



دانشگاه صنعتی امیر کبیر
(بلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

پروژه پایانی درس شبکه های کامپیوتری

پیاده سازی سیستم کلاینت - سرور و پرومیتوس

نگارش:

محمدسپهر توکلی کرمانی - ۹۸۳۱۱۱۱

استاد درس:

دکتر صادقیان

خرداد ۱۴۰۱

در اولین بخش ابتدا به پیاده سازی یک کلایت و یک سرور با قابلیت multithread بودن ، می پردازیم تا بتوانیم یک سرور با چندین کلاینت متصل به آن داشته باشیم و اگر سرور با خطا مواجه شد کلاینت قابلیت اتصال مجدد را داشته باشد.

برای پیاده سازی از کتابخانه socket استفاده می کنیم و برای بخش چند ریسمانی از کتابخانه threading استفاده می کنیم.

بخش اول : پیاده سازی agent

برای پیاده سازی کلایت (agent) به شکل زیر عمل می کنیم و برای تهیه متریک ها از کتابخانه psutil و برای ارسال آنها به سرور از قالب JSON استفاده می کنیم. در اینجا ۴ متریک در نظر می گیریم :

(۱) مقدار درصد مصرفی رم

(۲) مقدار درصد مصرفی پردازنده در طول ۴ ثانیه

(۳) مقدار رم مصرفی در پردازش فعلی (agent) بر حسب گیگابایت

(۴) تعداد وقفه های سیستم از زمان بوت شدن

با توجه به موارد ذکر شده کد پیاده شده برای agent به شکل زیر خواهد بود :

```

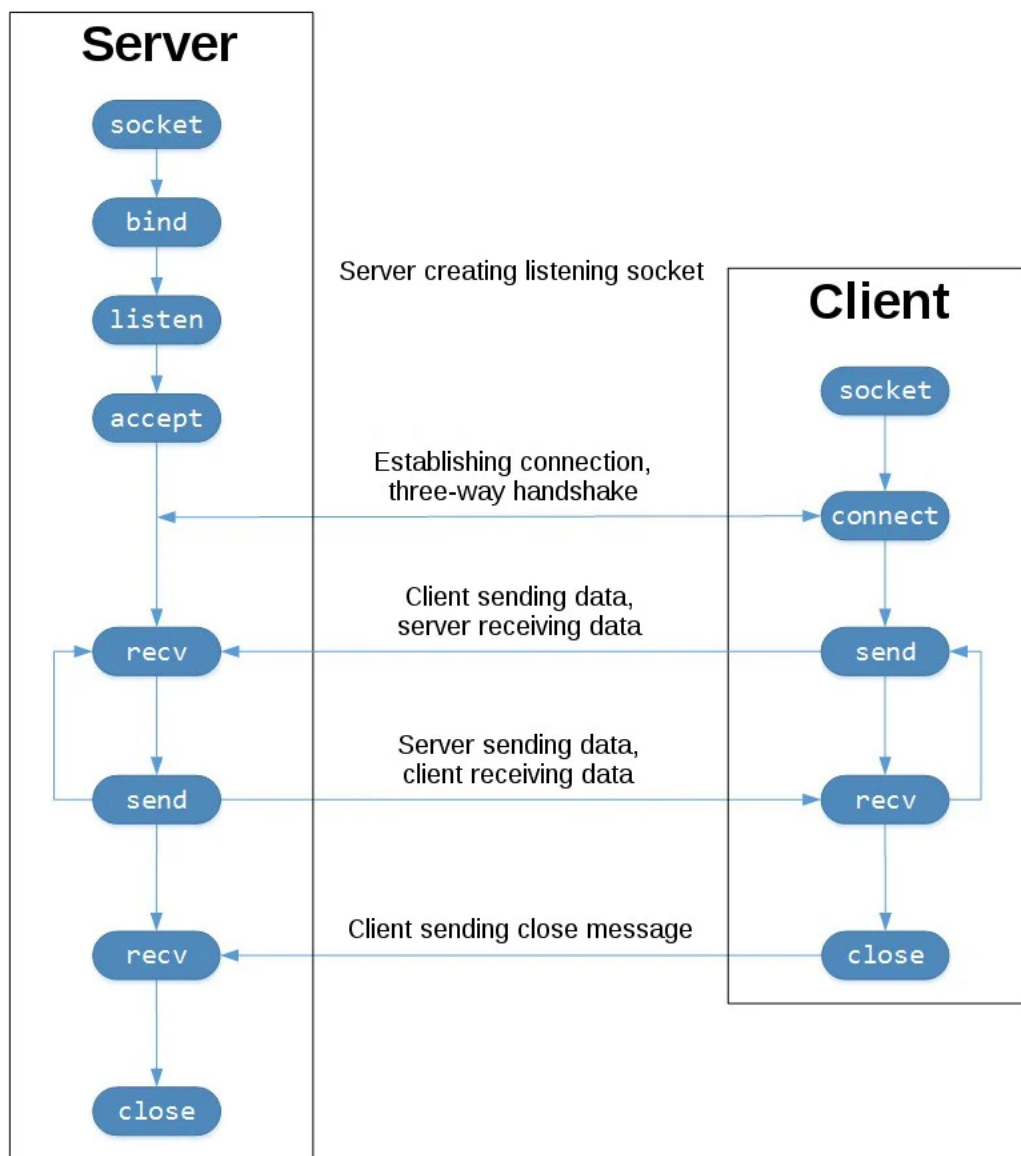
agent.py > ...
1  from re import T
2  import socket
3  from time import sleep
4  import os
5  import psutil
6  import json
7
8  IP = socket.gethostbyname(socket.gethostname())
9  PORT = 5578
10 ADDR = (IP, PORT)
11 SIZE = 1024
12 FORMAT = "utf-8"
13
14 def main():
15     while True:
16         try:
17             client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
18             client.connect(ADDR)
19             print(f"[CONNECTED] Client connected to server at {IP}:{PORT}")
20
21             connected = True
22             numberOfMsg = 1
23             while connected:
24
25                 pid = os.getpid()
26                 python_process = psutil.Process(pid)
27                 memoryUse = python_process.memory_info()[0]/2.**30 # memory use in GB
28                 myMetrics = {
29                     'currentProcessRAM: ': memoryUse,
30                     'RAM usage: ': psutil.virtual_memory()[2],
31                     'CPU usage: ': psutil.cpu_percent(4),
32                     'CPU interrupts: ': psutil.cpu_stats()[1]
33                 }
34
35                 # Serializing json
36                 msg = json.dumps(myMetrics)
37
38                 #send message
39                 client.send(msg.encode(FORMAT))
40                 print("Message "+str(numberOfMsg)+" sent to server.")
41                 numberOfMsg += 1
42
43                 sleep(5)
44             except Exception as e:
45                 print(e)
46                 print("server connection failed ! Do you want to connect again ? y/n")
47                 pm = input()
48                 if(pm=="n"):
49                     break
50                 else:
51                     continue
52
53 if __name__ == "__main__":
54     main()

