

پروژه ی شبکه

دانشجو: نگین خیرمند

شماره دانشجویی: ۹۸۳۱۰۲۳

توضیحات پروژه بخش اول:

۱. یک برنامه agent بنویسید که متریک‌های مهم برای بررسی وضعیت سیستم را از سیستمی که بر روی آن اجرا شده است دریافت و در بازه‌های زمانی مختلف برای سرور واسط به کمک سوکت ارسال کند.

نکته ۱: ارتباط بین agent و سرور واسط صرفاً به کمک سوکت و بدون استفاده از هیچ لایبرری دیگری انجام شود.

نکته ۲: یک پروتکل برای تشخیص ساختار دیتا بین agent و سرور واسط نیاز هست. می‌توانید از json یا هر ساختاری استفاده کنید.

نکته ۳: ایجنت باید به گونه‌ای پیاده‌سازی شود که در صورت ایجاد هر مشکلی در ارتباط با سرور بتواند پس از رفع مشکل مجدد ارتباط را ایجاد کند.

نکته ۴: تمام کدها با زبان پایتون یا گو نوشته شود.

ابتدا یک سری method نوشته شده که داده های سیستم را دریافت و در فرمت مناسب(dictionary) قرار میدهند:

```
10 def getCpuPercentage(time):
11     # print('The CPU usage is: ', psutil.cpu_percent(4))
12     return psutil.cpu_percent(time)
13
14 def getRamPercentage():
15     return psutil.virtual_memory()[2]
16
17 def getData():
18     data = {
19         "cpu_percent": getCpuPercentage(1),
20         "ram_percent": getRamPercentage()
21     }
22     data = json.dumps(data)
23     return data
24
```

سپس method ای که مسول قسمت socket programming است را نوشته ام:

```

37 def create_agent():
38     HOST = "127.0.0.1" # The server's hostname or IP address
39     PORT = 2004 # The port used by the server
40     try:
41         global s
42         with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
43             s.connect((HOST, PORT))
44             port_number = s.recv(1024).decode()
45             print("agent on port", port_number, "started, connection to the server stablished")
46             time.sleep(1)
47             while(True):
48                 a = str(getData())
49                 time.sleep(1)
50                 s.sendall(bytes(bytearray(a, encoding='utf-8')))
51                 data = s.recv(1024)
52                 if not data:
53                     end()
54                     # exit()
55                     return
56             except KeyboardInterrupt:
57                 print("keyboard interrupt detected, exiting...")
58                 end()
59                 sys.exit(0)
60         except:
61             end()
62         return
63

```

با توجه به اینکه قرار است روی پرت 2004 برنامه ی سرور را run بکنم برای همین پورت target را ۲۰۰۴ قرار داده ام.

و یک TCP connection به وجود آورده و داده های dictionary را به صورت JSON برای agent server فرستاده ام. برای handle کردن exception ها هم از try و except استفاده کردم.

و در نهایت برای اینکه اگر exception هم داد باز تلاش برای ساختن یک connection جدید بکنم در قسمت main صدا زدن این تابع را در while True قرار داده ام.

```

64
65 if __name__ == '__main__':
66     try:
67         while(True):
68             create_agent()
69             end()
70         except KeyboardInterrupt:
71             print("keyboard interrupt detected, exiting...")
72             end()
73             sys.exit(0)

```

توضیحات پروژه بخش دوم:

۲. یک برنامه server بنویسید که بتواند کانکشن سوکت از agent های متفاوت دریافت کند. و هر زمان که یک agent متریک های جدید را برای این سرور ارسال می کند متریک های دریافتی را به کمک لایبرری های پرومیتئوس کلاینت ادغام و بر روی یک اندپوینت http نمایش دهید.

نکته: در این بخش لازم است با خواندن داکيومنت های پرومیتئوس مشخص کنید که از چه نوع دیتاتایپی برای متریک خود استفاده کنید. همچنین به کمک برچسب گذاری مشخص کنید که اطلاعات برای کدام agent است.

<https://prometheus.io/docs/instrumenting/clientlibs>

برای نشان دادن داده های دریافتی از agent های مختلف از class agentServer استفاده میشود.

```
You, 10 minutes ago | 1 author (You)
21 class agentServer(object):
22
23     def __init__(self):
24         pass
25
26     def collect(self):
27         global socketData
28
29         for key, value in socketData.items():
30             index = key[0] + ":" + str(key[1])
31             g = GaugeMetricFamily("CPU_Usage", 'Help text', labels=['CPU_Usage'])
32             g.add_metric([ index ], value["cpu_percent"])
33             yield g
34
35             c = GaugeMetricFamily("RAM_Usage", 'Help text', labels=['RAM_Usage'])
36             c.add_metric([ index ], value["ram_percent"])
37             yield c
38
39     def create_metric():
40         start_http_server(8000)
41         REGISTRY.register(agentServer())
42
43
```

