УДК 004

# Мониторинг с помощью Micrometer, Prometheus и Grafana

*Еровлева Регина Викторовна*
*Приамурский государственный университет имени Шолом-Алейхема*
*Студент*

*Еровлев Павел Андреевич*
*Приамурский государственный университет имени Шолом-Алейхема*
*Студент*

**Аннотация**
В данной статье будут рассмотрены возможности создания мониторинга для отслеживания работоспособности приложения. Мониторинг будет осуществлен с помощью средств Micrometer, Prometheus, Grafana.
**Ключевые слова:** Docker, Grafana, Micrometer, Prometheus, Spring boot

# Monitoring with Micrometer, Prometheus and Grafana

*Erovleva Regina Viktorovna*
*Sholom-Aleichem Priamursky State University*
*student*

*Erovlev Pavel Andreevich*
*Sholom-Aleichem Priamursky State University*
*Student*

**Abstract**
This article will explore the possibilities of creating monitoring to track the health of an application. Monitoring will be carried out using Micrometer, Prometheus, Grafana tools.
**Keywords:** Docker, Grafana, Micrometer, Prometheus, Spring boot

Мониторинг является важным аспектом ремонтопригодности приложений, поэтому неудивительно, что существует множество платформ, которые позволяют должным образом контролировать свои приложения. Обычно платформы мониторинга работают с использованием базы данных временных рядов, которая является базой данных, оптимизированной для временной информации, такой как показатели приложений. С другой стороны, есть часть визуализации, которая позволяет отображать временные диаграммы для визуализации данных.

Некоторые продукты предлагают и то, и другое вместе (Graphite, Prometheus), в то время как другие платформы либо управляют частью базы данных (InfluxDB), а другие платформы управляют визуализацией (Grafana).

В своей статье А.И.Егунова, А.А.Аббакумов, М.А.Воропаева, Ю.С. Вечканова рассматривают проблемы повышения эффективности образовательной деятельности вуза и использование научной интеллектуальной собственности в преподавательской деятельности. Так же предложили реализацию информационного хранилища и поисковой системы в виде J2EE-приложения с использованием фреймворка Spring [1]. П.В. Прохоров, Н.В. Разговоров рассмотрели в своей работе современные подходы и технологии в разработке серверных приложений на примере онлайн-магазина с использованием Spring [2]. В своей статье В.С. Жданова описала подход в реализации серверной части клиент-серверного мобильного приложения для просмотра расписания [3]. Так же А.Д. Нарижный, Н.Е. Губенко провели сравнительный анализ технологий, которые имеют схожую функциональность и которые предназначены для одних и тех же задач. Это технологии стеков Spring и JavaEE, которые предназначены для разработки Enterprise-приложений [4]. В.Л. Волушкова рассмотрела в своей работе технологии программирования на примере языка Java [5].

Micrometer предоставляет чистый API, к которому можно получить доступ, и имеет мост ко многим платформам мониторинга, включая «Prometheus».

Чтобы иметь возможность отслеживать приложение при загрузке Spring, необходимо добавить следующие зависимости (рис.1).

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

Рисунок 1 – Добавление зависимости

Теперь можно настроить «Actuator» для доступа к конечной точке «Prometheus», настроив свойства в «application.properties» (рис.2).

```
management.endpoints.web.exposure.include=prometheus
```

Рисунок 2 – Добавление свойств

Если запустить приложение и перейти по адресу «http://localhost:8080/actator», то будет видно, что оно предоставляет конечную точку «Prometheus» (рис.3).

```
{
  "_links": {
    "self": {
      "href": "http://localhost:8080/actuator",
      "templated": false
    },
    "prometheus": {
      "href": "http://localhost:8080/actuator/prometheus",
      "templated": false
    }
  }
}
```


Рисунок 3 – Результат вывода

Если нажать на эту ссылку, то будет видно, что множество показателей отображается в формате, который может быть легко прочитан «Prometheus».

