|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №4**

по дисциплине «Технологии виртуализации клиент-серверных приложений»

**Студент группы** ИКБО-20-19 Анваржонов Ж Т

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** старший преподаватель Волков М.Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Москва 2022

**Задание**

Вам необходимо создать Spring Boot сервис и произвести мониторинг его работы. В работе должны быть отражены все пункты создания docker-compose файла, проверена работоспособность сервиса и систем мониторинга. Произведена нагрузка на систему с целью проверки работоспособности. Все пункты должны быть отражены в отчете в формате снимков экрана.

Требования к системе:

1. Наличие сервера с CRUD набором для взаимодействия с базой данных и выгрузкой данных с GrayLog по эндпоинту
   1. CRUD набор должен содержать эндпоинты для добавления, обновления, чтения и удаления записей
   2. Количество используемых моделей данных должно быть не менее 3, а также содержать как минимум 1 связь 1-N, 1 связь N-N.
   3. При обращении к эндпоинту логов он должен выгружать данные в формате csv из GrayLog
2. PostgreSQL в качестве СУБД с которой будет производиться взаимодействие/сниматься метрики и логи
3. Zabbix для мониторинга базовой работоспособности сервера
4. Prometheus для сбора данных с PostgreSQL, Grafana для вывода этих данных
5. GrayLog для сбора данных с PostgreSQL
6. Adminer для управления базой данных PostgreSQL

**Выполнение работы:**

## Настройка Graylog.

1. Настроем окружение для GrayLog сервера

GrayLog сервер требует компоненты ElasticSearch, ModgoDB, поэтому при разворачивании в docker, необходимо добавить эти зависимости тоже

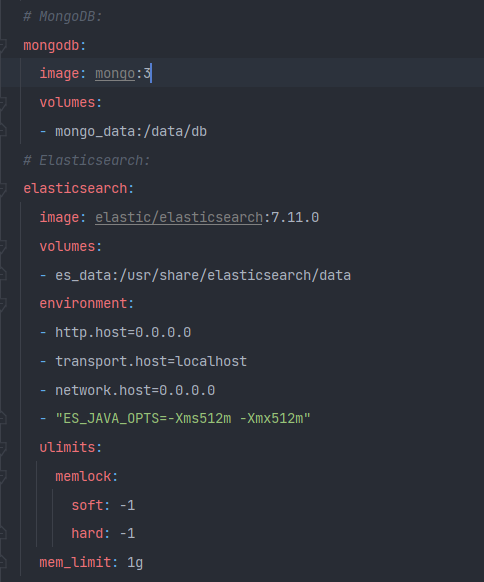


Рисунок – сервисы mongodb и elasticsearch в docker-compose

При разворачивании необходимо открыть UDP-порты, чтобы логирование проходило через gelf udp