```

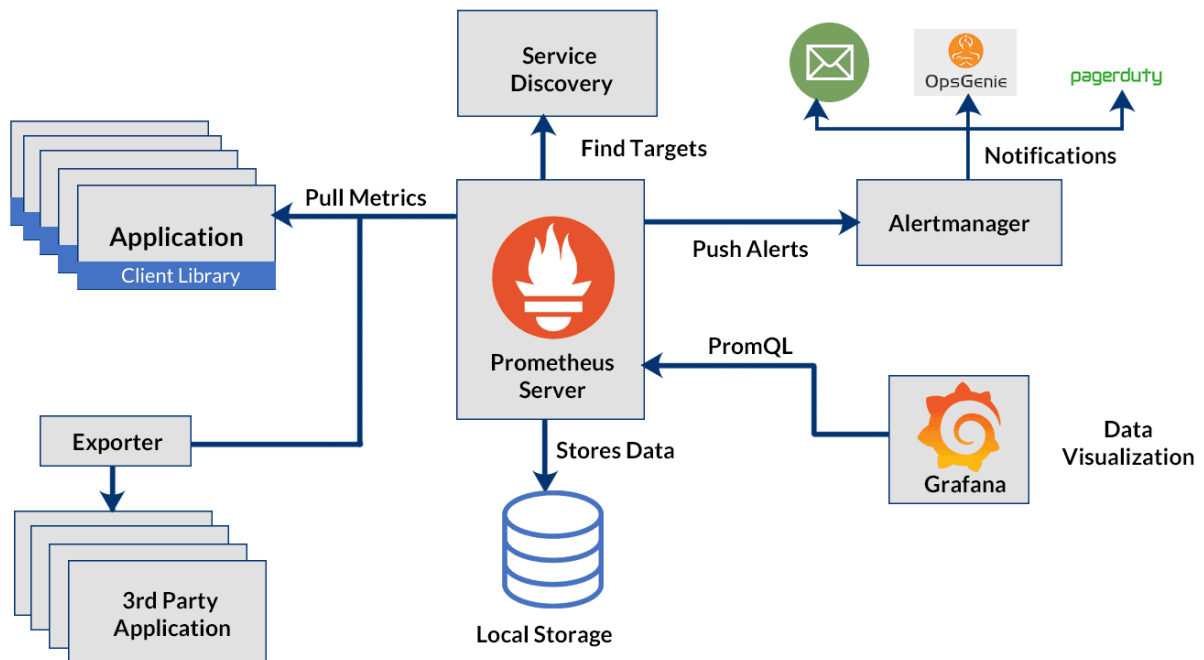
بخش دوم : پیاده سازی server

در این قسمت نیز با استفاده از کتابخانه socket سرور را پیاده می کنیم و آدرس ip لوکال (۱۲۷.۰.۰.۱) و هم چنین شماره پورت ۵۵۷۸ را برای آن در نظر می گیریم. Agent نیز با استفاده از این مقادیر به سرور متصل می شود و برای آن دیتا (متریک ها) را ارسال می کند. در اینجا سرور ما multithreaded است و با accept کردن هر کلاینت یک thread برای آن می سازد تا بتوانیم چندین کلاینت همزمان متصل به سرور داشته باشیم.

فرآیند اصلی اتصال به شکل زیر خواهد بود :



بخش سوم : اتصال سرور به پرومیتئوس



پرومیتئوس چیست ؟

Prometheus یک جعبه ابزار مانیتورینگ و هشدار سیستم منبع باز است که در ابتدا در SoundCloud ساخته شد. Prometheus معیارهای خود را به عنوان داده های سری زمانی جمع آوری و ذخیره می کند، یعنی اطلاعات متریک با مهر زمانی که در آن ثبت شده است، در کنار جفت های اختیاری کلید-مقدار به نام برچسب ها ذخیره می شود.

در این بخش متریک هایی که agent به سرور فرستاده است (در قالب json می باشد) را استخراج کرده و با استفاده از متریک Gauge به پرومیتئوس می دهیم تا تغییرات آن متریک را در گذر زمان مشاهده کنیم. حالا بگوییم چرا از متریک Gauge استفاده کرده ایم:

پرومتئوس در کل ۴ متریک دارد :

: counter(۱)

شمارنده یک متریک تجمعی است که نشان دهنده یک شمارنده یکنواخت در حال افزایش است که مقدار آن تنها با راه اندازی مجدد می تواند افزایش یابد یا به صفر بازنشانی شود.

مثال :

```
from prometheus_client import Counter
c = Counter('my_failures', 'Description of counter')
c.inc()      # Increment by 1
c.inc(1.6)   # Increment by given value
```

: gauge(۲)

گیج متریکی است که یک مقدار عددی واحد را نشان می دهد که می تواند بالا و پایین برود. گیج ها معمولاً برای مقادیر اندازه گیری شده مانند دما یا میزان مصرف فعلی حافظه استفاده می شوند، اما همچنین «شمارش هایی» که می توانند بالا و پایین بروند، مانند تعداد درخواست های همزمان، استفاده می شوند.