که طبق داکيومنتيشن نوشته شده:

https://github.com/prometheus/client_python

برای قسمت socket programming هم پس از ساختن socket و bind کردن آن، چون سرور باید توانایی ارتباط با چند agent به صورت همزمان را داشته باشد باید از thread ها استفاده کرد.

```
if __name__ == '__main__':
    create_metric()
    create_socket()
    bind_socket()
    try:
        while True:
            Client, address = ServerSideSocket.accept()

            print('Connected to: ' + address[0] + ':' + str(address[1]))

            start_new_thread(multi_threaded_client, (Client, address, ))
            ThreadCount += 1
            list_agents.append((Client, address))

        ServerSideSocket.close()
    except KeyboardInterrupt:
        print("keyboard interrupt")
        # print(str(e))
        print("ending")
        # ServerSideSocket.close()
        sys.exit(0)
```

```

90 def multi_threaded_client(connection, address):
91     try:
92         global ThreadCount, socketData, list_agents
93         connection.send(str.encode(str(address[1])))
94         while True:
95             data = connection.recv(2048)
96             #the data is the dictionary as json
97             if not data:
98                 break
99             agent_data_dict = json.loads(data.decode())
100             socketData[address] = agent_data_dict
101             connection.sendall(str.encode("alive"))
102             print_data_agents()
103             print("ending connection with agent:"+address[0]+str(address[1]))
104             list_agents.remove((connection, address))
105             del socketData[address]
106             ThreadCount-=1
107             end_connection(connection)
108         except:
109             list_agents.remove((connection, address))
110             del socketData[address]
111             ThreadCount-=1
112             end_connection(connection)
113             print("a connection with ", address[0], str(address[1]), "was forcibly closed by the remote host")
114             print_data_agents()
115             return
116

```

تنها نکته ای که کد دارد این است که socketData یک dictionary از dictionary است که هر key آن مربوط به connection ها با agent های مختلف است و value آن یک dictionary است که فرمت آن در کد agent مشخص شده است.

توضیحات پروژه بخش سوم:

۳. در این مرحله لازم است که ابزار پرومیتئوس را اجرا کنید و آن را به اندپوینت متریک که در مرحله قبل آماده کردید متصل کنید.

نکته: برای اتصال پرومیتئوس به سرور متریک ها می توانید از مستندات پرومیتئوس کمک بگیرید.

ابتدا target به فایل yml مربوط به prometheus اضافه شد و scrape_interval و evaluation_interval هم برای update کردن استفاده میشوند به 2s تغییر یافتند:

```

- job_name: "mine"
  # metrics_path defaults to '/metrics'
  # scheme defaults to 'http'.

static_configs:
  - targets: ["localhost:8000"]

```

```

1 # my global config
2 global:
3     scrape_interval: 2s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
4     evaluation_interval: 2s # Evaluate rules every 15 seconds. The default is every 1 minute.
5     # scrape_timeout is set to the global default (10s).
6

```

سپس prometheus.exe را run کردم:

و برنامه های خود را نیز run کردم:

```
client server
PS C:\Users\venus\Desktop\uni2\6th_SEMESTER\Computer Networking\project> python .\client.py
agent on port 3231 started, connection to the server established
agent on port 3245 started, connection to the server established

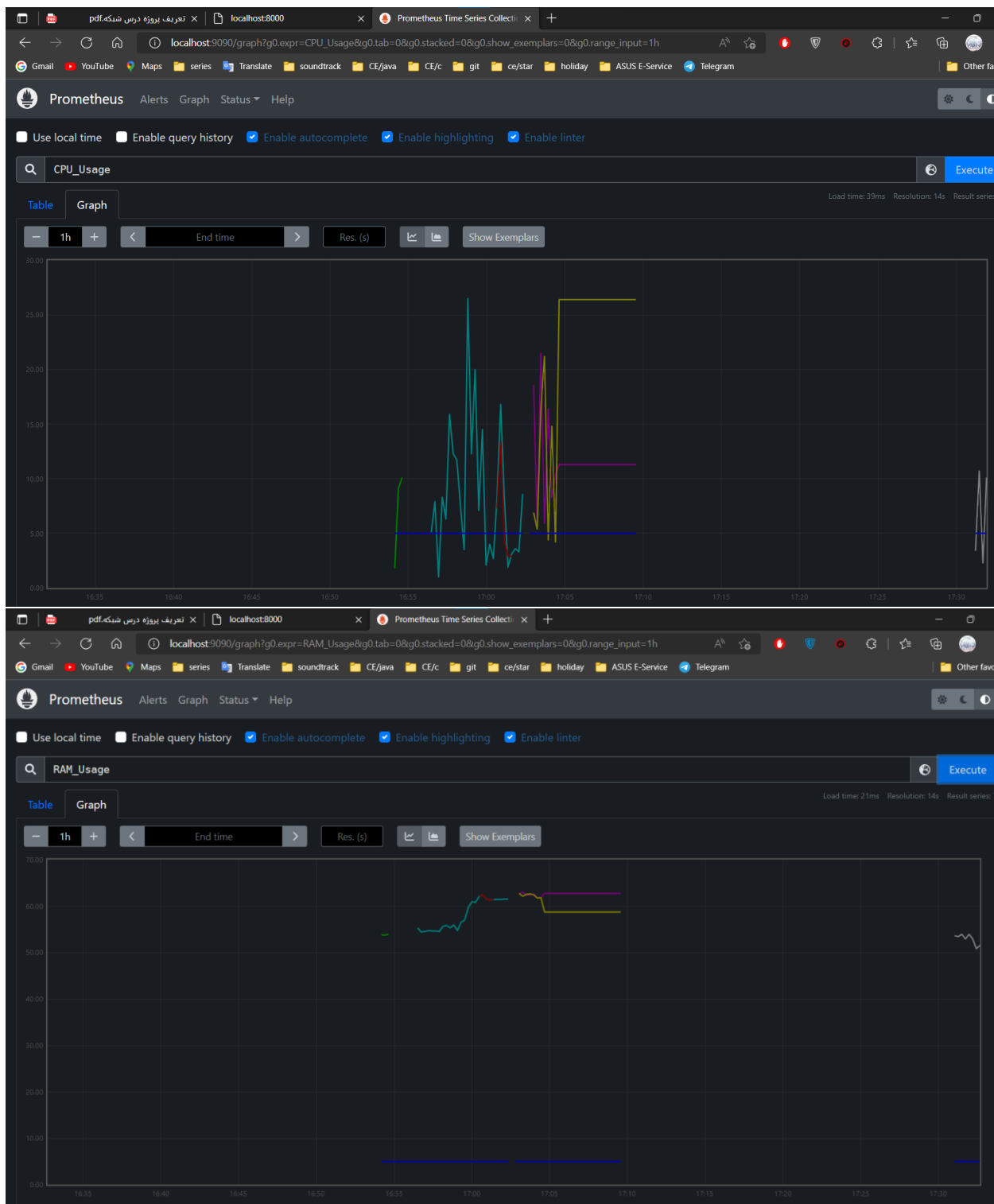
client server
PS C:\Users\venus\Desktop\uni2\6th_SEMESTER\Computer Networking\project> python .\client.py
agent on port 3231 started, connection to the server established
```