Настроить «Prometheus» и «Grafana» можно разными способами. Сейчас проведем настройку с использованием «Docker» для настройки контейнеров. «Prometheus» и «Grafana» имеют свои собственные официальные изображения, готовые к использованию (рис.4а, 4б).

```
version: '3.7'

services:
  movie-quote-service:
    image: g00glen00b/movie-quote-service:0.0.1
    ports:
      - 8080:8080
    networks:
      monitoring:
        aliases:
          - movie-quote-service
  grafana:
    image: grafana/grafana:5.4.3
    ports:
      - 3000:3000
    volumes:
      - ./grafana:/var/lib/grafana
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=password
    networks:
      monitoring:
        aliases:
          - grafana
```

Рисунок 4а – Настройка

```
  prometheus:
    image: prom/prometheus:v2.6.1
    ports:
      - 9090:9090
    volumes:
      - ./config/prometheus.yml:/etc/prometheus/prometheus.yml
      - ./prometheus:/prometheus
    networks:
      monitoring:
        aliases:
          - prometheus
networks:
  monitoring:
```

Рисунок 4б – Настройка

Для контейнера Grafana настроено имя пользователя и пароль, настроены переменные среды «GF_SECURITY_ADMIN_USER» и «GF_SECURITY_ADMIN_PASSWORD».

Кроме того, установлен объем для «/var/lib/grafana», поскольку этот каталог будет содержать все данные. Это означает, что можно безопасно удалить контейнер, и все данные останутся.

Для контейнера «Prometheus» сделаем то же самое: создав том для «/prometheus», где можно будет гарантировать, что все данные останутся.

Далее сделаем том для файла конфигурации под названием «prometheus.yml», который будет содержать конечные точки, которые «Prometheus» должен опрашивать для получения данных.

Последнее, что я хочу упомянуть об этом файле конфигурации, это то, что я использую сеть, называемую «мониторинг», так что оба контейнера могут иметь доступ друг к другу.

«Prometheus» нужно настраивать отдельно, создав файл «prometheus.yml». Настроим «Prometheus» так, чтобы он сканировал следующие места (рис.5):

- Приложение для загрузки Spring.
- Сам Prometheus.
- Веб-приложение Grafana.

```
scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 1m
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'movie-quote-service'
    scrape_interval: 1m
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['movie-quote-service:8080']
  - job_name: 'grafana'
    scrape_interval: 1m
    metrics_path: '/metrics'
    static_configs:
      - targets: ['localhost:3000']
```

Рисунок 5 - prometheus.yml

Эта конфигурация будет очищать все конечные точки «/prometheus» каждую минуту и добавлять данные в свою базу данных временных рядов.

Следующим шагом будет войти в «Grafana». Она должна работать на порту 3000, или другом заданным, поэтому, перейдя по адресу «http://localhost:3000/login», произойдет вход в систему как администратор с учетными данными, которые были настроены ранее.

Первым шагом после входа в систему является создание источника данных, которым будет «Prometheus». Все, что нужно сделать, это настроить URL-адрес как «http://prometheus:9090» (рис.6).

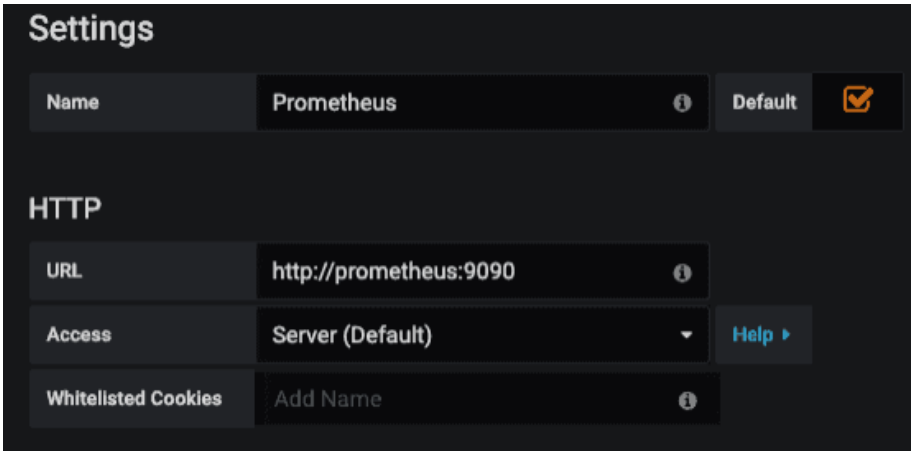


Рисунок 6 – Настройка

После этого можно создать новую панель и добавить к нему график. Если выбрать «Prometheus» в качестве источника данных по умолчанию, то можно начать добавлять метрики, например «jvm_memory_used_bytes», которые будут содержать информацию об использовании памяти приложением.

Если добавить это в запрос, то получаем несколько графиков. Причина этого в том, что загрузка Spring предоставляет это с помощью различных тегов, таких как область, которая может быть «куча», «nonheap», и «id», который может быть «PS Eden Space», «PS Old Gen».

Если необходимо построить график для всего использования памяти кучи, то можно использовать другую метрику (рис.7).