مثال :

```
from prometheus_client import Gauge
g = Gauge('my_inprogress_requests', 'Description of gauge')
g.inc()      # Increment by 1
g.dec(10)     # Decrement by given value
g.set(4.2)    # Set to a given value
```

: histogram(۳)

یک هیستوگرام از مشاهدات (معمولاً چیزهایی مانند مدت زمان درخواست یا اندازه پاسخ) نمونه برداری می کند و آنها را در سطل های قابل تنظیم شمارش می کند. همچنین مجموع تمام مقادیر مشاهده شده را ارائه می دهد.

مثال :

```
from prometheus_client import Histogram
h = Histogram('request_latency_seconds', 'Description of histogram')
h.observe(4.7)    # Observe 4.7 (seconds in this case)
```

۴: summery()

شبیه به هیستوگرام، یک خلاصه مشاهدات را نمونه می‌کند (معمولا چیزهایی مانند مدت زمان درخواست و اندازه پاسخ). در حالی که تعداد کل مشاهدات و مجموع تمام مقادیر مشاهده شده را نیز ارائه می‌دهد، کمیت‌های قابل تنظیم را در یک پنجره زمانی کشویی محاسبه می‌کند.

مثال :

```
from prometheus_client import Summary
s = Summary('request_latency_seconds', 'Description of summary')
s.observe(4.7) # Observe 4.7 (seconds in this case)
```

ما در اینجا نیاز به افزایش و کاهش متغیر مان برای متریک‌های ارسالی به توسط agent داریم ، بنابراین از متریک guage و متد set آن استفاده می‌کنیم. هم چنین برای هر متریک یک لیبل که نام agent ارسالی آن را مشخص می‌کند در نظر می‌گیریم. در اینجا ۴ متریک از نوع guage تعریف می‌کنیم :

```
metric1 = Gauge('process_ram_usage_percent', 'usage of my ram',['agent_number'])
metric2 = Gauge('process_cpu_usage_percent', 'usage of my cpu in 4 seconds',['agent_number'])
metric3 = Gauge('process_ram_gigabytes', 'usage of my ram in current process',['agent_number'])
metric4 = Gauge('process_cpu_interrupts_total', 'number of cpu interrupts since boot',['agent_number'])
```

```
metric1.labels("agent_" + str(number)).set(metrics[0])
metric2.labels("agent_" + str(number)).set(metrics[1])
metric3.labels("agent_" + str(number)).set(metrics[2])
metric4.labels("agent_" + str(number)).set(metrics[3])
```

به منظور نمایش متریک‌ها روی پرومیتئوس سرور http را روی پورت 8001 بالا می‌آوریم :

```
start_http_server(8001)
```

