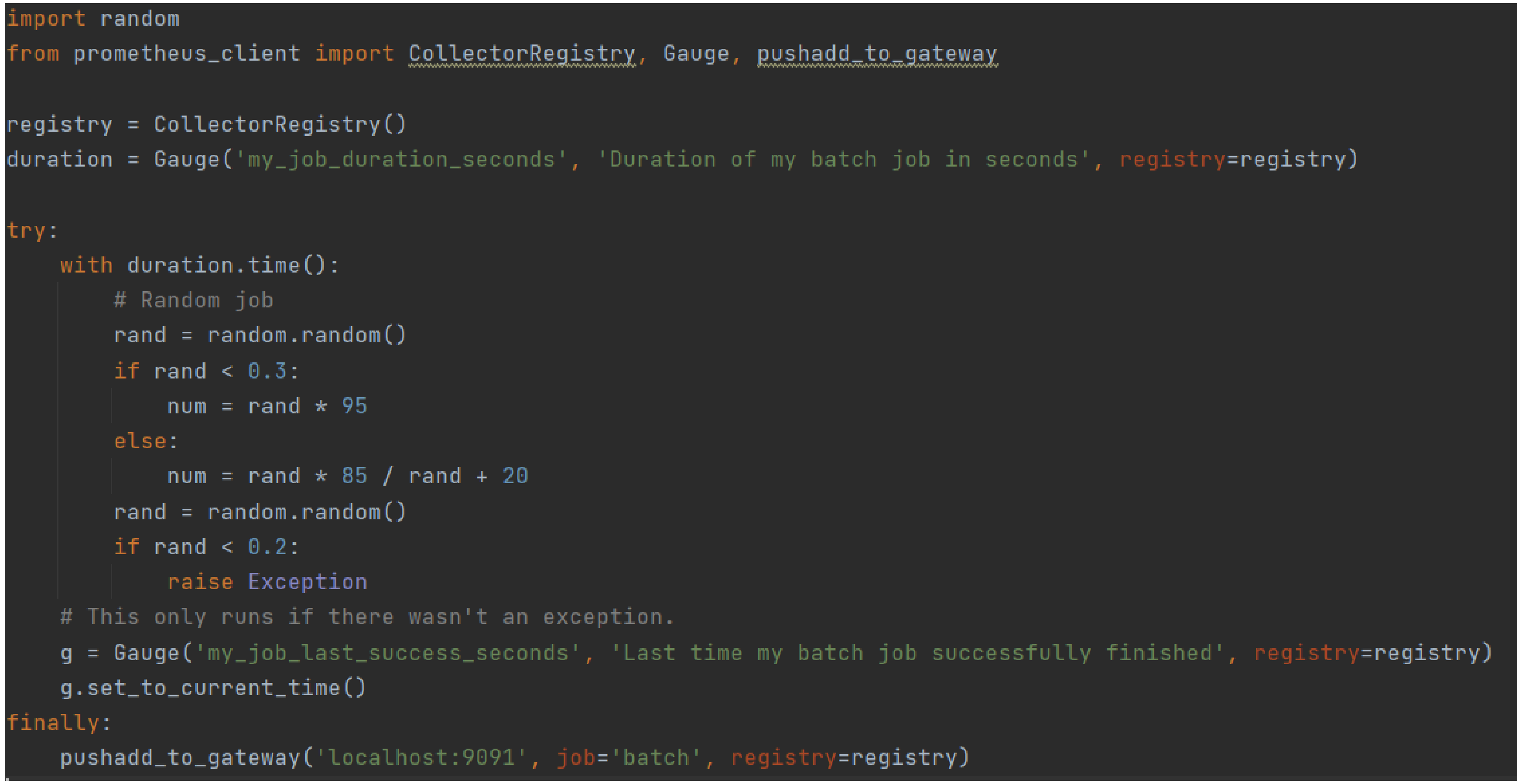
instance="localhost:9091" job="pushgateway" monitor="codelab-monitor"

ALERTMANAGER

- Inhibicija
  - Sprečavanje da se pošalje notifikacija za neki događaj u određenim uslovima
  - Ako je neki ozbiljniji alert u firing modu, ne okidati manje ozbiljne alertove
- Utišavanje
  - Privremeno onesposobljavanje alerta
  - Npr. Gašenje servisa zbog održavanja
- Rutiranje
  - Routing tree – putanje kojima se šalju različite notifikacije
  - Različiti timovi, production/development, ...
- Grupisanje
  - Za sve događaje iz istog reka/datacentra/servisa slati samo 1 notifikaciju
- Prigušivanje i ponavljanje
  - Sprečavanje spama - za slučaj da se pojavi novi alert iz grupe za koju je već poslata notifikacija, taj alert će da se priguši
  - Po potrebi će ista notifikacija da se pošalje više puta ako se ne vidi reakcija
- Obaveštavanje
  - Nakon svih prethodnih koraka, obaveštenja se šalju na prijemu adresu