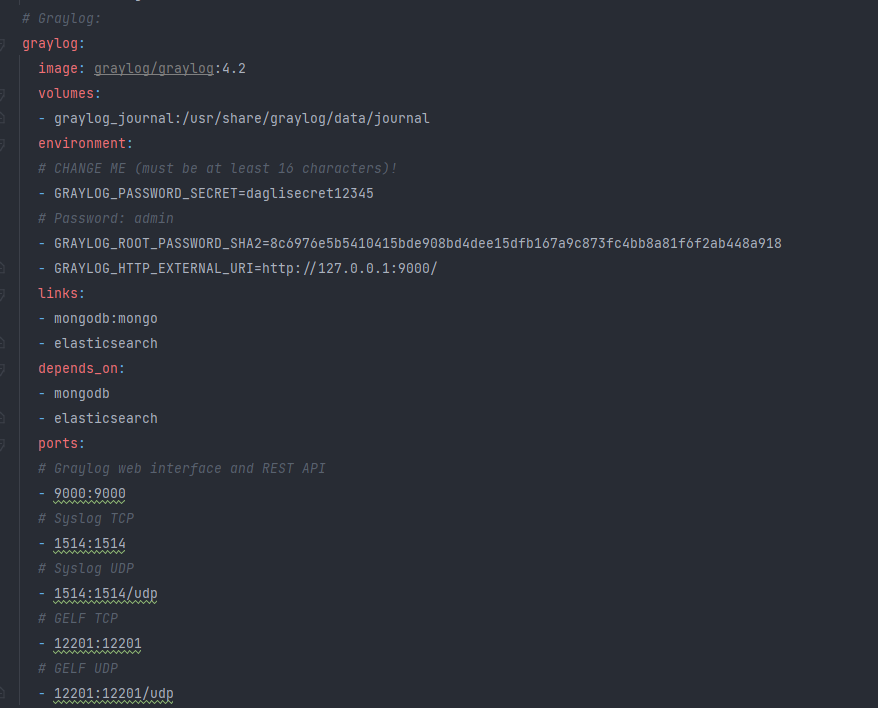


Рисунок – настройка graylog в docker-compose

После поднятия GrayLog сервера необходимо настроить input для GELF UDP соединений

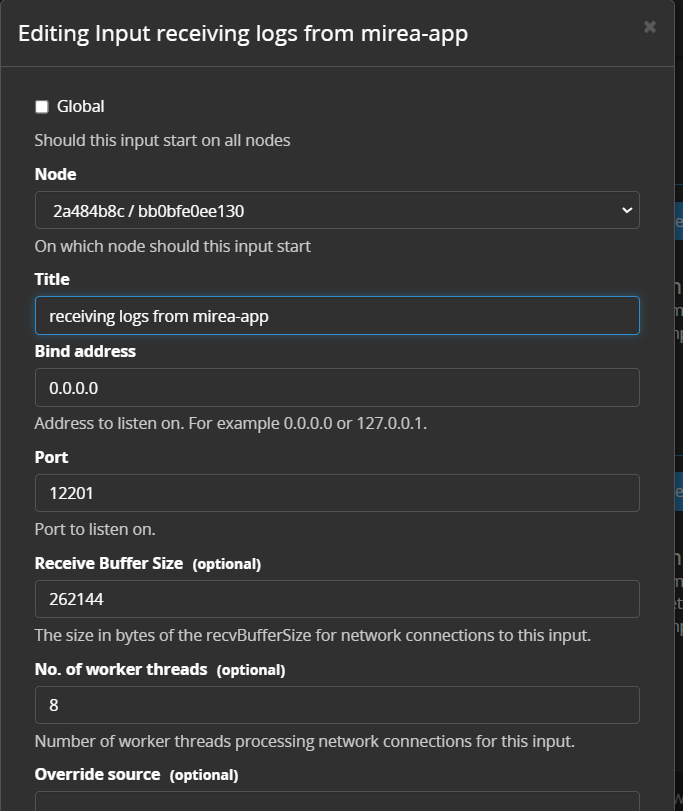


Рисунок – настройка input на сервере graylog’a

1. Настроем логирование через UDP на сервер GrayLog

Чтобы логи из нашего spring-boot приложения отправлялись на graylog сервер, необходимо добавить appender для gelf udp в logback конфигурации

<configuration *debug* = "true">  
  
 <property *name*="GRAYLOG\_PORT" *value*="12201" />  
 <property *name*="GRAYLOG\_HOST" *value*="localhost" />  
  
 <appender *name*="STDOUT" *class*="ch.qos.logback.core.ConsoleAppender">  
 <encoder>  
 <pattern>%green(%date) %highlight(%-5level) %yellow([%file:%line]) %blue(: %msg%n)</pattern>  
 <charset>UTF-8</charset>  
 </encoder>  
 </appender>  
  
 <appender *name*="GELF" *class*="de.siegmar.logbackgelf.GelfUdpAppender">  
 <graylogHost>${GRAYLOG\_HOST}</graylogHost>  
 <graylogPort>${GRAYLOG\_PORT}</graylogPort>  
 <encoder *class*="de.siegmar.logbackgelf.GelfEncoder">  
 <originHost>${GRAYLOG\_HOST}</originHost>  
 <includeRawMessage>false</includeRawMessage>  
 <includeMarker>true</includeMarker>  
 <includeMdcData>true</includeMdcData>  
 <includeCallerData>true</includeCallerData>  
 <includeRootCauseData>true</includeRootCauseData>  
 <includeLevelName>true</includeLevelName>  
 <shortPatternLayout *class*="ch.qos.logback.classic.PatternLayout">  
 <pattern>%m%nopex</pattern>  
 </shortPatternLayout>  
 <fullPatternLayout *class*="ch.qos.logback.classic.PatternLayout">  
 <pattern>%m%n</pattern>  
 </fullPatternLayout>  
 <staticField>app\_name:graylog-spring</staticField>  
 </encoder>  
 </appender>  
  
