Finubit - observability lab

Objective

The purpose of this test is to evaluate your ability to set up monitoring, collect relevant metrics, and visualize service health in a meaningful way. This test consists of setting up Prometheus and Grafana, instrumenting the given services, creating dashboards, and designing alert strategies.

Background

You will be working with a simple banking system architecture consisting of two services.

Front Service

Represents the customer-facing frontend of the bank.
Exposes two APIs:

1. Deposit:
   Allows customers to deposit money into their account.

```
curl --location 'http://localhost:5000/deposit' \
--header 'Content-Type: application/json' \
--data '{
  "amount":100
}'
```

2. Withdraw:
Allows customers to withdraw money from their account.

```
curl --location 'http://localhost:5000/withdraw' \
--header 'Content-Type: application/json' \
--data '{
  "amount":100
}'
```

Note: The withdraw API is intentionally slow to simulate real-world processing delays.

Core Service

Represents the backend of the bank, which is not directly accessible to customers.
Handles the actual business logic for deposit and withdrawal operations.
Both services are containerized using Docker and can be run locally using Docker Compose.

Preparation

1. Make sure to install the required software:
   - Python
   - Docker & Docker Compose
   - Prometheus & Grafana (optional if running via Docker)
2. Clone the code from the repository:

   ```
   git clone https://github.com/barkai36/finubit-observ-lab
   ```

3. Change to the project directory:

   ```
   cd finubit-observ-lab
   ```

4. Run the services using Docker Compose:

   ```
   docker-compose up --build
   ```

Tasks

1. Install Prometheus and Grafana

- Install Prometheus and Grafana on your local machine.
- You may use Docker to simplify installation.
- Configure Prometheus to scrape metrics from the services.

2. Modify the Services for Metrics Collection

- Add Prometheus instrumentation to the `front` and `core` services.
- Expose key metrics such as:

    - Number of requests
    - Response times
    - Error rates (status codes >= 400)
    - Processing times (especially for withdraw which is intentionally slow)
- Ensure Prometheus can scrape these metrics.

3. Create a Grafana Dashboard

- Create a Grafana dashboard with at least **4 visual elements**:
    1. **API Response Times** – Display average, P90, and P99 response times.
    2. **Error Rate** – Show percentage of errors out of total requests.
    3. **Request Throughput** – Number of requests per second.
    4. **Service Latency Comparison** – Compare latency between deposit and withdraw.
- Export the dashboard as JSON file.

4. Detecting Slow APIs (Latency > 2000ms)

- Explain how you would use the dashboard to detect APIs that take longer than 2000ms.
- Describe which chart or metric would indicate the issue.

5. Detecting High Error Rates (>30%)

- Explain how you would use the dashboard to detect when an API has an error rate exceeding 30%.
- Describe which metric(s) and visualizations would highlight this issue.

6. Define 5 Alerts for Service Monitoring

- Write down 5 theoretical alerts to monitor service health. Examples:
- Choose **one** of the alerts and describe a **runbook** to NOC, how to investigate and troubleshoot the issue.

Submission Guidelines

- Provide a GitHub repo with:
    - Your modified front and core services with Prometheus instrumentation.
    - docker-compose.yml (if using Docker).

    - Screenshots of your Grafana dashboard.
    - The JSON file of the exported dashboard.
    - A markdown file answering the theoretical questions.
    -
- Ensure the README explains how to set up and run everything.

Good luck!