Prometheus Assignment

Name:Hemanth K G
Emp id: TAS288

Question:

- Export the metrics (like request per second, memory usage, cpu usage etc) in the existing mini project given to Interns
- Install Prometheus and Grafana using Docker (with docker-compose)
- Configure prometheus (scrape configs) such way that it can scrape the metrics from default metric path of the application job
- Validate the entire configuration to check if the data is coming or not in Prometheus UI
- Create the Dashboards in Grafana on top of the metrics exported by adding the Prometheus as a Datasource.

Project Directory:

```
prometheus_assignment/
├── templates/
│   ├── bucket.html
│   └── index.html
├── app.py
├── requirements.txt
├── prometheus.yml
├── docker-compose.yml
└── Dockerfile
```

App.py

```
import os
import time
from urllib.parse import unquote
import boto3
from dotenv import load_dotenv
from flask import Flask, request, render_template, redirect, url_for, flash
from prometheus_client import Counter, Summary, make_wsgi_app
from werkzeug.middleware.dispatcher import DispatcherMiddleware
app = Flask(__name__)
app.secret_key = "AWS_SECRET_ACCESS_KEY"
```

```
load_dotenv()

AWS_ACCESS_KEY = os.getenv('AWS_ACCESS_KEY_ID')
AWS_SECRET_KEY = os.getenv('AWS_SECRET_ACCESS_KEY')

s3_client = boto3.client(
    's3',
    aws_access_key_id=AWS_ACCESS_KEY,
    aws_secret_access_key=AWS_SECRET_KEY,
)
# Metrics
REQUEST_COUNT = Counter('flask_requests_total', 'Total HTTP requests', ['method',
'endpoint'])
REQUEST_LATENCY = Summary('flask_request_latency_seconds', 'Request latency in
seconds')
app.wsgi_app = DispatcherMiddleware(
    app.wsgi_app,
    {'/metrics': make_wsgi_app()}
)

@app.before_request
def before_request():
    request.start_time = time.time()

@app.after_request
def after_request(response):
    latency = time.time() - request.start_time
    REQUEST_LATENCY.observe(latency)
    REQUEST_COUNT.labels(method=request.method, endpoint=request.path).inc()
    return response

@app.route('/')
def index():
    buckets = s3_client.list_buckets().get('Buckets')
    return render_template('index.html', buckets=buckets)

@app.route('/bucket/<bucket_name>')
def bucket_content(bucket_name):
    objects = s3_client.list_objects_v2(Bucket=bucket_name).get('Contents', [])
    buckets = s3_client.list_buckets().get('Buckets')
    return render_template('bucket.html', bucket_name=bucket_name, objects=objects,
buckets=buckets)
```

```
@app.route('/create_bucket', methods=['POST'])
def create_bucket():
    bucket_name = request.form['bucket_name']
    try:
        s3_client.create_bucket(Bucket=bucket_name)
        flash(f"Bucket {bucket_name} created successfully!", "success")
    except Exception as e:
        flash(str(e), "danger")
    return redirect(url_for('index'))

@app.route('/delete_bucket/<bucket_name>', methods=['POST'])
def delete_bucket(bucket_name):

    try:
        s3_client.delete_bucket(Bucket=bucket_name)
        flash(f"Bucket {bucket_name} deleted successfully!", "success")
    except Exception as e:
        flash(str(e), "danger")
    return redirect(url_for('index'))