پس از توضیحات فوق و اتصال به پرومیتئوس توسط متریک‌ها ، کد سرور به شرح زیر خواهد بود :

```

server.py > ...
1  import socket
2  import threading
3  import json
4  from prometheus_client import Gauge, start_http_server
5
6  IP = socket.gethostname(socket.gethostname())
7  PORT = 5578
8  ADDR = (IP, PORT)
9  SIZE = 1024
10 FORMAT = "utf-8"
11
12 metric1 = Gauge('process_ram_usage_percent', 'usage of my ram', ['agent_number'])
13 metric2 = Gauge('process_cpu_usage_percent', 'usage of my cpu in 4 seconds', ['agent_number'])
14 metric3 = Gauge('process_ram_gigabytes', 'usage of my ram in current process', ['agent_number'])
15 metric4 = Gauge('process_cpu_interrupts_total', 'number of cpu interrupts since boot', ['agent_number'])
16
17 def handle_client(conn, addr, number):
18     print(f"[NEW CONNECTION] {addr} connected.")
19
20     connected = True
21     while connected:
22         msg = conn.recv(SIZE).decode(FORMAT)
23
24         print(f"[{addr}] {msg}")
25         metricsFromClient = json.loads(msg)
26
27         metrics = []
28         metrics.append(metricsFromClient['RAM usage: '])
29         metrics.append(metricsFromClient['CPU usage: '])
30         metrics.append(metricsFromClient['currentProcessRAM: '])
31         metrics.append(metricsFromClient['CPU interrupts: '])
32
33         metric1.labels("agent_" + str(number)).set(metrics[0])
34         metric2.labels("agent_" + str(number)).set(metrics[1])
35         metric3.labels("agent_" + str(number)).set(metrics[2])
36         metric4.labels("agent_" + str(number)).set(metrics[3])
37
38     conn.close()
39
40 def main():
41     print("[STARTING] Server is starting...")
42     server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
43     server.bind(ADDR)
44     server.listen()
45
46     print(f"[LISTENING] Server is listening on {IP}:{PORT}")
47     number = 1
48     start_http_server(8001)
49
50     while True:
51         conn, addr = server.accept()
52         thread = threading.Thread(target=handle_client, args=(conn, addr, number))
53         thread.start()
54         print(f"[ACTIVE CONNECTIONS] {number}")
55         number += 1
56
57 if __name__ == "__main__":
58     main()

```


بخش چهارم: اجرای برنامه کلایت و سرور

در این بخش به اجرای کد های نوشته شده در بخش های قبلی می پردازیم ، اما قبل از آن لازم است تا فایل yml مربوط به کانفیگ پرومتهوس را کمی تغییر داده و یک job به آن اضافه کنیم تا یک target جدید به وجود آوریم:

```
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus"

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

    static_configs:
      - targets: ["localhost:9090"]

  - job_name: "my_exporter"
    static_configs:
      - targets: ["localhost:8001"]
```

حالا برنامه سرور را اجرا می کنیم :

```
Sepehrs-MacBook-Pro:shabake-project sepehr$ python3 server.py
[STARTING] Server is starting...
[LISTENING] Server is listening on 127.0.0.1:5578
```

اکنون سرور آماده اتصال است ، پس حالا agent را اجرا می کنیم :

```
Sepehrs-MacBook-Pro:shabake-project sepehr$ python3 agent.py
[CONNECTED] Client connected to server at 127.0.0.1:5578
Message 1 sent to server.
```

و agent به سرور متصل شده و شروع به ارسال متریک ها در قالب json میکند :

```
Sepehrs-MacBook-Pro:shabake-project sepehr$ python3 server.py
[STARTING] Server is starting...
[LISTENING] Server is listening on 127.0.0.1:5578
[NEW CONNECTION] ('127.0.0.1', 49810) connected.
[ACTIVE CONNECTIONS] 1
[('127.0.0.1', 49810)] {"currentProccessRAM: ": 0.00927734375, "RAM usage: ": 58.4, "CPU us
age: ": 2.8, "CPU interrupts: ": 706148}
```

حالا اگر سرور با خطا مواجه شود ، کلاینت قابلیت تصمیم گیری برای اتصال مجدد را دارد:

```
server connection failed ! Do you want to connect again ? y/n
```