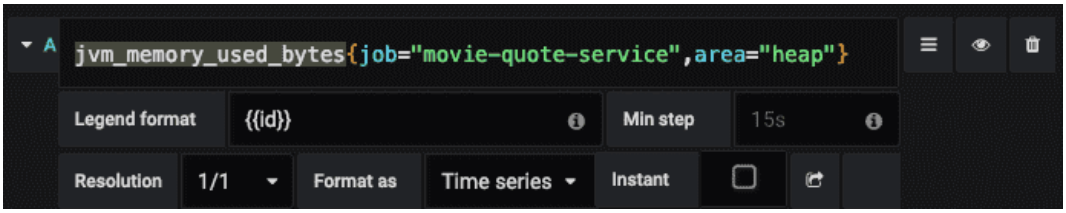


Рисунок 7 – Настройка метрики

Можно использовать динамическую легенду используя «{id}». Далее сохраним команду и запустим (рис.8).

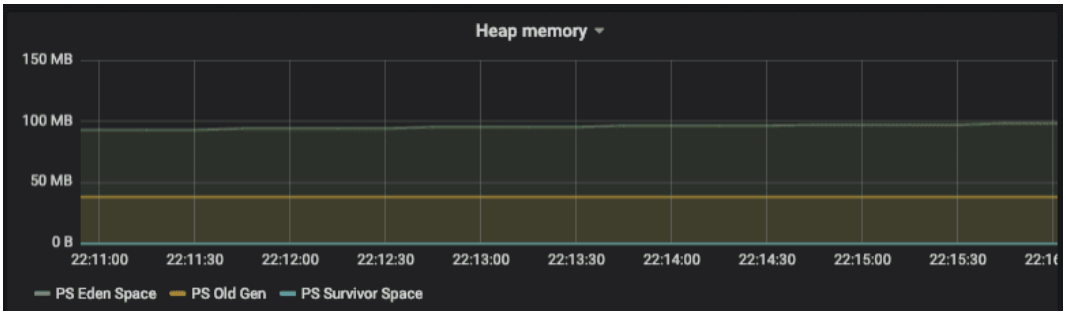


Рисунок 8 – График памяти

В дополнение к графикам также можно настроить отдельные статистические данные, которые могут быть интересны для таких показателей, как время безотказной работы приложения (рис.9).

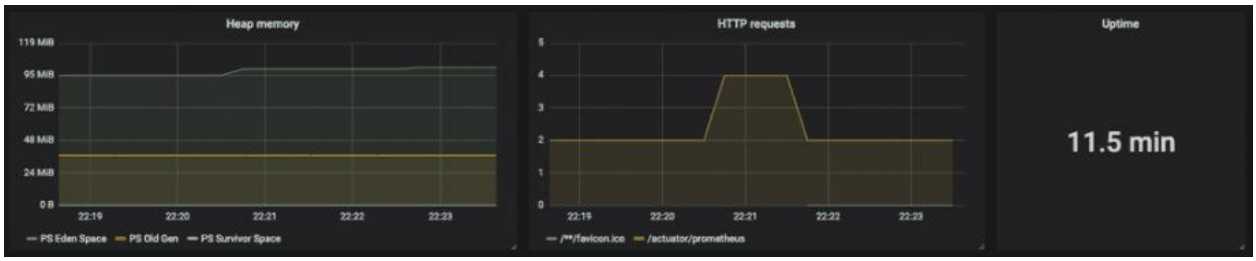


Рисунок 9 – График работы приложения

В данной статье был рассмотрен процесс использования метрических данных и вывод данных в визуализацию на примере графиков.

**Библиографический список**

1. Егунова А.И., Аббакумов А.А., Воропаева М.А., Вечканова Ю.С. Система управления хранилищем электронных образовательных ресурсов //

Образовательные технологии и общество. 2019. №3. С. 145-154.
2. Прохоров П.В., Разговоров Н.В. Современные подходы в backend разработке на примере онлайн-магазина // Прикладная математика и фундаментальная информатика. 2020. №2. С. 23-28.
3. Жданова В.С. Серверный модуль системы уведомления об изменениях в расписании занятий // Молодость. Интеллект. Инициатива. 2017. С. 21-22.
4. Нарижный А.Д., Губенко Н.Е. Сравнительный анализ стеков технологий spring и javaee (jakartaee) для разработки enterprise приложений // Информатика, управляющие системы, математическое и компьютерное моделирование 2020. №3. С. 549-462.
5. Волушкова В.Л. Архитектурные решения java для доступа к данным // Теоретические основы программирования. Учебное пособие 2019. 137 с.