Bookstore API with Prometheus Monitoring

This project demonstrates a Flask-based Bookstore API with Prometheus monitoring and Grafana visualization. It consists of multiple containerized services orchestrated with Docker Compose.

Project Overview

- **Bookstore API**: A RESTful Flask application for managing books
- **Prometheus**: Monitoring system and time series database
- **Node Exporter**: Exposes system metrics for Prometheus
- **Grafana**: Visualization platform for monitoring data

Prerequisites

- Docker and Docker Compose installed

Project Structure

```
assignment-2/
│
├── bookstore_api/
│   ├── app.py              # Flask application with endpoints and Prometheus metrics
│   ├── Dockerfile          # Docker configuration for the API
│   └── requirements.txt    # Python dependencies
│
├── docker-compose.yml      # Docker Compose configuration
├── prometheus.yml          # Prometheus configuration
├── prometheus-data/        # Prometheus data persistence (created on startup)
├── grafana-data/           # Grafana data persistence (created on startup)
└── README.md               # This file
```

Running the Application

1. Start the services:

```
docker-compose up -d
```

This command builds the Flask API image and starts all containers in detached mode.

2. Verify services are running:

```
docker-compose ps
```

Accessing Services

- **Bookstore API**: http://localhost:8080/books
- **Prometheus**: http://localhost:9090
- **Grafana**: http://localhost:5000
    - Default credentials: admin/admin
    - You'll be prompted to change the password on first login

API Endpoints

Endpoint	Method	Description
`/books`	GET	List all books
`/books/<id>`	GET	Get a specific book
`/books`	POST	Add a new book
`/books/<id>`	PUT	Update a book
`/books/<id>`	DELETE	Delete a book

Endpoint	Method	Description

Using the API in a Web Browser

1. **View all books**:
   - Simply open your browser and navigate to: http://localhost:8080/books
   - You'll see a JSON response listing all available books
2. **View a specific book**:
   - Navigate to http://localhost:8080/books/1 to see details of book with ID 1
   - The browser will display the book's information in JSON format
3. **Add, Update, or Delete books**: Since browsers primarily use GET requests when navigating to URLs, for other HTTP methods (POST, PUT, DELETE), you'll need one of these options:

   Option 1: Browser Extensions

   - Install a REST client extension like "RESTer" or "Talend API Tester" for Chrome/Firefox
   - Create a new request in the extension
   - Set the appropriate URL (e.g., http://localhost:8080/books)
   - Select the HTTP method (POST, PUT, or DELETE)
   - For POST and PUT, add a JSON body like:

   ```
   {
     "id": 3,
     "title": "Designing Data-Intensive Applications",
     "author": "Martin Kleppmann",
     "price": 38.0
   }
   ```

   - Set Content-Type header to "application/json"
   - Send the request

   Option 2: Online REST Clients

   - Use online tools like Swagger UI, Postman Web, or ReqBin
   - Configure and send your requests as described in Option 1

   Option 3: Developer Tools

   - Open your browser's Developer Tools (F12 or Ctrl+Shift+I)
   - Go to the Console tab
   - Use the fetch API to make requests, for example:

   ```
   // Add a new book
   fetch("http://localhost:8080/books", {
     method: "POST",
     headers: { "Content-Type": "application/json" },
     body: JSON.stringify({
       id: 3,
       title: "Designing Data-Intensive Applications",
       author: "Martin Kleppmann",
       price: 38.0,
     }),
   })
     .then((res) => res.json())
     .then(console.log);
   ```

Setting up Monitoring

Prometheus

Prometheus is automatically configured to scrape metrics from the Flask API and Node Exporter.