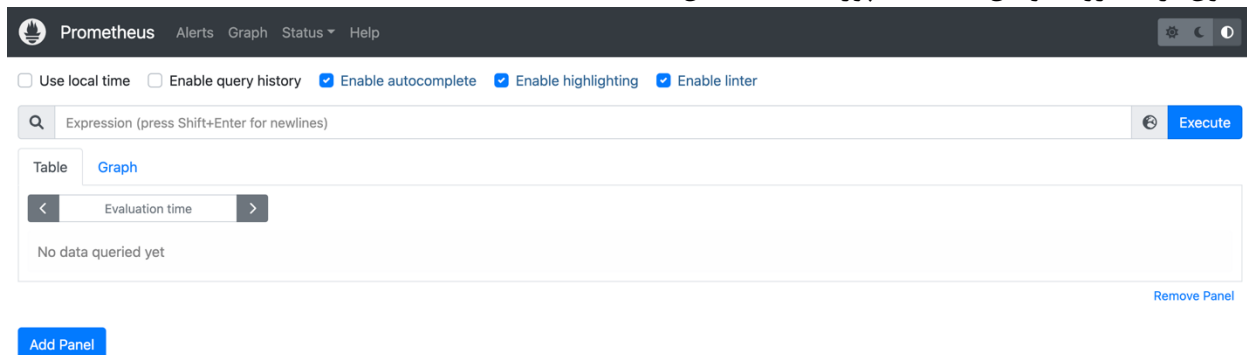
در اینجا اگر n وارد کنیم کار کلاینت تمام می شود و اگر y وارد کنیم برای ارتباط مجدد تلاش می کند.

بخش پنجم : اجرای برنامه پرومیتئوس و مشاهده متریک ها

در بخش ابتدا برنامه پرومیتئوس را اجرا می کنیم :

```
Sepehrs-MacBook-Pro:prometheus-2.35.0.darwin-amd64 sepehr$ ./prometheus
ts=2022-06-04T10:31:27.730Z caller=main.go:488 level=info msg="No time or size retention was set so using the default time retention" duration=15d
ts=2022-06-04T10:31:27.730Z caller=main.go:525 level=info msg="Starting Prometheus" version="(version=2.35.0, branch=HEAD, revision=6656cd29fe6ac92bab91ecec0fe162ef0f187654)"
ts=2022-06-04T10:31:27.730Z caller=main.go:530 level=info build_context="(go=go1.18.1, user=root@2de5cf526c15, date=20220421-09:43:22)"
ts=2022-06-04T10:31:27.730Z caller=main.go:531 level=info host_details=(darwin)
ts=2022-06-04T10:31:27.730Z caller=main.go:532 level=info fd_limits="(soft=256, hard=unlimited)"
ts=2022-06-04T10:31:27.730Z caller=main.go:533 level=info vm_limits="(soft=unlimited, hard=unlimited)"
ts=2022-06-04T10:31:27.736Z caller=web.go:541 level=info component=web msg="Start listening for connections" address=0.0.0.0:9090
ts=2022-06-04T10:31:27.737Z caller=main.go:957 level=info msg="Starting TSDB ..."
ts=2022-06-04T10:31:27.740Z caller=tls_config.go:195 level=info component=web msg="TLS is disabled." http2=false
ts=2022-06-04T10:31:27.740Z caller=repair.go:57 level=info component=tsdb msg="Found healthy block" mint=1653824302402 maxt=1653825600000 ulid=01G487R3QCJR6E6S24DXWJWJ4K
ts=2022-06-04T10:31:27.741Z caller=repair.go:57 level=info component=tsdb msg="Found healthy block" mint=1654170727402 maxt=1654171200000 ulid=01G4Q2Z3GH5A1S46S8JZPEWHE2
ts=2022-06-04T10:31:27.742Z caller=repair.go:57 level=info component=tsdb msg="Found healthy block" mint=1654171207405 maxt=1654178400000 ulid=01G4Q2Z46J32MTRNNVPSK65F1D
ts=2022-06-04T10:31:27.743Z caller=repair.go:57 level=info component=tsdb msg="Found healthy block" mint=1653825607349 maxt=1653854400000 ulid=01G4Q2Z5KM1ABYF9SZZHJ2XADD
ts=2022-06-04T10:31:27.755Z caller=head.go:493 level=info component=tsdb msg="Replaying on-disk memory mappable chunks if any"
ts=2022-06-04T10:31:27.755Z caller=head.go:536 level=info component=tsdb msg="On-disk memory mappable chunks replay completed" duration=123.841µs
