

DA5402: Machine Learning Operations Laboratory

Assignment 6

Nandhakishore C S (DA24M011)

April 1, 2025

Problem Statement

Let's now learn to build an AI application with instrumentation using Prometheus. We learned how to use exporters to monitor standard application and about building an exporter for our application via Prometheus instrumentation library. Let's try creating an exporter for exporting the system (node) level metrics without using node_exporter or windows_exporter.

Task 1 [20 points]

Use the Linux iostat command to get the system's io and CPU statistics into suitable Prometheus metrics.

setup

- I am running all of my programs in a macbook and it doesn't support iostat command very well. For this assignment I have created a docker image and I am doing all my programs there.
1. Setting up a Ubuntu docker image with Python3 and copying the necessary files from local volume to docker
 2. For Task1, we need the following python libraries: p prometheus_client and subprocess. A requirements.txt file is created to install these when the image is created.
 3. Build the docker , name it as prometheus-exporter

```
$ docker build -t prometheus-exporter
```

4. Run the docker and do port mapping to 18000 (from question)

```
$ docker run -d --name prometheus-exporter -p 18000:18000 prometheus-exporter
```

5. The docker file is as follows:

```
FROM ubuntu:20.04 # Use a lightweight Ubuntu-based image

# Set environment variables to avoid interactive prompts during package installation
ENV DEBIAN_FRONTEND=noninteractive

# Install necessary Linux packages and Python 3.9
RUN apt-get update && apt-get install -y \
    python3.9 \
    python3-pip \
    sysstat # Provides the 'iostat' command \
    && rm -rf /var/lib/apt/lists/* # Clean up to reduce image size

# Set the working directory
WORKDIR /app

# Copy the script and requirements file into the container
COPY collect_metrics.py /app/prometheus_exporter.py
COPY requirements.txt /app/requirements.txt

# Install required Python packages
RUN python3.9 -m pip install -r requirements.txt

# Expose the Prometheus metrics port
EXPOSE 18000

# Run the script with Python 3.9
CMD ["python3.9", "/app/prometheus_exporter.py"]
```