حالا برای دیدن داده های agent به ۸۰۰۰ رفتیم:

```
localhost:8000 Prometheus Time Series Collectio
localhost:8000
Gmail YouTube Maps series Translate soundtrack CE/java CE/c git ce/star holiday

# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 87.0
python_gc_objects_collected_total{generation="1"} 272.0
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"} 38.0
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="6",version="3.9.6"} 1.0
# HELP CPU_Usage Help text
# TYPE CPU_Usage gauge
CPU_Usage{CPU_Usage="s:t"} 5.0
# HELP RAM_Usage Help text
# TYPE RAM_Usage gauge
RAM_Usage{RAM_Usage="s:t"} 5.0
# HELP CPU_Usage Help text
# TYPE CPU_Usage gauge
CPU_Usage{CPU_Usage="127.0.0.1:3245"} 4.0
# HELP RAM_Usage Help text
# TYPE RAM_Usage gauge
RAM_Usage{RAM_Usage="127.0.0.1:3245"} 53.6
```

و برای دیدن prometheus به ۹۰۹۰ که پورت default است:



قسمت امتیازی:

از توضیحات در مورد نصب داکر گذر میکنم. میتوانید به این لینک مراجعه کنید، من همین راه را رفته ام.

<https://support.netfoundry.io/hc/en-us/articles/360057865692-Installing-Docker-and-docker-compose-for-Ubuntu-20-04>

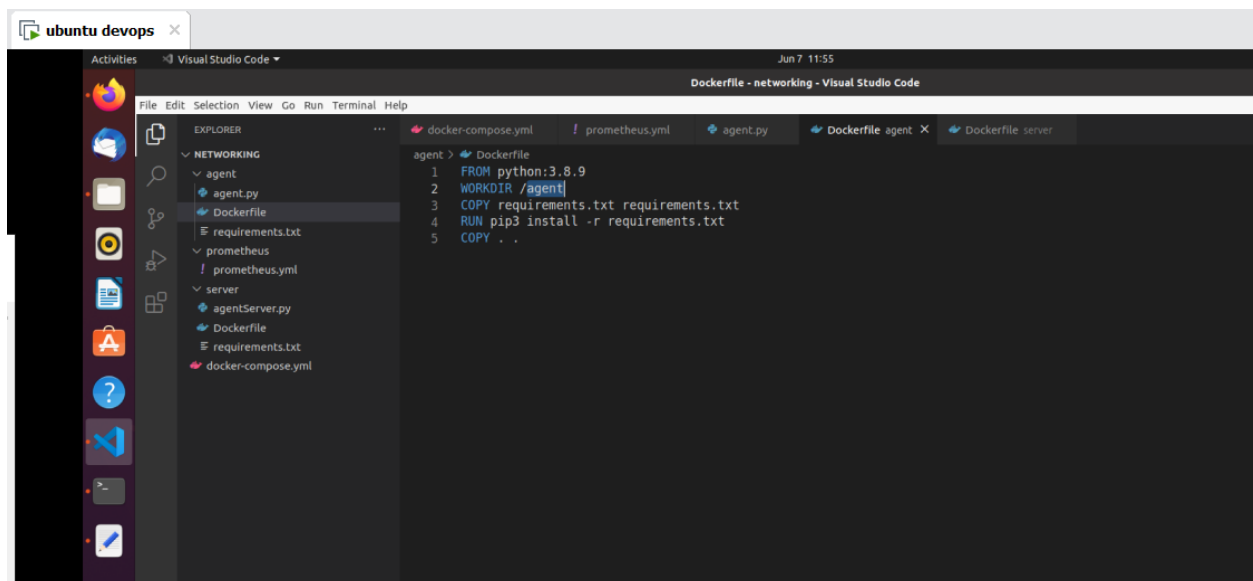
```
Activities Terminal Jun 7 10:42
kheirmand@ubuntu: ~/Desktop/networking
kheirmand@ubuntu:~/Desktop/networking$ touch agent.py
kheirmand@ubuntu:~/Desktop/networking$ touch agentServer.py
kheirmand@ubuntu:~/Desktop/networking$ code .
```