@app.route('/upload_file/<bucket_name>', methods=['POST'])
def upload_file(bucket_name):

    file = request.files['file']
    if file:
        try:
            s3_client.upload_fileobj(file, bucket_name, file.filename)
            flash(f"File {file.filename} uploaded successfully!", "success")
        except Exception as e:
            flash(str(e), "danger")
    return redirect(url_for('bucket_content', bucket_name=bucket_name))

@app.route('/delete_file/<bucket_name>/<file_key>', methods=['POST'])
def delete_file(bucket_name, file_key):

    try:
        s3_client.delete_object(Bucket=bucket_name, Key=file_key)
        flash(f"File {file_key} deleted successfully!", "success")
    except Exception as e:
        flash(str(e), "danger")
    return redirect(url_for('bucket_content', bucket_name=bucket_name))

@app.route('/create_folder/<bucket_name>', methods=['POST'])
def create_folder(bucket_name):
    folder_name = request.form['folder_name']
    if not folder_name.endswith('/'):
        folder_name += '/'
    try:
        s3_client.put_object(Bucket=bucket_name, Key=folder_name)
        flash(f"Folder {folder_name} created successfully!", "success")
    except Exception as e:
        flash(str(e), "danger")
    return redirect(url_for('bucket_content', bucket_name=bucket_name))

@app.route('/delete_folder/<bucket_name>/<path:folder_key>', methods=['POST'])
def delete_folder(bucket_name, folder_key):
    """Delete a folder (prefix) from an S3 bucket."""

    folder_key = unquote(folder_key)
    try:

        s3_client.delete_object(Bucket=bucket_name, Key=folder_key)
        flash(f"Folder {folder_key} deleted successfully!", "success")
    except Exception as e:
        flash(str(e), "danger")
    return redirect(url_for('bucket_content', bucket_name=bucket_name))

@app.route('/copy_file/<source_bucket>/<source_key>/<target_bucket>', methods=['POST'])
def copy_file(source_bucket, source_key, target_bucket):
    """Copy a file from one bucket to another."""
    target_bucket = request.form['target_bucket']
    try:
        copy_source = {'Bucket': source_bucket, 'Key': source_key}
        s3_client.copy(copy_source, target_bucket, source_key)
        flash(f"File {source_key} copied to {target_bucket} successfully!", "success")
    except Exception as e:
        flash(str(e), "danger")
    return redirect(url_for('bucket_content', bucket_name=source_bucket))

@app.route('/move_file/<source_bucket>/<source_key>/<target_bucket>', methods=['POST'])
def move_file(source_bucket, source_key, target_bucket):
    """Move a file from one bucket to another."""
    target_bucket = request.form['target_bucket']
    try:

        copy_source = {'Bucket': source_bucket, 'Key': source_key}
        s3_client.copy(copy_source, target_bucket, source_key)
        s3_client.delete_object(Bucket=source_bucket, Key=source_key)
        flash(f"File {source_key} moved to {target_bucket} successfully!", "success")
    except Exception as e:
        flash(str(e), "danger")
    return redirect(url_for('bucket_content', bucket_name=source_bucket))

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Docker-compose.yml:

```
version: "3.8"
networks:
  app_network:
    driver: bridge

services:
  python_app:
    build:
      context: .
    networks:
      - app_network
    ports:
      - "5000:5000"

  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    networks:
      - app_network
    ports:
      - "9090:9090"

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin
    ports:
      - "3000:3000"
    networks:
      - app_network
    volumes:
      - grafana-data:/var/lib/grafana

volumes:
  grafana-data:Docker-compose.yml:
```

Dockerfile:

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

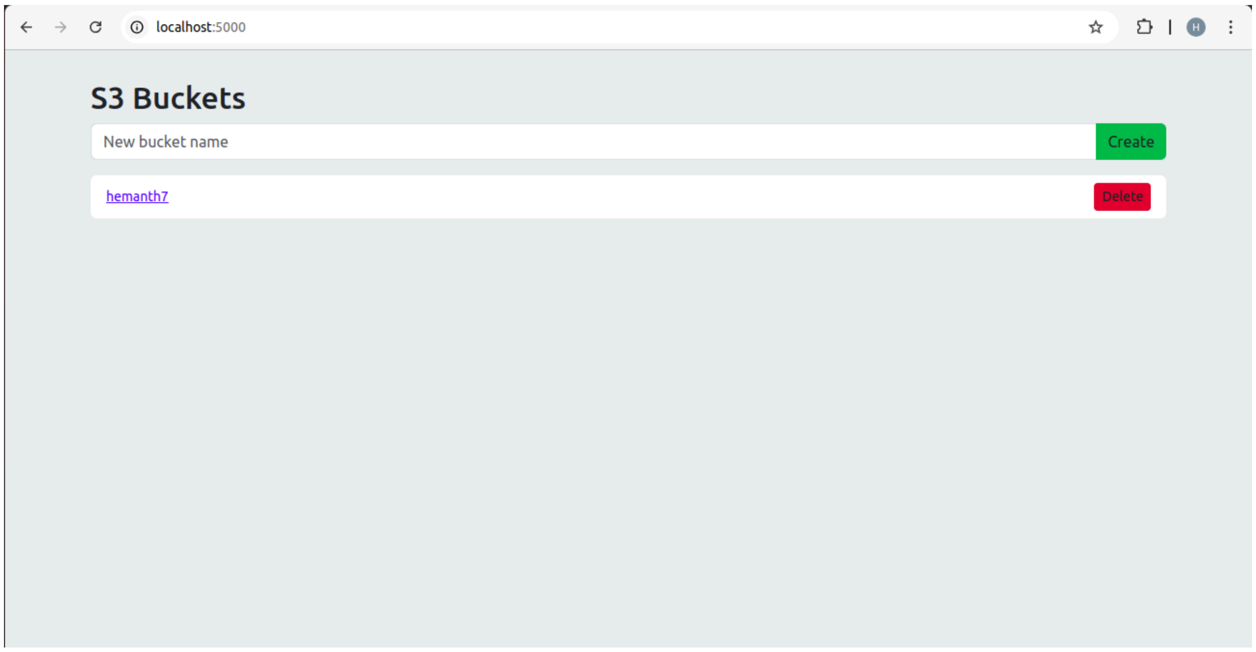
Prometheus.yml:

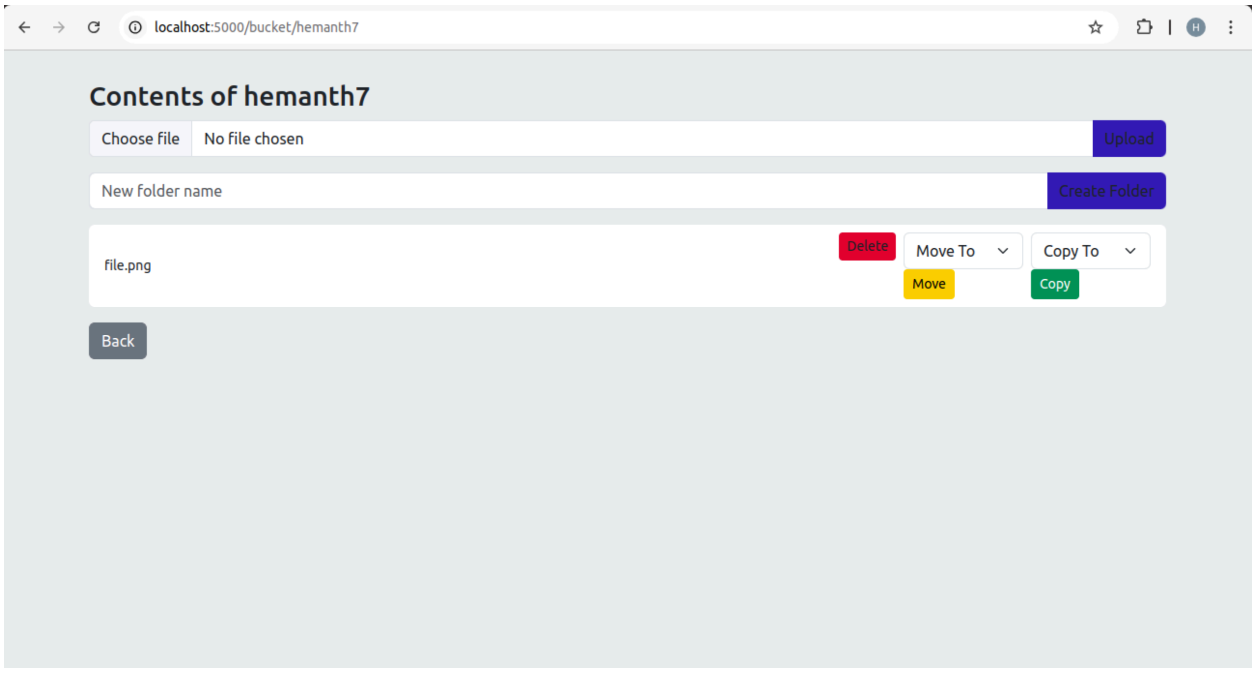
```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'python_app'
    static_configs:
      - targets: ['python_appflask:5000']
```

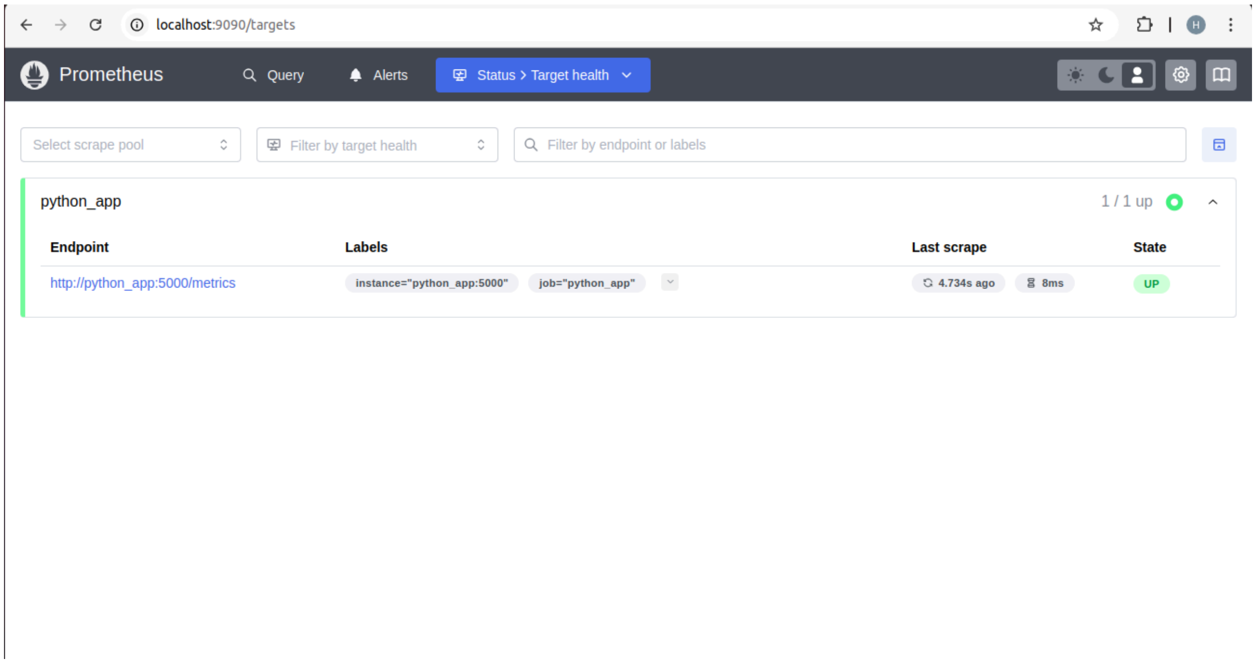
Requirements.txt:

```
flask
prometheus-client
python-dotenv
boto3
```

Screenshots:

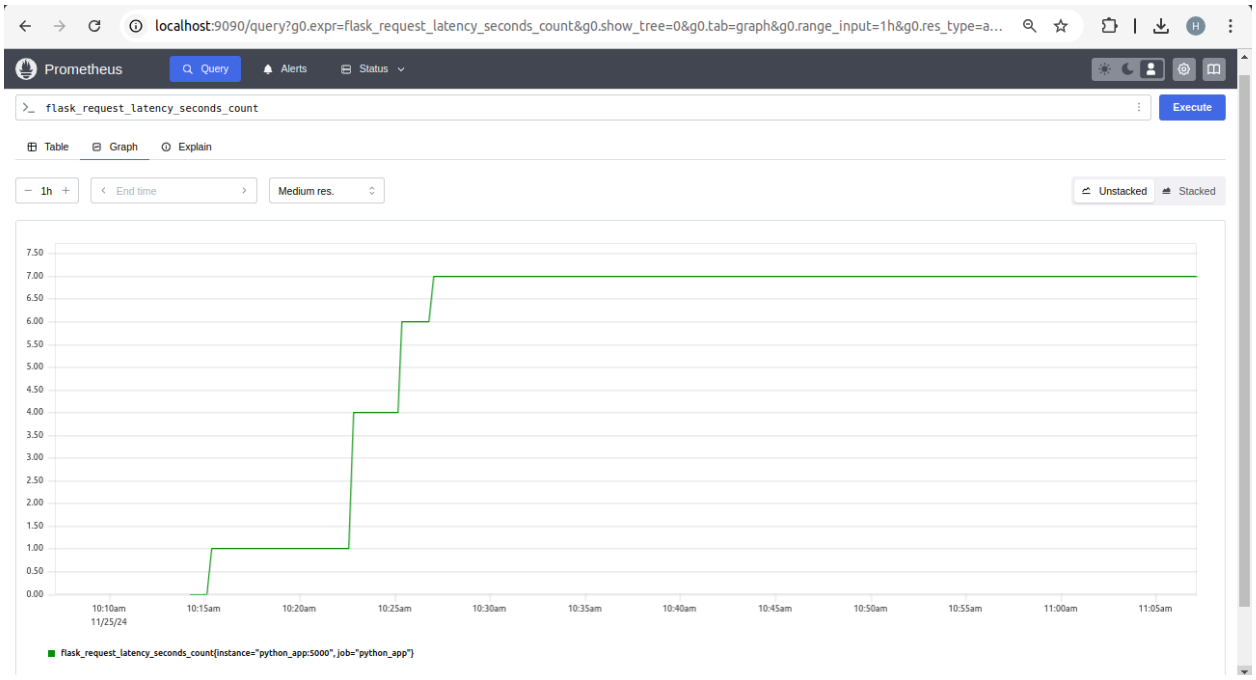






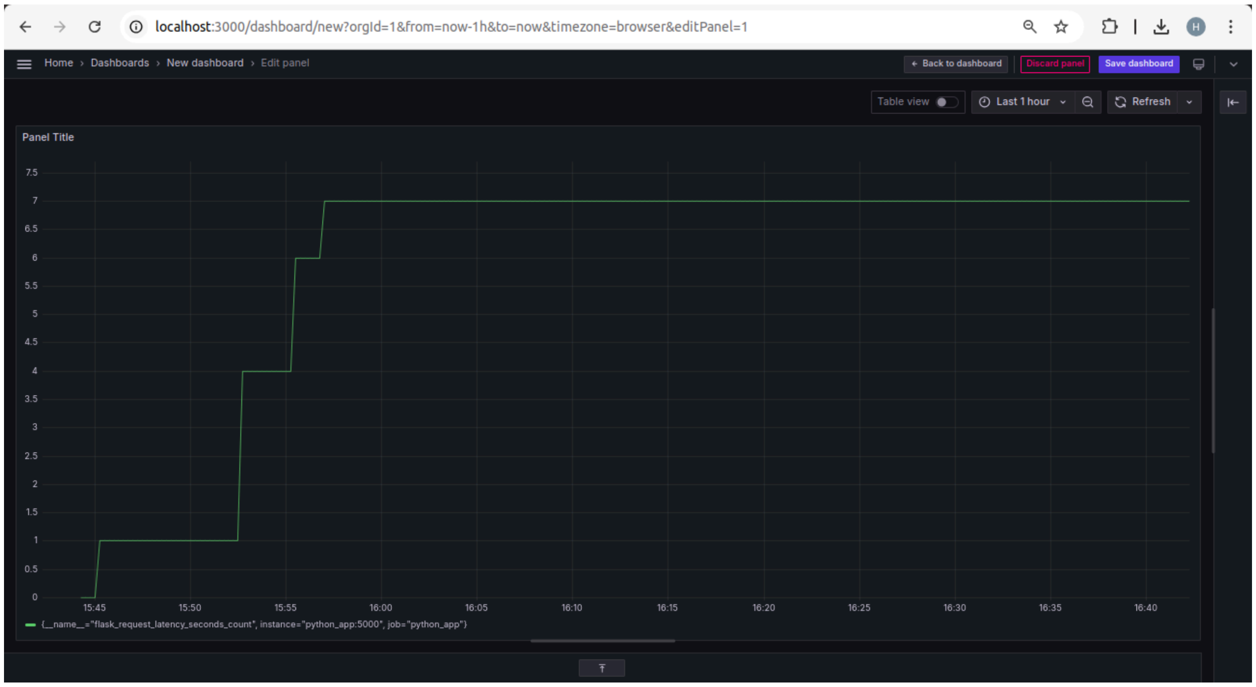
python_app 1 / 1 up

Endpoint	Labels	Last scrape	State
http://python_app:5000/metrics	instance="python_app:5000" job="python_app"	4.734s ago 8ms	UP



```
flask_request_latency_seconds_count
```

🟩 flask_request_latency_seconds_count{instance="python_app:5000", job="python_app"}


localhost:3000/dashboard/new?orgId=1&from=now-1h&to=now&timezone=browser&editPanel=1