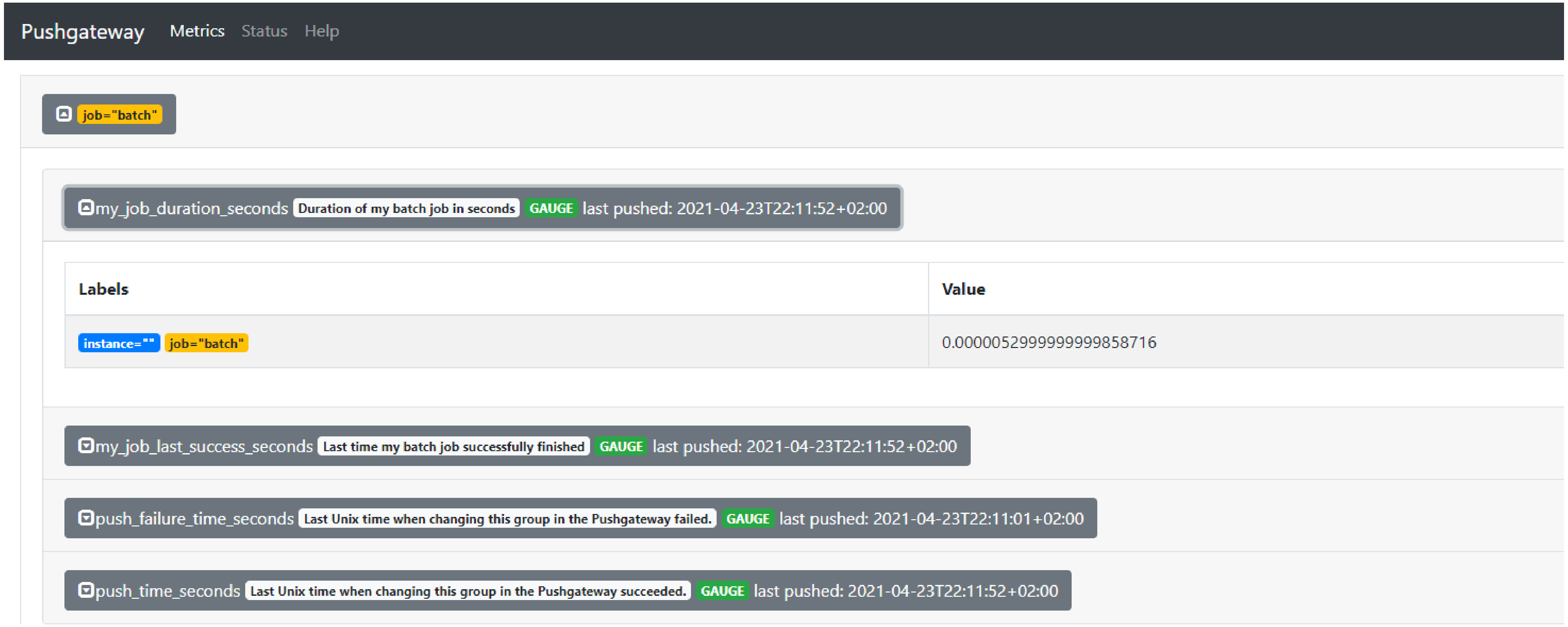
BATCH POSLOVI

- Poslovi koji se obavljaju periodično, odrade neku obradu i nestanu



```python
import random
from prometheus_client import CollectorRegistry, Gauge, pushadd_to_gateway

registry = CollectorRegistry()
duration = Gauge('my_job_duration_seconds', 'Duration of my batch job in seconds', registry=registry)

try:
    with duration.time():
        # Random job
        rand = random.random()
        if rand < 0.3:
            num = rand * 95
        else:
            num = rand * 85 / rand + 20
        rand = random.random()
        if rand < 0.2:
            raise Exception
    # This only runs if there wasn't an exception.
    g = Gauge('my_job_last_success_seconds', 'Last time my batch job successfully finished', registry=registry)
    g.set_to_current_time()
finally:
    pushadd_to_gateway('localhost:9091', job='batch', registry=registry)
```

PUSHGATEWAY



Pushgateway Metrics Status Help

job="batch"

my_job_duration_seconds Duration of my batch job in seconds GAUGE last pushed: 2021-04-23T22:11:52+02:00