سپس کد های متناظر را در فایل های ساخته شده نوشتیم:

```
100         ThreadCount+=1
107     end_connection(connection)
108 except:
109     list_agents.remove((connection, address))
110     del socketData[address]
111     ThreadCount-=1
112     end_connection(connection)
113     print("a connection with ", address[0], str(address[1]), "
114     print_data_agents()
115     return
116
117 if __name__ == '__main__':
118     create_metric()
119     create_socket()
120     bind_socket()
121     try:
122         while True:
123             Client, address = ServerSideSocket.accept()
124
125             print('Connected to: ' + address[0] + ':' + str(address[1]))
126
127             start_new_thread(multi_threaded_client, (Client, address))
128             ThreadCount += 1
129             list_agents.append((Client, address))
130
131         ServerSideSocket.close()
132 except KeyboardInterrupt:
133     print("keyboard interrupt")
134     # print(str(e))
135     print("ending")
136     # ServerSideSocket.close()
137     sys.exit(0)
138
139
```

```
kheirmand@ubuntu: ~/Desktop/networking
kheirmand@ubuntu:~/Desktop/networking$ touch agent.py
kheirmand@ubuntu:~/Desktop/networking$ touch agentServer.py
kheirmand@ubuntu:~/Desktop/networking$ code .
kheirmand@ubuntu:~/Desktop/networking$ mkdir agent
kheirmand@ubuntu:~/Desktop/networking$ mkdir server
kheirmand@ubuntu:~/Desktop/networking$ mv agent.py agent/agent.py
kheirmand@ubuntu:~/Desktop/networking$ mv agentServer.py server/agentServer.py
kheirmand@ubuntu:~/Desktop/networking$
```

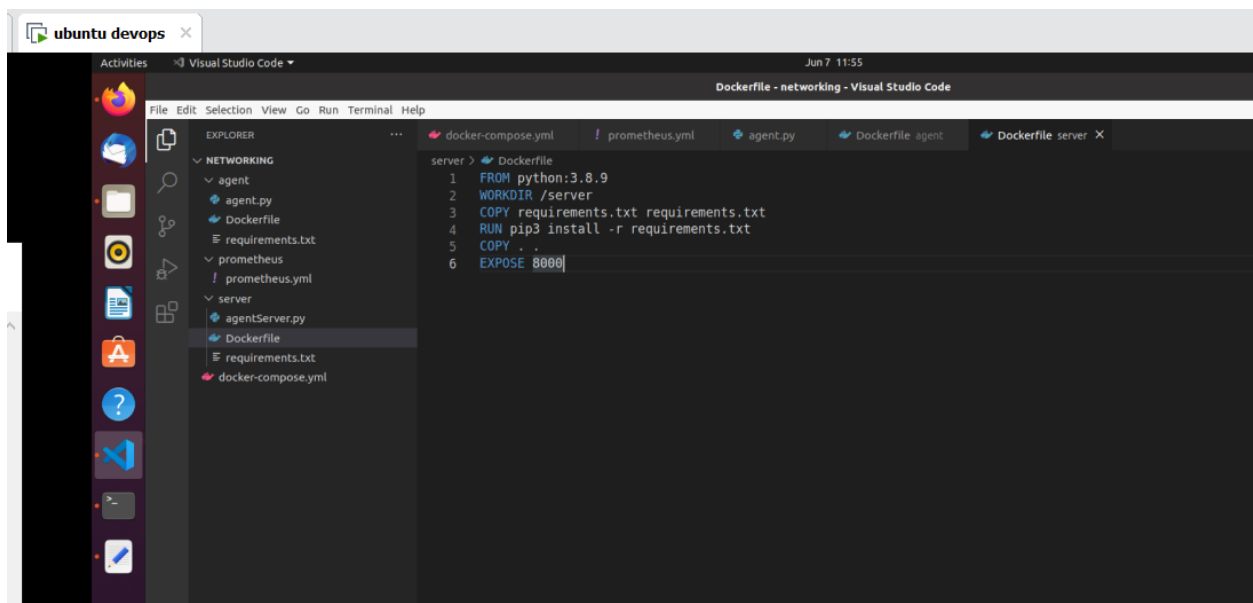
در نهایت با اضافه شدن Dockerfile ها به هر کدام از directory های server و agent با کد های زیر:
برای agent:



The screenshot shows the Visual Studio Code interface with the 'agent' Dockerfile open. The Explorer sidebar on the left shows the project structure with folders 'agent' and 'server'. The 'agent' folder is expanded, showing 'agent.py', 'Dockerfile', and 'requirements.txt'. The 'server' folder is also expanded, showing 'agentServer.py', 'Dockerfile', and 'requirements.txt'. The main editor area shows the content of the 'agent' Dockerfile:

```
agent > Dockerfile
1 FROM python:3.8.9
2 WORKDIR /agent
3 COPY requirements.txt requirements.txt
4 RUN pip3 install -r requirements.txt
5 COPY . .
```

برای server



The screenshot shows the Visual Studio Code interface with the 'server' Dockerfile open. The Explorer sidebar on the left shows the project structure. The main editor area shows the content of the 'server' Dockerfile:

```
server > Dockerfile
1 FROM python:3.8.9
2 WORKDIR /server
3 COPY requirements.txt requirements.txt
4 RUN pip3 install -r requirements.txt
5 COPY . .
6 EXPOSE 8000
```


و freeze کردن dependency ها در فایل requirements.txt مربوط به هر فولدر با کامند زیر (باید virtual env ها فعال باشند):

```
kheirmand@ubuntu: ~/Desktop/networking
kheirmand@ubuntu:~/Desktop/networking$ python3 -m pip freeze
```

میتوان image های مورد نظر را ساخت (من پس از نوشتن docker-compose با کامند زیر آن ها را ساخته ام)

Docker-compose build

در dockerfile ها کار خاصی انجام ندادم. صرفا working directory مربوط به image مشخص شده، تمامی dependency ها مثل psutil نصب شده و در نهایت تلم فایل ها که در اینجا صرفا agent.py یا agentServer.py هست به داخل image کپی شده.

برای server از انجایی که روی پورت ۸۰۰۰ قرار است metric ها باشند این پورت expose هم شده تا دیگر container ها در network ان بتوانند ان را ببینند.

برای docker-compose.yml که در فولدر روت قرار دارد، این نوشته شده:

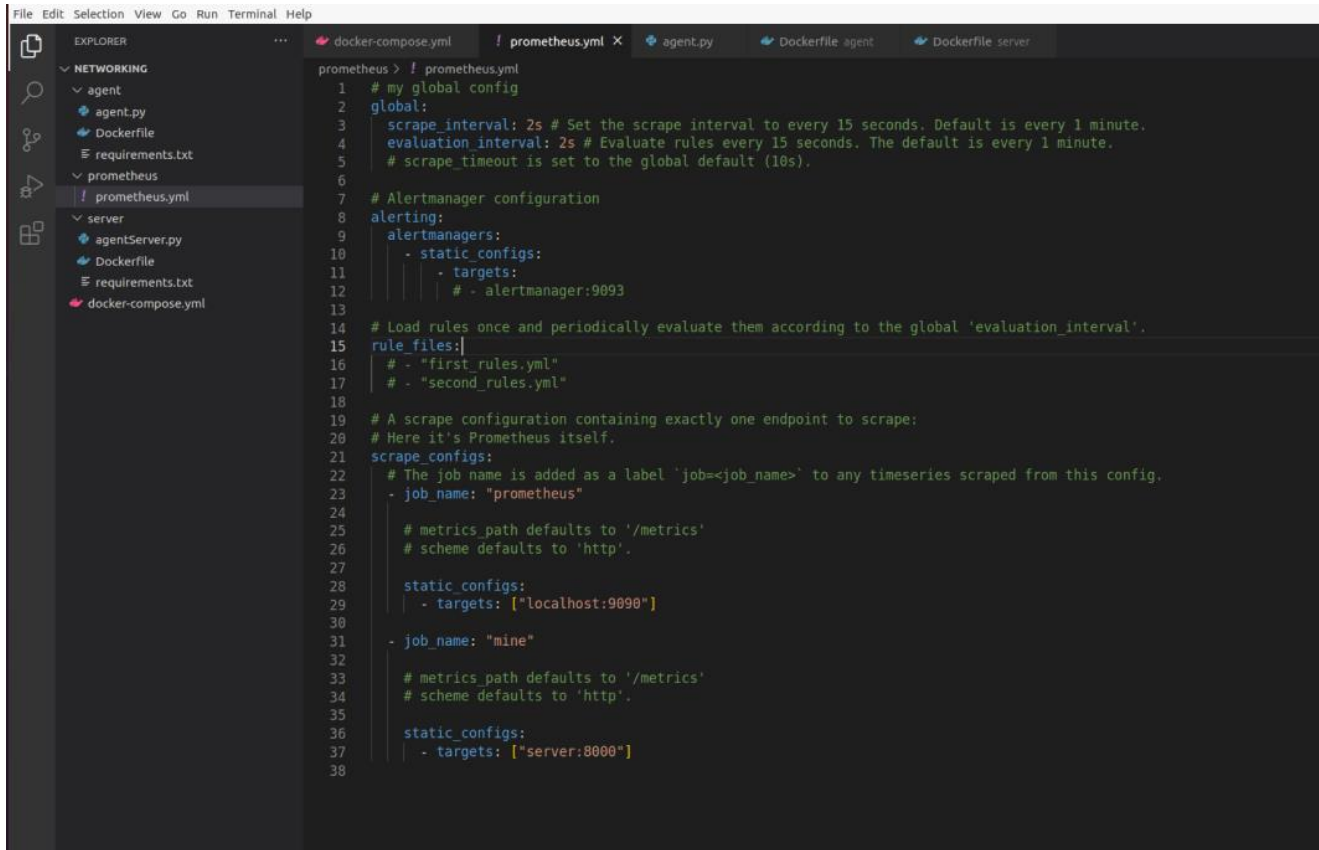
```
EXPLORER
NETWORKING
  agent
  agent.py
  Dockerfile
  requirements.txt
  prometheus
  prometheus.yml
  server
  agentServer.py
  Dockerfile
  requirements.txt
  docker-compose.yml
  docker-compose.yml
1 version: '3.7'
2
3 services:
4
5   server:
6     container_name: server
7     build:
8       dockerfile: Dockerfile
9       context: ./server
10    command: bash -c "python ./agentServer.py"
11    networks:
12      - network
13
14   prometheus:
15     image: prom/prometheus:latest
16     volumes:
17       - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
18     ports:
19       - "80:9090"
20     # links:
21     #   - server:localhost
22     networks:
23       - network
24
25   agent:
26     container_name: agent
27     build:
28       dockerfile: Dockerfile
29       context: ./agent
30     restart: always
31     command: bash -c "python ./agent.py"
32     networks:
33       - network
34     # depends_on:
35     #   - server
36
37 networks:
38   network:
```