ts=2022-06-04T10:31:27.755Z caller=head.go:542 level=info component=tsdb msg="Replaying WAL, this may take a while"
ts=2022-06-04T10:31:27.762Z caller=head.go:578 level=info component=tsdb msg="WAL checkpoint loaded"
ts=2022-06-04T10:31:27.766Z caller=head.go:613 level=info component=tsdb msg="WAL segment loaded" segment=7 maxSegment=10
ts=2022-06-04T10:31:27.767Z caller=head.go:613 level=info component=tsdb msg="WAL segment loaded" segment=8 maxSegment=10
ts=2022-06-04T10:31:27.770Z caller=head.go:613 level=info component=tsdb msg="WAL segment loaded" segment=9 maxSegment=10
ts=2022-06-04T10:31:27.770Z caller=head.go:613 level=info component=tsdb msg="WAL segment loaded" segment=10 maxSegment=10
ts=2022-06-04T10:31:27.770Z caller=head.go:619 level=info component=tsdb msg="WAL replay completed" checkpoint_replay_duration=7.035604ms wal_replay_duration=8.634489ms total_replay_duration=15.813454ms
ts=2022-06-04T10:31:27.772Z caller=main.go:978 level=info fs_type=1a
ts=2022-06-04T10:31:27.772Z caller=main.go:981 level=info msg="TSDB started"
ts=2022-06-04T10:31:27.772Z caller=main.go:1162 level=info msg="Loading configuration file" filename=prometheus.yml
ts=2022-06-04T10:31:28.453Z caller=main.go:1199 level=info msg="Completed loading of configuration file" filename=prometheus.yml totalDuration=681.256615ms db_storage=523ns remote_storage=6.657µs web_handler=348ns query_engine=591ns scrape=679.444031ms scrape_sd=103.777µs notify=979.845µs notify_sd=34.969µs rules=5.936µs tracing=403.621µs
ts=2022-06-04T10:31:28.453Z caller=main.go:930 level=info msg="Server is ready to receive web requests."
ts=2022-06-04T10:31:34.899Z caller=compact.go:510 level=info component=tsdb msg="write block resulted in empty block" mint=1654178400000 maxt=1654185600000 duration=102.713757ms
ts=2022-06-04T10:31:34.902Z caller=head.go:840 level=info component=tsdb msg="Head GC completed" duration=1.459075ms
ts=2022-06-04T10:31:34.904Z caller=checkpoint.go:98 level=info component=tsdb msg="Creating checkpoint" from_segment=7 to_segment=8 mint=1654185600000
ts=2022-06-04T10:31:34.982Z caller=head.go:1009 level=info component=tsdb msg="WAL checkpoint complete" first=7 last=8 duration=77.694885ms
```

