Name: Aditya Sharma
Roll No.: CH21B006

DA5402: Assignment #6

File Structure:

```
assignment-root/
├── .github/            # GitHub configuration directory
├── .gitignore          # Git ignore rules
├── MLOps_A6_Report.pdf   # Assignment report document
├── README.md           # Project documentation
├── prometheus.yml       # Prometheus configuration for Task 3
└── task1_2.py          # Combined implementation of Tasks 1 and 2
```

Task 1 [20 points]

In the Linux environment, we have a command name iostat which lists the disk level IO statistics like the snapshot below.
We can run this command repeatedly using watch -n1 iostat to have a live monitoring from the command line. You may also redirect the output of iostat into a file for further processing using the command iostat > /tmp/io, which will appear like below.
You can process the text dump to collect the statistics. The objective here is to convert the collected statistics into suitable Prometheus metric types with appropriate names. And you repeat the data collection every second (using crontab or via scripting), so Prometheus service can pick the metrics at the scrape_interval >= 1s.
The metrics to be exported are:

- *io_read_rate{device=``xxx”}*
- *io_write_rate{device=``xxx”}*
- *io_tps{device=``xxx”}*
- *io_read_bytes{device=``xxx”}*
- *io_write_bytes{device=``xxx”}*
- *cpu_avg_percent{mode=``xxx”} xxx \in {user, nice, system, iowait, idle}*

Task 2 [20 points]

Likewise, we can read the current memory information from /proc/meminfo, which appears like below:

Note the, both tasks should be implemented as individual functions in a single Python script, which would expose the metrics at port 18000. So, http://localhost:18000/metrics should report all the exported metrics from both functions. As usual, use of logging mechanism is mandatory. Comments around the source code block is crucial.

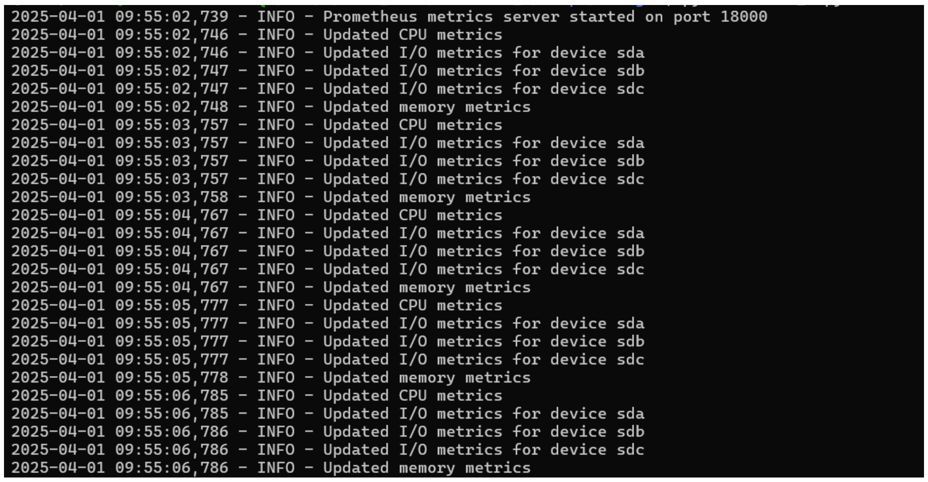
The code for both task 1 and task 2 has been implemented in a single python script with 2 individual functions. The name of the file is task1_2.py

Before running this code, the following dependencies will have to be installed:

- prometheus-client: *pip install prometheus-client*
- systat:
  I did the following in my ubuntu terminal to install systat:
  sudo apt update
  sudo apt install sysstat