The user defined function is as follows:

```
def collect_io_stats():
    try:
        # Execute iostat command with detailed output (-x) and CPU stats (-c)
        result = subprocess.run(['iostat', '-d', '-x', '-c'] capture_output=True, text=True, check=
True)

        logger.info("Successfully collected iostat data")
        logger.debug(f"Full iostat output:\n{result.stdout}")

        # Parse iostat output
        lines = result.stdout.splitlines()
        parsing_disk = False
        parsing_cpu = False

        # Track devices to ensure all metrics are set, even if no data
        devices_seen = set()

        for line in lines:
            if not line.strip():
                continue

            # Switch between disk and CPU sections
            if 'Device' in line:
                parsing_disk = True
                parsing_cpu = False
                continue
            elif 'avg-cpu' in line:
                parsing_disk = False
                parsing_cpu = True
                continue

            fields = line.split()

            # Parse disk statistics
            if parsing_disk and len(fields) >= 14:
                device = fields[0]
                devices_seen.add(device)
                read_rate = float(fields[2])      # r/s (reads per second)
                write_rate = float(fields[3])     # w/s (writes per second)
                read_kb = float(fields[4])        # kB/s
                write_kb = float(fields[5])       # kB/s
                tps = float(fields[13])           # tps

                logger.debug(f"Device {device}: r/s={read_rate}, w/s={write_rate}, kB/s={read_kb},
kB/s={write_kb}, tps={tps}")

                io_read_rate.labels(device=device).set(read_rate)
                io_write_rate.labels(device=device).set(write_rate)
                io_tps.labels(device=device).set(tps)
                io_read_bytes.labels(device=device).set(read_kb * 1024)
                io_write_bytes.labels(device=device).set(write_kb * 1024)

            # Parse CPU statistics
            if parsing_cpu and len(fields) >= 6:
                cpu_avg_percent.labels(mode='user').set(float(fields[0]))
                cpu_avg_percent.labels(mode='nice').set(float(fields[1]))
                cpu_avg_percent.labels(mode='system').set(float(fields[2]))
                cpu_avg_percent.labels(mode='iowait').set(float(fields[4]))
                cpu_avg_percent.labels(mode='idle').set(float(fields[5]))
                logger.debug(f"CPU stats: user={fields[0]}, nice={fields[1]}, system={fields[2]},
iowait={fields[4]}, idle={fields[5]}")

            # Ensure all metrics are set for all seen devices (default to 0 if not updated)
            for device in devices_seen:
                if not io_read_rate.labels(device=device)._value.get():
                    io_read_rate.labels(device=device).set(0)
                    logger.debug(f"Set io_read_rate for {device} to 0 as no value was parsed")
                if not io_write_rate.labels(device=device)._value.get():
                    io_write_rate.labels(device=device).set(0)
                    logger.debug(f"Set io_write_rate for {device} to 0 as no value was parsed")

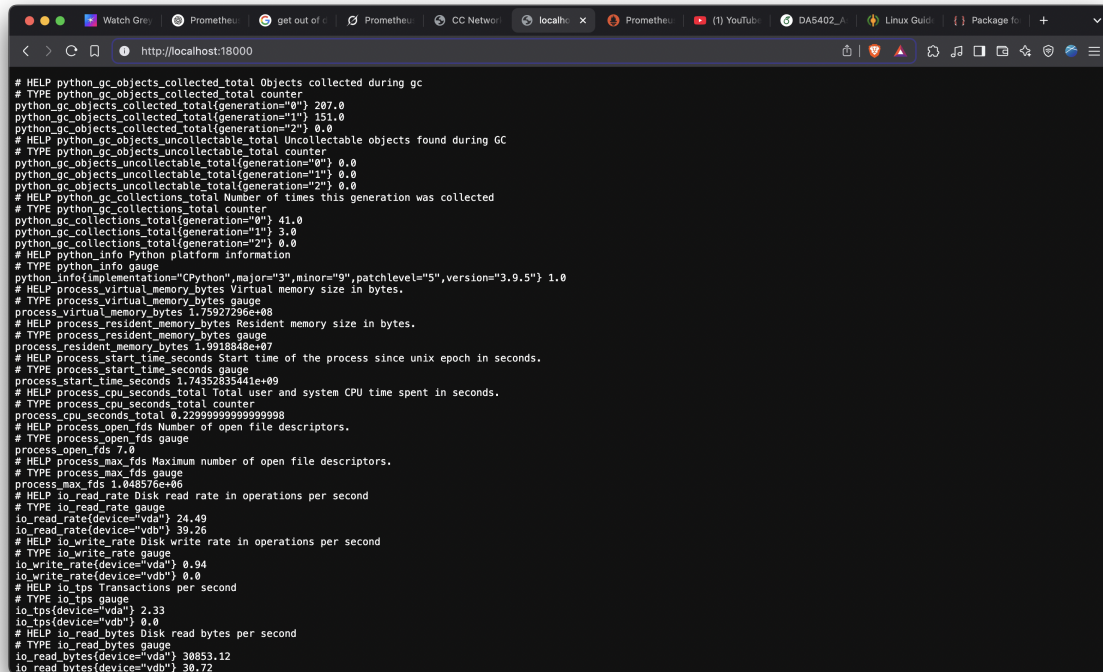
    except subprocess.CalledProcessError as e:
        logger.error(f"Failed to collect iostat data: {e}")
    except Exception as e:
        logger.error(f"Error processing iostat data: {e}")
```

The metrics are scrapped for every 2 seconds.

