Assignment 6 - System Metrics Exporter

1. Introduction

This document provides a detailed explanation of how to implement **System Metrics Exporter** using Prometheus. The project collects and exposes system metrics such as **CPU usage, memory usage, and disk IO statistics**. The Prometheus server is configured to scrape these metrics for monitoring.

2. Project Overview

Objective

- Collect system-level metrics using Python.
- Expose metrics in a format that Prometheus can scrape.
- Configure Prometheus to collect and visualize the metrics.

Technologies Used

- **Python 3.x**
- **Prometheus**
- **Prometheus Client Library**
- **psutil** (for system resource monitoring)
- **Logging Module** (for error handling and debugging)

3. File Structure

```
Assignment6/
│-- system_metrics_exporter.py  # Python script to collect and expose system metrics
│-- prometheus.yml  # Prometheus configuration file
│-- README.md  # Readme file explaining the assignment
│-- documentation.pdf  # Detailed documentation of the implementation
```

4. Implementation Details

4.1 System Metrics Exporter

The script **system_metrics_exporter.py** collects system statistics and exposes them via an HTTP endpoint for Prometheus to scrape.

Metrics Collected:

1. **Disk IO Statistics** (using iostat command):
   - io_read_rate: Disk read rate (reads per second)
   - io_write_rate: Disk write rate (writes per second)
   - io_tps: Transfers per second
   - io_read_bytes: Total bytes read
   - io_write_bytes: Total bytes written
2. **CPU Usage** (using psutil.cpu_times_percent()):
   - cpu_avg_percent with different modes: user, system, idle, wait
3. **Memory Statistics** (from /proc/meminfo):
   - mem_total, mem_free, mem_available, mem_buffers, mem_cached
   - Swap Memory: swap_total, swap_free, swap_cached

How It Works:

- The script starts an HTTP server on port **18000** using prometheus_client.start_http_server().
- It continuously collects the system metrics every **1 second** and updates the Prometheus **Gauge** objects.
- Logging is implemented to track errors and successful metric collection.

4.2 Prometheus Configuration

The **prometheus.yml** file contains the configuration for Prometheus to scrape the exposed metrics:

```
global:
  scrape_interval: 15s  # Default scrape interval
scrape_configs:
  - job_name: 'custom_metrics'
    scrape_interval: 2s  # Setting the scrape interval to 2 seconds
    static_configs:
      - targets: ['localhost:18000']
```

- **Scrape interval** is set to **2 seconds** for high-frequency data collection.
- Prometheus scrapes metrics from http://localhost:18000/metrics.

5. Running the System

5.1 Prerequisites

Ensure that **Python 3.x** and **Prometheus** are installed. Install required Python packages:

```
pip install psutil prometheus_client
```

5.2 Running the Exporter

Start the Python exporter script:

```
python system_metrics_exporter.py
```

Check if the metrics are available:

```
curl http://localhost:18000/metrics
```

5.3 Running Prometheus

Start the Prometheus server with the provided configuration file:

```
prometheus --config.file=prometheus.yml
```

Access the Prometheus web interface:

http://localhost:9090

Query collected metrics like:

io_read_rate
cpu_avg_percent
mem_available

6. Logging and Error Handling

- **Logging is implemented at each step** to ensure proper debugging.

- Errors in metric collection are logged with logging.error().
- Successful metric updates are logged with logging.info().

7. Conclusion

This assignment successfully implements a **custom system metrics exporter** using Prometheus. The system collects, exposes, and allows Prometheus to scrape and visualize metrics efficiently.