Then, the file can be run using
python task1_2.py OR
python3 task1_2.py

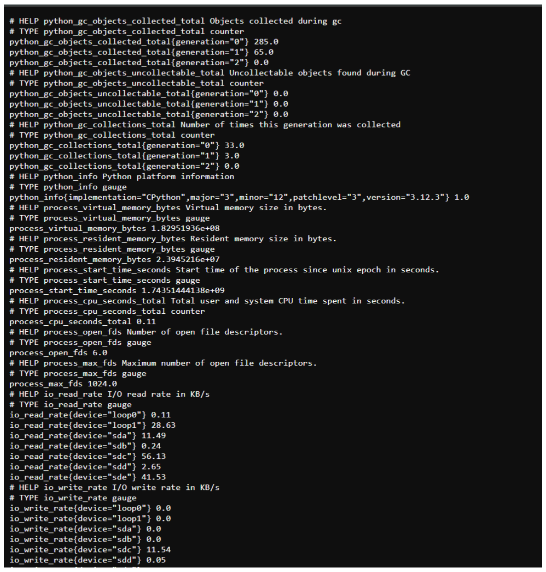
After running, the following types of logs should be visible in the terminal:



```
2025-04-01 09:55:02,739 - INFO - Prometheus metrics server started on port 18000
2025-04-01 09:55:02,746 - INFO - Updated CPU metrics
2025-04-01 09:55:02,746 - INFO - Updated I/O metrics for device sda
2025-04-01 09:55:02,747 - INFO - Updated I/O metrics for device sdb
2025-04-01 09:55:02,747 - INFO - Updated I/O metrics for device sdc
2025-04-01 09:55:02,748 - INFO - Updated memory metrics
2025-04-01 09:55:03,757 - INFO - Updated CPU metrics
2025-04-01 09:55:03,757 - INFO - Updated I/O metrics for device sda
2025-04-01 09:55:03,757 - INFO - Updated I/O metrics for device sdb
2025-04-01 09:55:03,757 - INFO - Updated I/O metrics for device sdc
2025-04-01 09:55:03,758 - INFO - Updated memory metrics
2025-04-01 09:55:04,767 - INFO - Updated CPU metrics
2025-04-01 09:55:04,767 - INFO - Updated I/O metrics for device sda
2025-04-01 09:55:04,767 - INFO - Updated I/O metrics for device sdb
2025-04-01 09:55:04,767 - INFO - Updated I/O metrics for device sdc
2025-04-01 09:55:04,767 - INFO - Updated memory metrics
2025-04-01 09:55:05,777 - INFO - Updated CPU metrics
2025-04-01 09:55:05,777 - INFO - Updated I/O metrics for device sda
2025-04-01 09:55:05,777 - INFO - Updated I/O metrics for device sdb
2025-04-01 09:55:05,777 - INFO - Updated I/O metrics for device sdc
2025-04-01 09:55:05,778 - INFO - Updated memory metrics
2025-04-01 09:55:06,785 - INFO - Updated CPU metrics
2025-04-01 09:55:06,785 - INFO - Updated I/O metrics for device sda
2025-04-01 09:55:06,786 - INFO - Updated I/O metrics for device sdb
2025-04-01 09:55:06,786 - INFO - Updated I/O metrics for device sdc
2025-04-01 09:55:06,786 - INFO - Updated memory metrics
```

Fig. 1: Terminal Status after correctly running python task1_2.py

Now, the following comes up if we go to localhost:18000/metrics



```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 285.0
python_gc_objects_collected_total{generation="1"} 65.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects 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"} 33.0
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="12",patchlevel="3",version="3.12.3"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.82951936e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.3945216e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.74351444138e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.11
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 6.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1024.0
# HELP io_read_rate I/O read rate in KB/s
# TYPE io_read_rate gauge
io_read_rate{device="loop0"} 0.11
io_read_rate{device="loop1"} 28.63
io_read_rate{device="sda"} 11.49
io_read_rate{device="sdb"} 0.24
io_read_rate{device="sdc"} 56.13
io_read_rate{device="sdd"} 2.65
io_read_rate{device="sde"} 41.53
# HELP io_write_rate I/O write rate in KB/s
# TYPE io_write_rate gauge
io_write_rate{device="loop0"} 0.0
io_write_rate{device="loop1"} 0.0
io_write_rate{device="sda"} 0.0
io_write_rate{device="sdb"} 0.0
io_write_rate{device="sdc"} 11.54
io_write_rate{device="sdd"} 0.05
```

Fig. 2: Small snippet of localhost:18000/metrics

Note that the image is only a snippet. It does not show everything present in localhost:18000/metrics.