The metrics are named using Gauge Counter functions from prometheus client.

```
# Define Prometheus metrics for I/O statistics
io_read_rate = Gauge('io_read_rate', 'Disk read rate in operations per second', ['device'])
io_write_rate = Gauge('io_write_rate', 'Disk write rate in operations per second', ['device'])
io_tps = Gauge('io_tps', 'Transactions per second', ['device'])
io_read_bytes = Gauge('io_read_bytes', 'Disk read bytes per second', ['device'])
io_write_bytes = Gauge('io_write_bytes', 'Disk write bytes per second', ['device'])
cpu_avg_percent = Gauge('cpu_avg_percent', 'CPU utilization percentage', ['mode'])
```

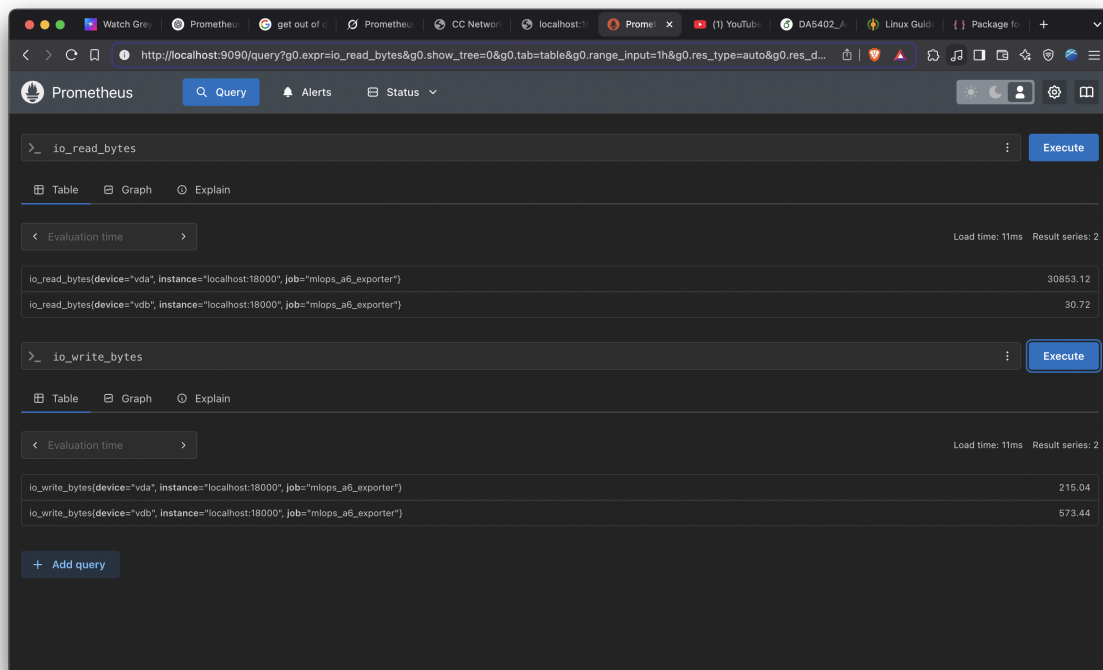
When the docker is started, the program is executed automatically and metrics are exposed.



```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 207.0
python_gc_objects_collected_total{generation="1"} 151.0
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"} 41.0
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="5",version="3.9.5"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.75927296e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 1.9918848e+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 1.743283544e+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 0.22999999999999998
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 7.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP io_read_rate Disk read rate in operations per second
# TYPE io_read_rate gauge
io_read_rate{device="vda"} 24.49
io_read_rate{device="vdb"} 39.26
# HELP io_write_rate Disk write rate in operations per second
# TYPE io_write_rate gauge
io_write_rate{device="vda"} 0.84
io_write_rate{device="vdb"} 0.0
# HELP io_tps Transactions per second
# TYPE io_tps gauge
io_tps{device="vda"} 2.33
io_tps{device="vdb"} 0.0
# HELP io_read_bytes Disk read bytes per second
# TYPE io_read_bytes gauge
io_read_bytes{device="vda"} 30853.12
io_read_bytes{device="vdb"} 30.72
```

Figure 1: Metrics exposed in localhost 18000

They are also visible in Prometheus query page.