سرویس ها به این صورت:

- سرویس server که همان agentServer است با dockerfile مشخص (image) و کامند python ./agentserver.py ران شده و در شبکه ی network قرار دارد.
- سرویس prometheus که image ان از docker-hub در هنگام up -d باید pull شود (آخرین stable version برای این کار انتخاب شده) و پورت ۹۰۹۰ از container که در ان web client مربوط به prometheus هست را به پورت ۸۰ از pc فرووارد کرده ایم تا با زدن localhost:80/ یا همان localhost/ در browser بتوان مشاهده کرد و چون باید بتواند server container را ببیند و به پورت ۸۰۰۰ ان دسترسی داشته باشد باید در همان شبکه باشد (network). در ضمن برای اینکه target ما را بشناسد (همان server که در پورت ۸۰۰۰ خود metric ها را نشان میدهد) باید در فایل تنظیمات ان یکسری تغییرات داد و همچنین من باز هم اینجا زمان ایدیت شدن metric ها را با استفاده از پارامترهای scrape_interval و evaluation_interval در این فایل تغییر دادم.
- سرویس agent که Dockerfile مربوط به ان در فولدر agent قرار دارد و با کامند python agent.py ران میشود و در همان network قرار دارد تا بتواند سرویس server را ببیند. میتواند مانند ران کردن در پیسی و طبق screenshot های بالا

چند تا از این سرویس را در docker-compose گذاشت چون سرور به صورت multi agent هم میتواند کار کند(که بالا تر نشان داده شد) ولی برای سادگی انجام نشده.

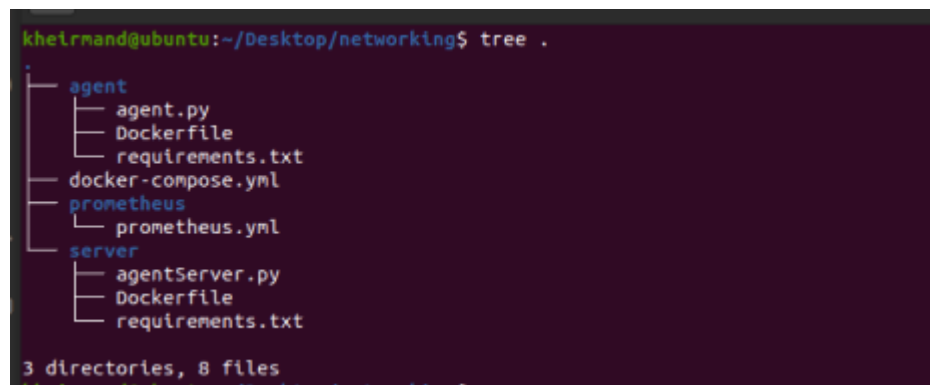
در پایان docker-compose هم شبکه ها ذکر شده که اینجا تنها یک شبکه با اسم network داریم. و در نهایت اینگونه شد:



```
prometheus > ! prometheus.yml
1 # my global config
2 global:
3   scrape_interval: 2s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
4   evaluation_interval: 2s # Evaluate rules every 15 seconds. The default is every 1 minute.
5   # scrape_timeout is set to the global default (10s).
6
7 # Alertmanager configuration
8 alerting:
9   alertmanagers:
10     - static_configs:
11       - targets:
12         # - alertmanager:9093
13
14 # Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
15 rule_files:
16   # - "first_rules.yml"
17   # - "second_rules.yml"
18
19 # A scrape configuration containing exactly one endpoint to scrape:
20 # Here it's Prometheus itself.
21 scrape_configs:
22   # The job name is added as a label 'job=<job_name>' to any timeseries scraped from this config.
23   - job_name: "prometheus"
24
25     # metrics_path defaults to '/metrics'
26     # scheme defaults to 'http'.
27
28     static_configs:
29       - targets: ["localhost:9090"]
30
31   - job_name: "mine"
32
33     # metrics_path defaults to '/metrics'
34     # scheme defaults to 'http'.
35
36     static_configs:
37       - targets: ["server:8000"]
38
```