Task 3 [10 points]

Setup Prometheus server and configure it to scrape from your instrumented application by setting the scrape interval to 2 seconds. Ensure that the metrics your exposed from the app are queryable from Prometheus UI console.

To complete the setup with Prometheus, a prometheus.yml file must be created.

The code for prometheus.yml is as follows:

```
global:
  scrape_interval: 2s       # Scrape metrics every 2 seconds
  evaluation_interval: 2s

scrape_configs:
  - job_name: "task1_2"
    static_configs:
      - targets: ["localhost:18000"]
```

To start the Prometheus service, I used port 9091 instead of 9090 to avoid conflicts with the other existing Prometheus service.

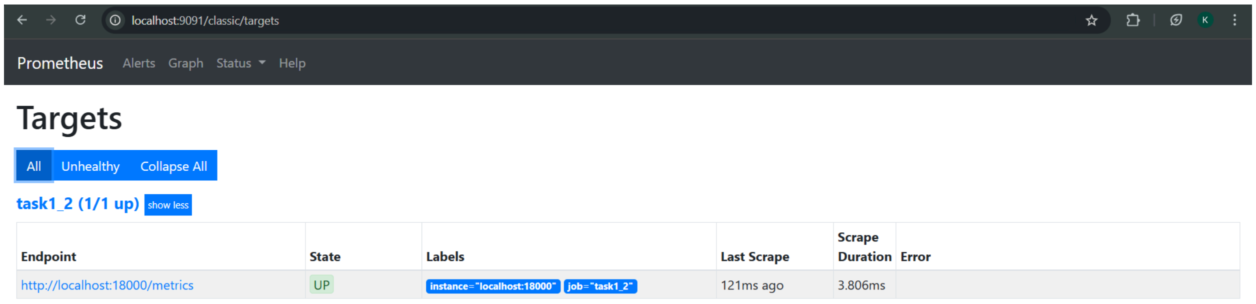
The command I used to start Prometheus in my project directory in the Ubuntu terminal is:

prometheus --config.file=prometheus.yml --storage.tsdb.path=~/prometheus_data --web.listen-address=:9091
(There could be other ways to start the Prometheus service for other systems. The above is how I did it.)

- The command tells Prometheus to use the configuration file prometheus.yml (via --config.file=prometheus.yml) and sets a dedicated storage directory (--storage.tsdb.path=~/prometheus_data) to persist its time-series data.
- It also sets the web interface to listen on port 9091 (using --web.listen-address=:9091) so that we can access the Prometheus UI and API on that port instead of the default 9090.

After running the command, we can navigate to localhost:9091 and navigate to different sections to view the functioning and results.

Below are some screenshots of my Prometheus dashboard sections:



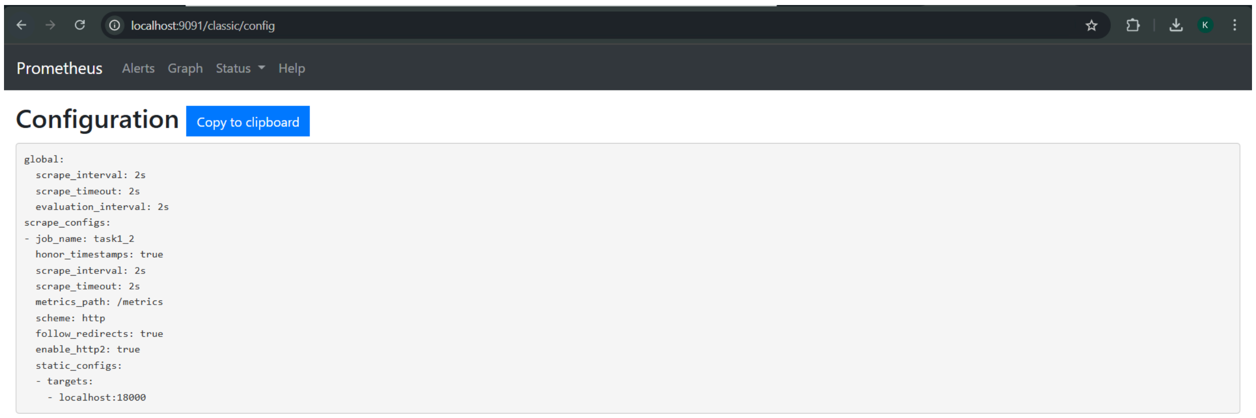
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:18000/metrics	UP	instance="localhost:18000" job="task1_2"	121ms ago	3.806ms	