اکنون برنامه روی لوکال هاست با پورت ۹۰۹۰ قابل مشاهده است :

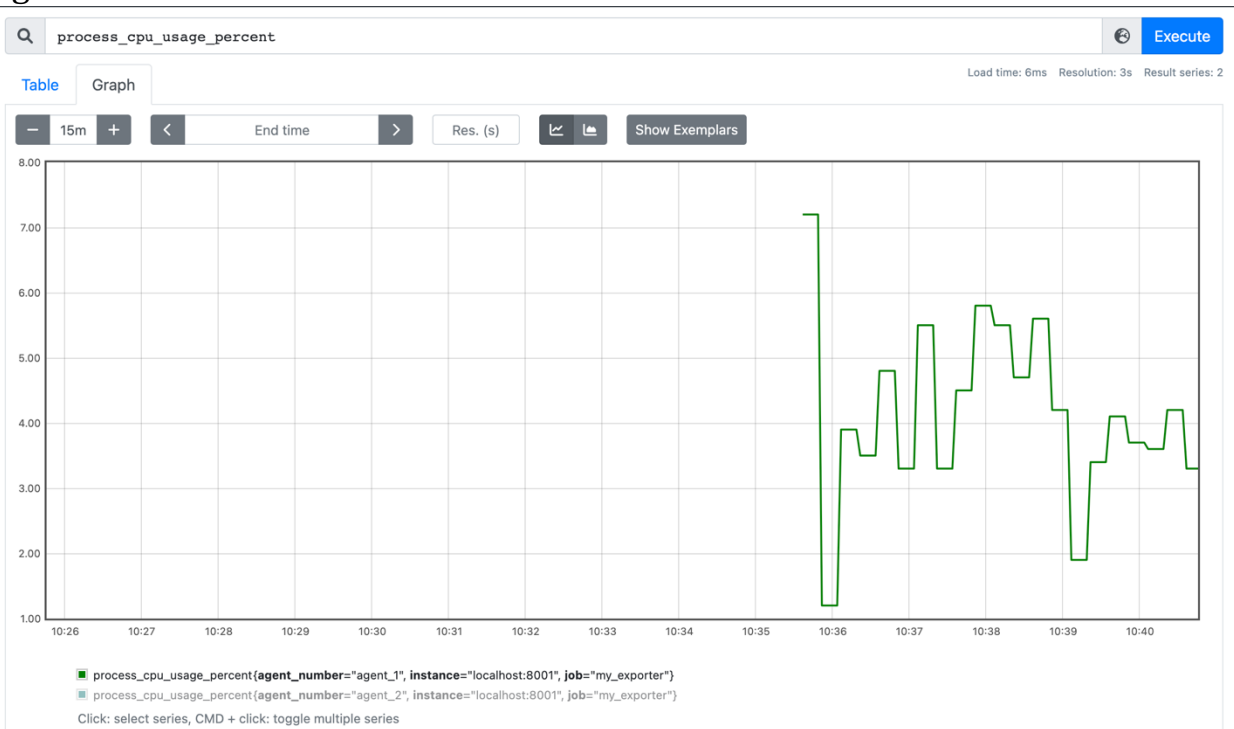


حالا سرور را به همراه ۲ agent (برای راحتی نوشتن گزارش ۲ تا ولی می توان بیشتر هم متصل کرد) اجرا می کنیم و در بخش target می توانیم متریک ها را مشاهده کنیم :

```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 131.0
python_gc_objects_collected_total{generation="1"} 244.0
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"} 39.0
python_gc_collections_total{generation="1"} 3.0
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="9",patchlevel="4",version="3.9.4"} 1.0
# HELP process_ram_usage_percent usage of my ram
# TYPE process_ram_usage_percent gauge
process_ram_usage_percent{agent_number="agent_1"} 58.1
process_ram_usage_percent{agent_number="agent_2"} 58.1
# HELP process_cpu_usage_percent usage of my cpu in 4 seconds
# TYPE process_cpu_usage_percent gauge
process_cpu_usage_percent{agent_number="agent_1"} 3.4
process_cpu_usage_percent{agent_number="agent_2"} 3.9
# HELP process_ram_gigabytes usage of my ram in current process
# TYPE process_ram_gigabytes gauge
process_ram_gigabytes{agent_number="agent_1"} 0.00943756103515625
process_ram_gigabytes{agent_number="agent_2"} 0.009246826171875
# HELP process_cpu_interrupts_total number of cpu interrupts since boot
# TYPE process_cpu_interrupts_total gauge
process_cpu_interrupts_total{agent_number="agent_1"} 709961.0
process_cpu_interrupts_total{agent_number="agent_2"} 703125.0
```

به عنوان مثال گراف متریک **process_cpu_usage_percent** را برای هر دو ایجنت مشاهده می کنیم:

Agent 1 :



Agent 2 :



مشاهده می کنیم که درصد استفاده از cpu در سیستم های agent ها در گذر زمان متفاوت بوده و به طور مداوم در حال افزایش یا کاهش است.
بقیه متریک ها هم به همین صورت قابل نمایش هستند :

متریک `process_ram_gigabytes` :



که نشان می دهد مقدار رم اختصاص داده شده به آن پردازش تفاوت خیلی زیادی در گذر زمان نکرده است.

متریک `process_cpu_interrupts_total` :



متریک : process_ram_usage_percent