اگر دقت کنید میبینید که فرق این فایل و prometheus.yml مربوط به prometheus ران شده در ویندوز (screenshot قبلی از فایل prometheus.yml) در خط آخر است. در انجا localhost:8000 و در اینجا server:8000 هست. چون localhost برای computer و ادرس 127.0.0.1 است ولی ما در شبکه ی network مربوط باید اسم سرویس را بزنیم که در اینجا server است.

درختواره ی پروژه:



```
kheirmand@ubuntu:~/Desktop/networking$ tree .
.
├── agent
│   ├── agent.py
│   ├── Dockerfile
│   └── requirements.txt
├── docker-compose.yml
├── prometheus
│   └── prometheus.yml
└── server
    ├── agentServer.py
    ├── Dockerfile
    └── requirements.txt

3 directories, 8 files
```

در نهایت build و up میکنیم:

```

kheirmand@ubuntu:~/Desktop/networking$ docker-compose build
prometheus uses an image, skipping
Building server
Sending build context to Docker daemon 7.68kB
Step 1/6 : FROM python:3.8.9
--> dd4ac8dff24c
Step 2/6 : WORKDIR /server
--> Using cache
--> 7ddea930d3cd
Step 3/6 : COPY requirements.txt requirements.txt
--> Using cache
--> c3b9f142e9bd
Step 4/6 : RUN pip3 install -r requirements.txt
--> Using cache
--> 98a98b0f2046
Step 5/6 : COPY . .
--> Using cache
--> 26b421655cbd
Step 6/6 : EXPOSE 8000
--> Using cache
--> 531ceb28a7e8
Successfully built 531ceb28a7e8
Successfully tagged networking_server:latest
Building agent
Sending build context to Docker daemon 5.632kB
Step 1/5 : FROM python:3.8.9
--> dd4ac8dff24c
Step 2/5 : WORKDIR /agent
--> Using cache
--> 9f2565d26cb7
Step 3/5 : COPY requirements.txt requirements.txt
--> Using cache
--> e0afb86e4acc
Step 4/5 : RUN pip3 install -r requirements.txt
--> Using cache
--> 6d74b852f19e
Step 5/5 : COPY . .
--> Using cache
--> ba443ee440af
Successfully built ba443ee440af
Successfully tagged networking_agent:latest
kheirmand@ubuntu:~/Desktop/networking$

```

```

kheirmand@ubuntu:~/Desktop/networking$ docker-compose up -d
agent is up-to-date
networking_prometheus_1 is up-to-date
server is up-to-date
kheirmand@ubuntu:~/Desktop/networking$