The screenshot shows the Prometheus query page with the following data:

Query	Table	Graph	Explain				
<code>>_ io_read_bytes</code>							
Evaluation time: Load time: 11ms Result series: 2							
<table border="1"><thead><tr><th>io_read_bytes(device="vda", instance="localhost:18000", job="mlops_a6_exporter")</th><th>30853.12</th></tr><tr><th>io_read_bytes(device="vdb", instance="localhost:18000", job="mlops_a6_exporter")</th><th>30.72</th></tr></thead></table>				io_read_bytes(device="vda", instance="localhost:18000", job="mlops_a6_exporter")	30853.12	io_read_bytes(device="vdb", instance="localhost:18000", job="mlops_a6_exporter")	30.72
io_read_bytes(device="vda", instance="localhost:18000", job="mlops_a6_exporter")	30853.12						
io_read_bytes(device="vdb", instance="localhost:18000", job="mlops_a6_exporter")	30.72						
<code>>_ io_write_bytes</code>							
Evaluation time: Load time: 11ms Result series: 2							
<table border="1"><thead><tr><th>io_write_bytes(device="vda", instance="localhost:18000", job="mlops_a6_exporter")</th><th>215.04</th></tr><tr><th>io_write_bytes(device="vdb", instance="localhost:18000", job="mlops_a6_exporter")</th><th>573.44</th></tr></thead></table>				io_write_bytes(device="vda", instance="localhost:18000", job="mlops_a6_exporter")	215.04	io_write_bytes(device="vdb", instance="localhost:18000", job="mlops_a6_exporter")	573.44
io_write_bytes(device="vda", instance="localhost:18000", job="mlops_a6_exporter")	215.04						
io_write_bytes(device="vdb", instance="localhost:18000", job="mlops_a6_exporter")	573.44						

+ Add query

Figure 2: Metrics exposed localhost 9090 query page

Task 2 [20 points]

Like wise, scrape the memory from the `/proc/meminfo` page.

A user defined python function is written to parse the contents from the meminfo file and they are named appropriately to expose them as metrics

```
# Collect memory statistics from /proc/meminfo and update Prometheus metrics
def collect_meminfo_stats():
    try:
        with open('/proc/meminfo', 'r') as f:
            meminfo = f.read()

            logger.info("Successfully collected meminfo data")

        # Parse each line of meminfo
        for line in meminfo.splitlines():
            if not line.strip():
                continue

            # Extract metric name and value
            match = re.match(r'(\w+):\s+(\d+)(?:\s+kB)?', line)
            if match:
                metric_name, value = match.groups()
                # Convert to snake_case and add prefix
                metric_key = f"meminfo_{metric_name.lower()}"

                # Create gauge if it doesn't exist
                if metric_key not in meminfo_metrics:
                    meminfo_metrics[metric_key] = Gauge(
                        metric_key,
                        f'Memory statistic: {metric_name}',
                        []
                    )

                # Set the value (convert kB to bytes)
                meminfo_metrics[metric_key].set(float(value) * 1024)
                logger.debug(f"Set {metric_key} to {float(value) * 1024}")
```

The outputs are again exposed to metrics page in localhost with port number 18000 and also in the Prometheus UI (refer figure 1 and figure 2).

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 enable Prometheus to capture the metrics exposed from the script sitting inside the docker container, the `.yaml` file in the Prometheus binary folder is modified to add this task as follows.

```
global:
    scrape_interval: 2s

rule_files:
    - "rules.yml"

scrape_configs:
    - job_name: "mlops_a6_exporter"
      scrape_interval: 2s
      static_configs:
        - targets: ["localhost:18000"]
```

For running code, refer the readme file in the github repository.