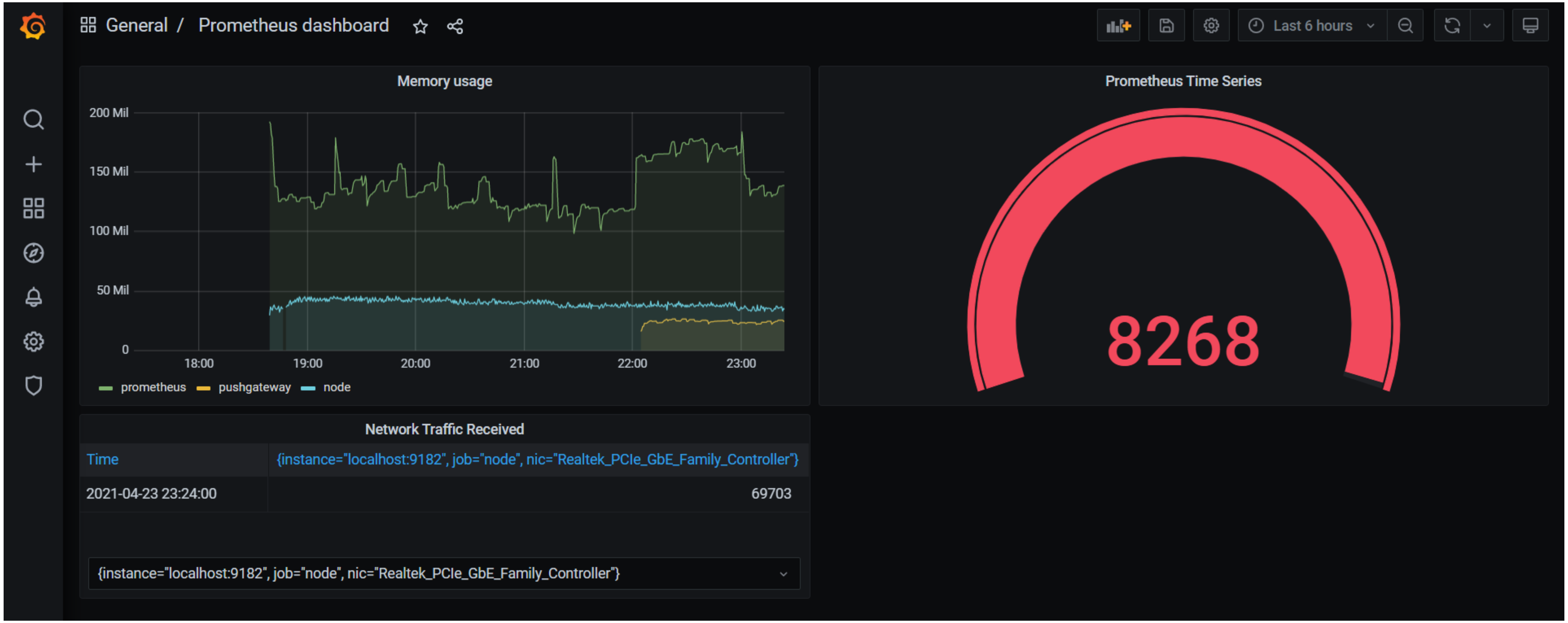
Labels	Value
instance="" job="batch"	0.000005299999999858716

my_job_last_success_seconds Last time my batch job successfully finished GAUGE last pushed: 2021-04-23T22:11:52+02:00

push_failure_time_seconds Last Unix time when changing this group in the Pushgateway failed. GAUGE last pushed: 2021-04-23T22:11:01+02:00

push_time_seconds Last Unix time when changing this group in the Pushgateway succeeded. GAUGE last pushed: 2021-04-23T22:11:52+02:00

GRAFANA



Time	{instance="localhost:9182", job="node", nic="Realtek_PCIe_GbE_Family_Controller"}
2021-04-23 23:24:00	69703

{instance="localhost:9182", job="node", nic="Realtek_PCIe_GbE_Family_Controller"}

LITERATURA

- Prometheus dokumentacija - https://prometheus.io/
- Brian Brazil – „Prometheus Up & Running – Infrastructure and Application performance monitoring“ – O'Reilly
- Google Cloud, Cloud Architecture Center
  - Remote monitoring and alerting for IoT - https://cloud.google.com/architecture/remote-monitoring-and-alerting-for-iot
  - Using Prometheus and Grafana for IoT monitoring - https://cloud.google.com/community/tutorials/cloud-iot-prometheus-monitoring
- Introduction to monitoring Microservices with Prometheus https://winderresearch.com/introduction-to-monitoring-microservices-with-prometheus/