```

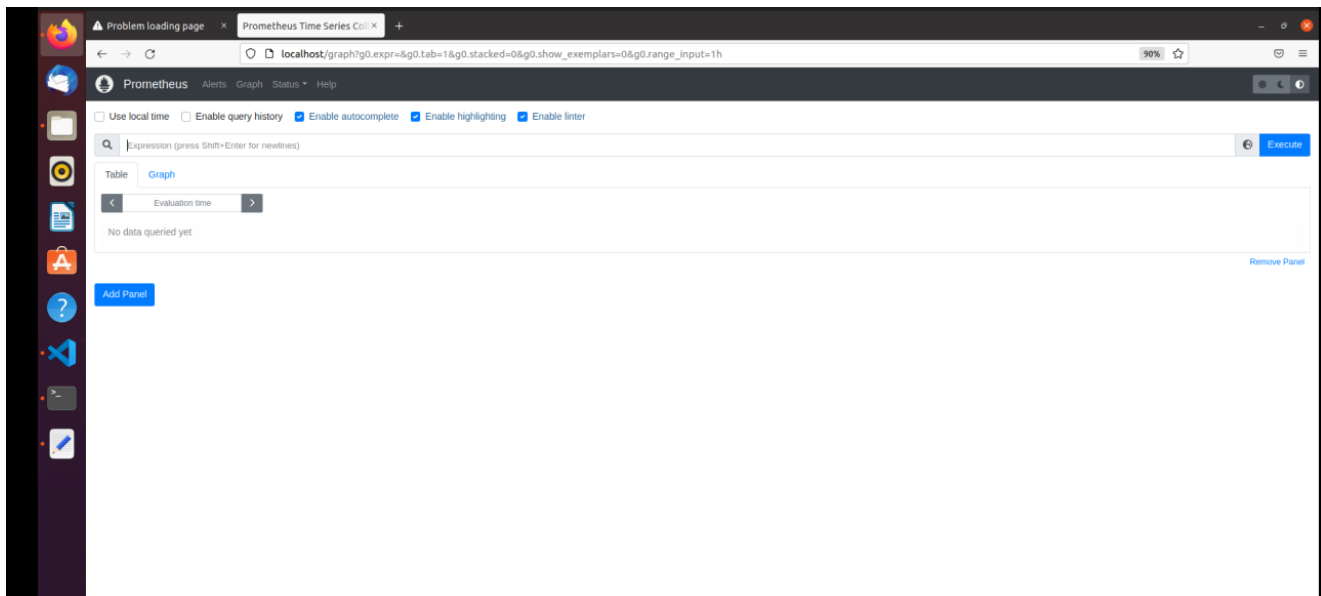
برای چک کردن اینکه کار میکنند باید ببینیم که container ها restart یا exit نشده باشند:

```

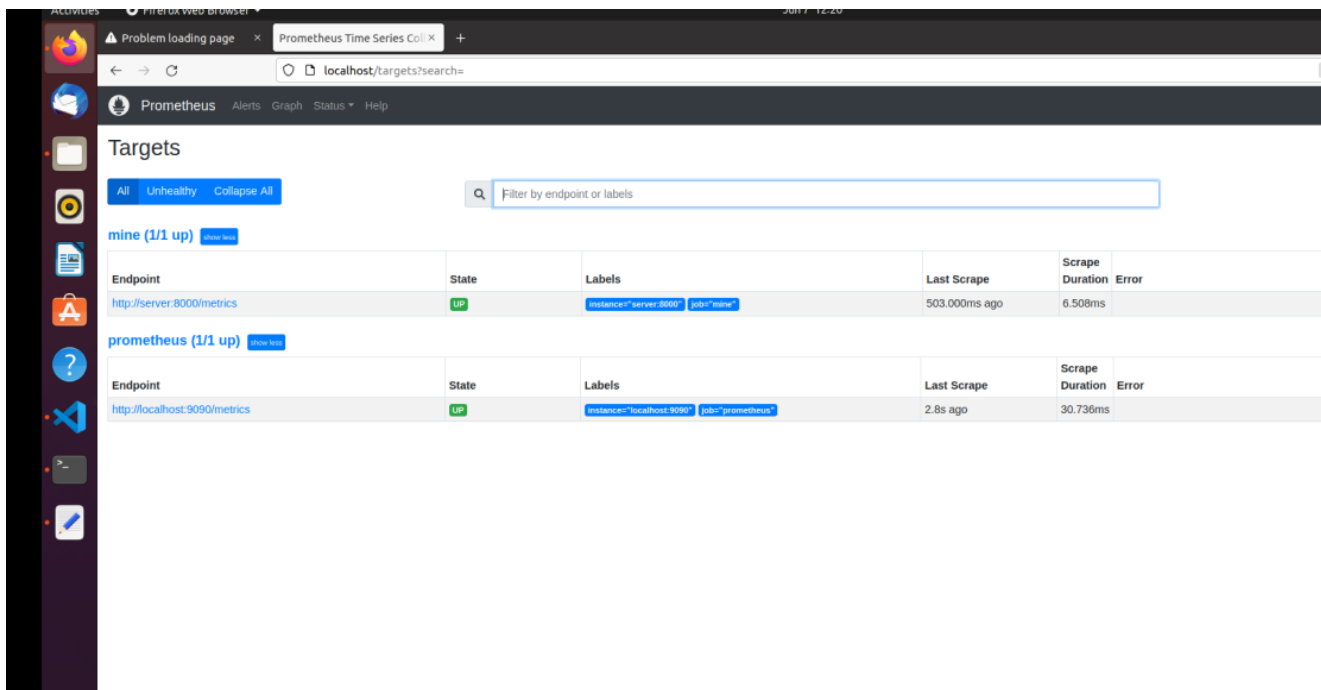
kheirmand@ubuntu:~/Desktop/networking$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
9d40eb65a516   prom/prometheus:latest              "/bin/prometheus --c..." 32 minutes ago Up 32 minutes 0.0.0.0:80->9090/tcp, :::80->9090/tcp networking_prometheus_1
3b5e0b204499   networking_server                    "bash -c 'python ./a..." 37 minutes ago Up 37 minutes 8000/tcp server
b80a8f7b6ced   networking_agent                     "bash -c 'python ./a..." 40 minutes ago Up 40 minutes agent
kheirmand@ubuntu:~/Desktop/networking$

```

برای دیدن prometheus باید به localhost رفت:



برای چک کردن اینکه به هم وصل شده اند به status و سپس target میرویم:



همانطور که میبینید کار میکند.