To verify it's collecting data:

1. Open http://localhost:9090 in your browser
2. Go to Status > Targets to confirm both targets are up
3. Try a query like `flask_http_requests_total` in the query box and click "Execute"
4. You can view the results as a graph or in table format

Useful Prometheus Queries

Try these PromQL queries in the Prometheus web interface:

Flask API Metrics:

- `flask_exporter_info`: Information about the Flask application
- `flask_http_request_duration_seconds_count`: Total count of HTTP requests

- `flask_http_request_duration_seconds_sum` : Sum of request durations
- `flask_http_request_duration_seconds_bucket` : Request duration histogram buckets
- `flask_http_request_exceptions_total` : Count of exceptions raised during requests
- `rate(flask_http_request_duration_seconds_count[5m])` : Request rate over 5 minutes
- `sum by (status) (rate(flask_http_request_duration_seconds_count{status="404"}[5m]))` : 404 error rate
- `sum by (path) (rate(flask_http_request_duration_seconds_count[5m]))` : Request rate by endpoint
- `histogram_quantile(0.50, rate(flask_http_request_duration_seconds_bucket[5m]))` : Median response time
- `histogram_quantile(0.90, rate(flask_http_request_duration_seconds_bucket[5m]))` : 90th percentile response time
- `histogram_quantile(0.95, rate(flask_http_request_duration_seconds_bucket[5m]))` : 95th percentile response time
- `histogram_quantile(0.99, rate(flask_http_request_duration_seconds_bucket[5m]))` : 99th percentile response time
- `sum(rate(flask_http_request_duration_seconds_sum[5m])) / sum(rate(flask_http_request_duration_seconds_count[5m]))` : Average response time

System Metrics (from Node Exporter):

- `node_cpu_seconds_total` : CPU usage
- `rate(node_cpu_seconds_total{mode="idle"}[1m])` : CPU idle rate
- `100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[1m])) * 100)` : CPU usage percentage
- `node_memory_MemAvailable_bytes` : Available memory
- `node_memory_MemTotal_bytes - node_memory_MemFree_bytes - node_memory_Buffers_bytes - node_memory_Cached_bytes` : Used memory
- `node_filesystem_avail_bytes` : Available disk space
- `node_filesystem_size_bytes - node_filesystem_avail_bytes` : Used disk space
- `rate(node_network_receive_bytes_total[1m])` : Network received bytes per second
- `rate(node_network_transmit_bytes_total[1m])` : Network transmitted bytes per second

Grafana

To configure Grafana to visualize the Prometheus data:

1. Open http://localhost:5000 in your browser and log in
2. Go to Configuration > Data sources
3. Add a Prometheus data source:
    - Name: Prometheus
    - URL: http://prometheus:9090
    - Click "Save & Test"
4. Import dashboards:
    - Create a new dashboard or import existing ones from https://grafana.com/grafana/dashboards/
    - For Node Exporter, dashboard ID 1860 is popular
    - For custom Flask metrics, create your own panels using metrics like:
        - `flask_http_requests_total`
        - `flask_http_request_duration_seconds_bucket`

Grafana Dashboard Examples

Here are some dashboards you can create in Grafana:

1. Flask API Overview Dashboard

Create a new dashboard with the following panels:

- **Request Rate**: Graph panel with query `rate(flask_http_request_duration_seconds_count[1m])`
- **Request Rate by Endpoint**: Graph panel with query `sum by (path) (rate(flask_http_request_duration_seconds_count[1m]))`
- **HTTP Status Codes**: Pie chart with query `sum by (status) (flask_http_request_duration_seconds_count)`
- **Response Time (95th percentile)**: Graph panel with query `histogram_quantile(0.95, sum(rate(flask_http_request_duration_seconds_bucket[5m])) by (le))`
- **Error Rate**: Graph panel with query `sum(rate(flask_http_request_duration_seconds_count{status=~"5.."}[1m]))`
- **Request Duration by Path**: Heatmap with query `sum(rate(flask_http_request_duration_seconds_bucket[1m])) by (le, path)`
- **Top 5 Slowest Endpoints**: Table with query:

```
topk(5, sum by (path) (rate(flask_http_request_duration_seconds_sum[5m])) /
sum by (path) (rate(flask_http_request_duration_seconds_count[5m])))
```

2. Node Resources Dashboard

Create a dashboard to monitor the host system:

- **CPU Usage**: Graph panel with query `100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[1m])) * 100)`
- **Memory Usage**: Graph panel with query:

```
100 * (1 - (
  node_memory_MemAvailable_bytes /
  node_memory_MemTotal_bytes
))
```

- **Disk Usage**: Gauge panel with query:

```
100 * (1 - (
  node_filesystem_avail_bytes{mountpoint="/",fstype!="rootfs"} /
  node_filesystem_size_bytes{mountpoint="/",fstype!="rootfs"}
))
```

- **Network Traffic**: Graph panel with queries:
  - Received: `rate(node_network_receive_bytes_total[1m])`
  - Transmitted: `rate(node_network_transmit_bytes_total[1m])`

3. Flask Application Health Dashboard

- **Request Success Rate**: Gauge panel with query:

```
sum(rate(flask_http_request_duration_seconds_count{status=~"2.."}[1m])) /
sum(rate(flask_http_request_duration_seconds_count[1m])) * 100
```

- **Average Response Time**: Graph panel with query `sum(rate(flask_http_request_duration_seconds_sum[5m])) / sum(rate(flask_http_request_duration_seconds_count[5m]))`
- **Request Distribution by Endpoint**: Bar gauge with query `sum by (path) (flask_http_request_duration_seconds_count)`
- **Response Time by Endpoint**: Table panel with query:

```
sum by (path) (rate(flask_http_request_duration_seconds_sum[5m])) /
sum by (path) (rate(flask_http_request_duration_seconds_count[5m]))
```

- **Endpoint Error Rate**: Table with query:

```
sum by (path) (rate(flask_http_request_duration_seconds_count{status=~"[45].."}[5m])) /
sum by (path) (rate(flask_http_request_duration_seconds_count[5m])) * 100
```

- **Request Duration Percentiles**: Graph with queries:
  - 50th: `histogram_quantile(0.5, sum(rate(flask_http_request_duration_seconds_bucket[5m])) by (le))`
  - 90th: `histogram_quantile(0.9, sum(rate(flask_http_request_duration_seconds_bucket[5m])) by (le))`
  - 95th: `histogram_quantile(0.95, sum(rate(flask_http_request_duration_seconds_bucket[5m])) by (le))`
  - 99th: `histogram_quantile(0.99, sum(rate(flask_http_request_duration_seconds_bucket[5m])) by (le))`

4. Advanced Flask Metrics Dashboard

Create a dashboard to analyze API performance in detail:

- **Request Throughput**: Graph panel showing requests per second:

```
sum(rate(flask_http_request_duration_seconds_count[5m]))
```

- **Apdex Score**: Gauge showing application performance index:

```
(
  sum(rate(flask_http_request_duration_seconds_bucket{le="0.1"}[5m])) +
  sum(rate(flask_http_request_duration_seconds_bucket{le="0.5"}[5m])) / 2
) / sum(rate(flask_http_request_duration_seconds_count[5m]))
```

- **Request Duration Breakdown**: Graph showing request duration grouped by status code:

```
sum by (status) (rate(flask_http_request_duration_seconds_sum[5m])) /
sum by (status) (rate(flask_http_request_duration_seconds_count[5m]))
```

- **Slow Endpoints Heatmap**: Heatmap showing distribution of request durations:

```
sum(rate(flask_http_request_duration_seconds_bucket[1m])) by (le, path)
```

Importing Pre-built Dashboards

For easy setup, you can import these pre-built dashboards:

1. In Grafana, go to Dashboards > New > Import
2. Enter one of these dashboard IDs:
   - 1860: Node Exporter Full (system metrics)
   - 9688: Flask Monitoring Dashboard
   - 3662: Prometheus 2.0 Overview
   - 11159: Node Exporter for Prometheus
3. Select your Prometheus data source and click Import

Stopping the Application

To stop all services:

```
docker-compose down
```

To stop and remove volumes (this will delete persistent data):

```
docker-compose down -v
```