Fig. 3: Prometheus Targets Section

The targets dashboard can be navigated through the UI in the Status drop-down menu. For me, it shows up at *localhost:9100/classic/targets*.

Prometheus targets dashboard showing successful integration with the custom disk I/O and memory metrics exporter (task1_2) running on port 18000, with an active UP status confirming the 2-second scrape interval is functioning properly.
The last scrape value always comes out to be less than 2 if this dashboard is refreshed after more than 2 seconds.

The scrape interval can also be confirmed by navigating to the Configuration section using the Status drop-down menu. For me, it shows up at *localhost:9100/classic/config*.

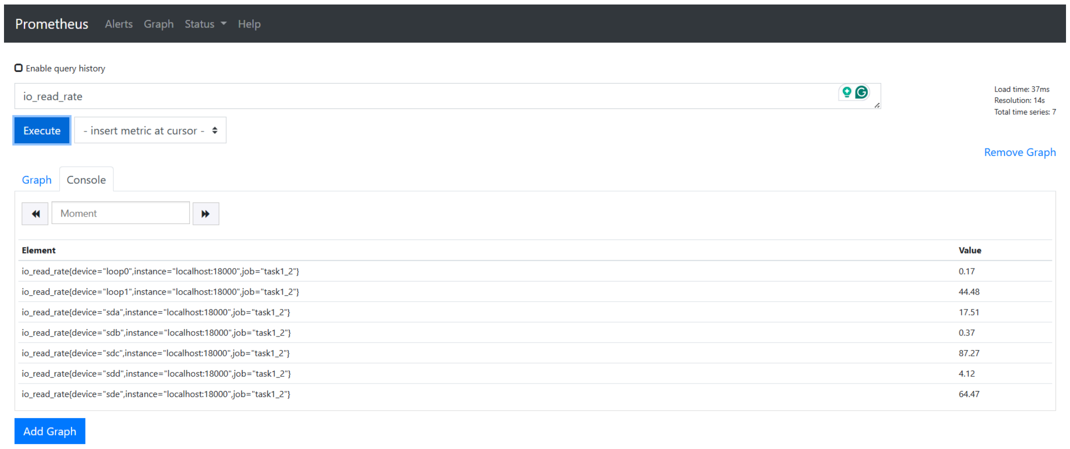


```
global:
  scrape_interval: 2s
  scrape_timeout: 2s
  evaluation_interval: 2s
scrape_configs:
- job_name: task1_2
  honor_timestamps: true
  scrape_interval: 2s
  scrape_timeout: 2s
  metrics_path: /metrics
  scheme: http
  follow_redirects: true
  enable_http2: true
  static_configs:
  - targets:
    - localhost:18000
```

Fig. 4: Prometheus Config Section

It can be seen that the scrape_interval is set to 2s.

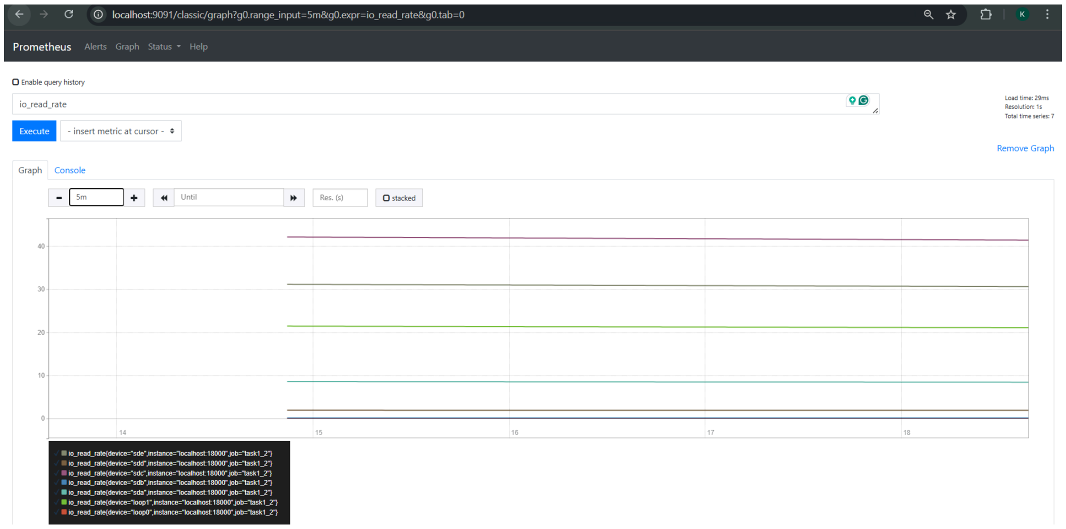
Now, by navigating to the Graph section, we can put queries for the metrics. Below are example images of me querying **io_read_rate** and **meminfo_total_bytes**. For io_read_rate, I have also shown the graph representation.



Load time: 37ms
Resolution: 14s
Total time series: 7

Element	Value
io_read_rate{device="loop0",instance="localhost:18000",job="task1_2"}	0.17
io_read_rate{device="loop1",instance="localhost:18000",job="task1_2"}	44.48
io_read_rate{device="sda",instance="localhost:18000",job="task1_2"}	17.51
io_read_rate{device="sdb",instance="localhost:18000",job="task1_2"}	0.37
io_read_rate{device="sdc",instance="localhost:18000",job="task1_2"}	87.27
io_read_rate{device="sdd",instance="localhost:18000",job="task1_2"}	4.12
io_read_rate{device="sde",instance="localhost:18000",job="task1_2"}	64.47

Fig. 5: io_read_rate console section in Graph



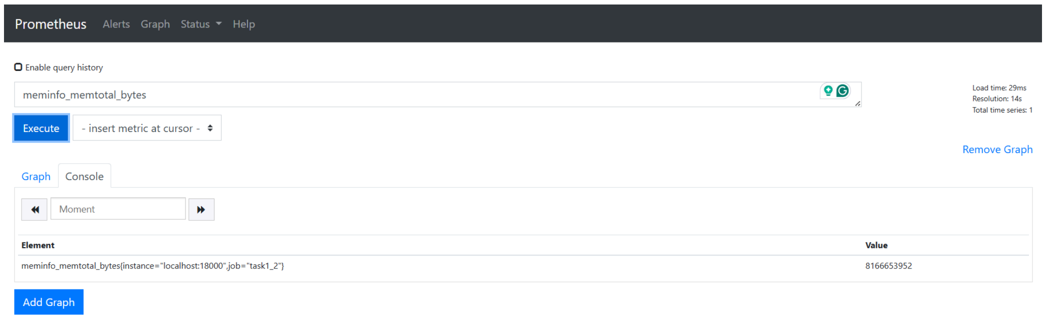
```
io_read_rate
```

Load time: 29ms
Resolution: 1s
Total time series: 7

5m

```
io_read_rate{device="sde",instance="localhost:18000",job="task1_2"}
io_read_rate{device="sdd",instance="localhost:18000",job="task1_2"}
io_read_rate{device="sdc",instance="localhost:18000",job="task1_2"}
io_read_rate{device="sdb",instance="localhost:18000",job="task1_2"}
io_read_rate{device="sda",instance="localhost:18000",job="task1_2"}
io_read_rate{device="loop1",instance="localhost:18000",job="task1_2"}
io_read_rate{device="loop0",instance="localhost:18000",job="task1_2"}
```

Fig. 6: io_read_rate graph section in Graph



```
meminfo_memtotal_bytes
```

Load time: 29ms
Resolution: 14s
Total time series: 1

Element	Value
meminfo_memtotal_bytes{instance="localhost:18000",job="task1_2"}	8166653952

Fig. 7: meiminfo_memtotal_bytes console section in Graph