  
 <root *level*="INFO">  
 <appender-ref *ref*="STDOUT" />  
 <appender-ref *ref*="GELF" />  
 </root>  
</configuration>

Рисунок – logback конфигурация

Запустим наше приложение и проверим корректность отправки логов на сервер

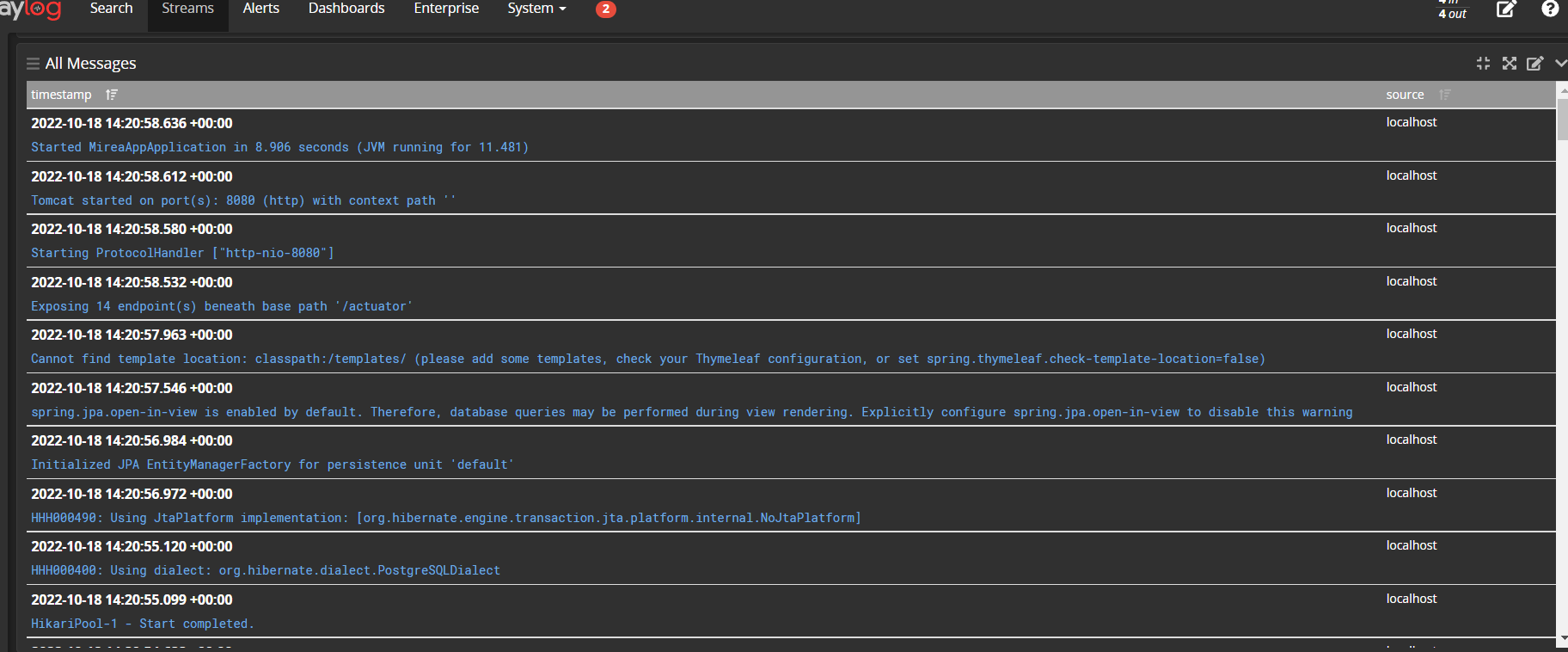


Рисунок – отображение логов на GrayLog-сервере

## Настройка Prometheus & Grafana

1. Настроем Spring Actuator для извлечения метрик

Подключим зависимости:



Рисунок – зависимости для метрик Prometheus, jmx



Рисунок – application.yml настроек actuator

1. Проверим работоспособность метрик Prometheus. Для этого прокинем запрос на эндпоинт /actuator/Prometheus

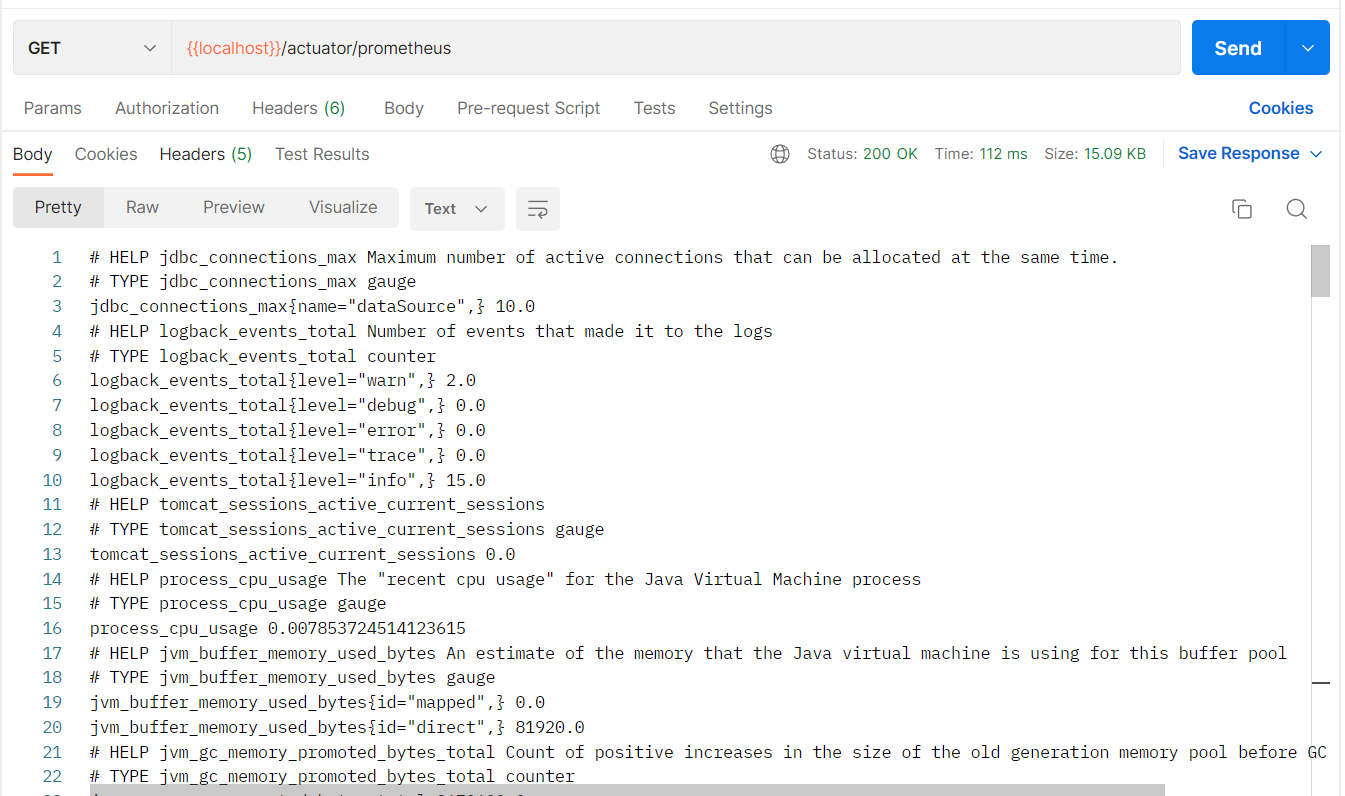


Рисунок – ответ от actuator

В ответе от actuator получаем настроенные метрики, доступные в Prometheus

1. Настроим Prometheus контейнер и требуемый к нему конфигурацию

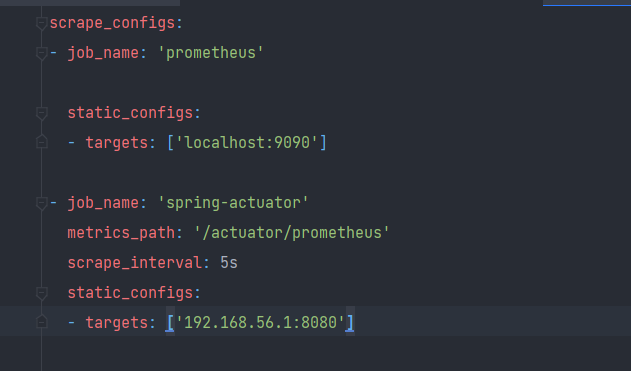


Рисунок – конфигурация для Prometheus

Добавим в наш docker-compose сервис для Prometheus, который будет монтировать нашу конфигурацию внутрь контейнера

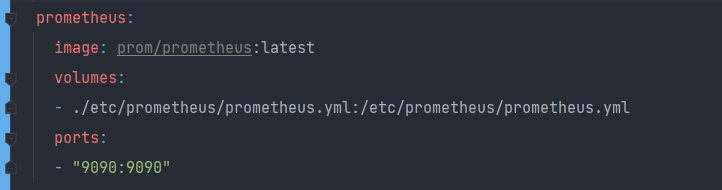


Рисунок - Docker-compose Prometheus

Перейдем по localhost:9090 и попробуем извлечь метрики нашего spring boot приложения:

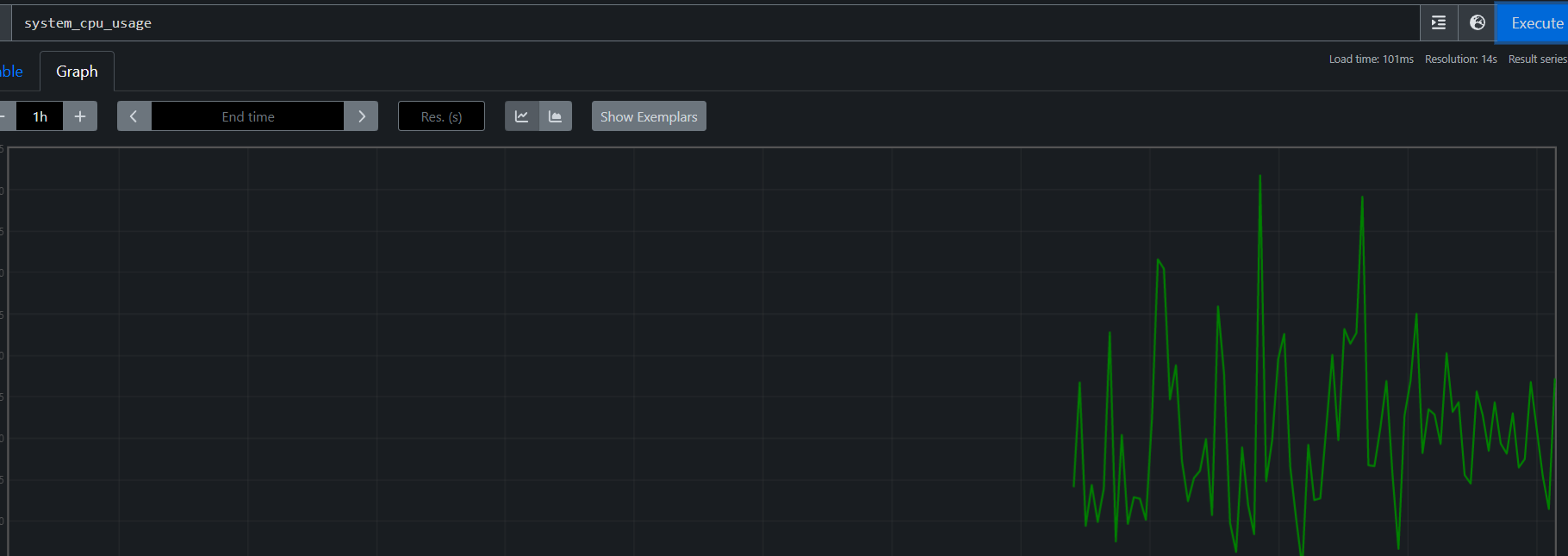


Рисунок – запрос метрик на количество потраченной памяти нашим приложением

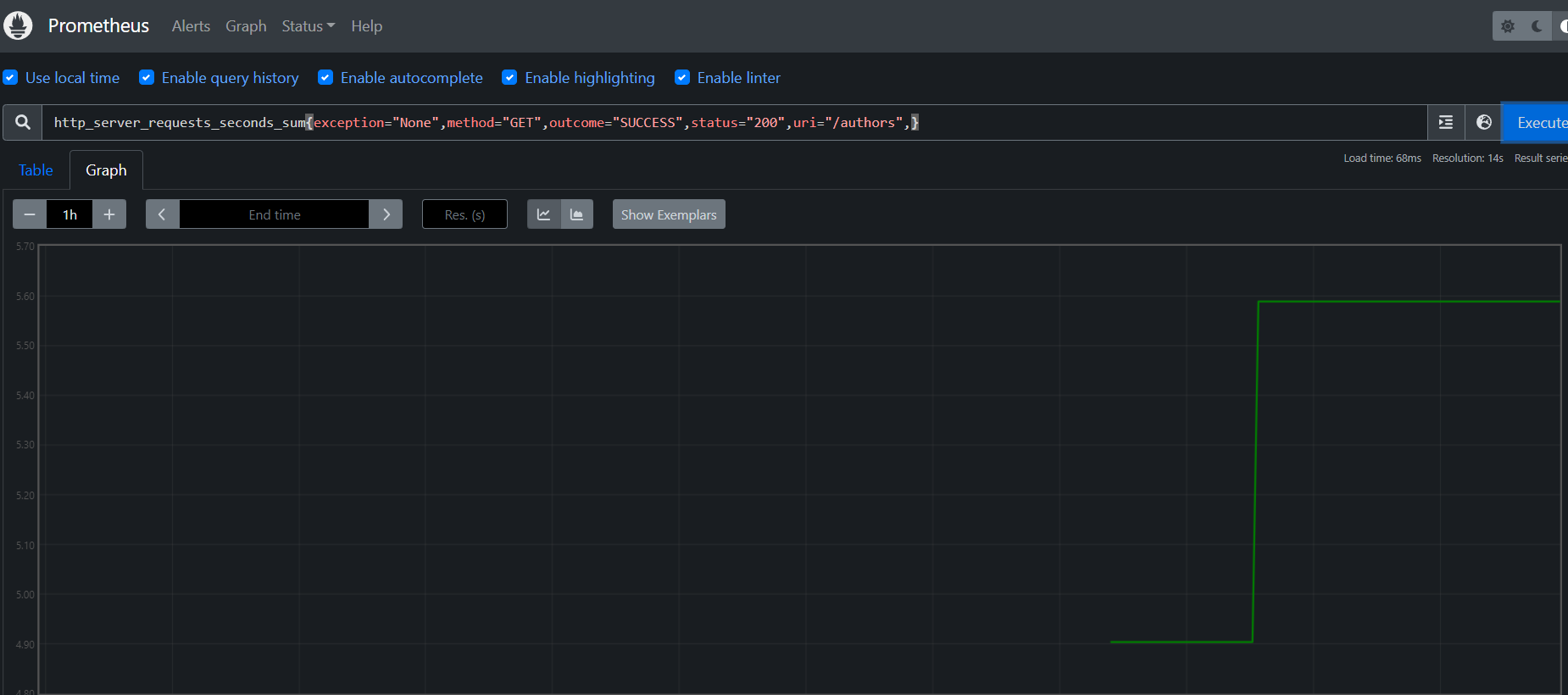


Рисунок – запрос метрики на удачные http запросы по эндпоинту /authors

1. Настроим Grafana для визуализации наших метрик

Настроим конфигурацию grafana, где укажем адрес нашего prometheus

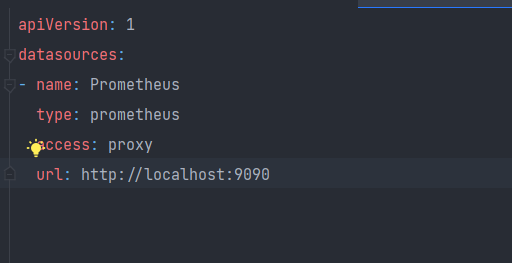


Рисунок – файл datasources.yml

Добавим контейнер grafana в наш docker-compose:

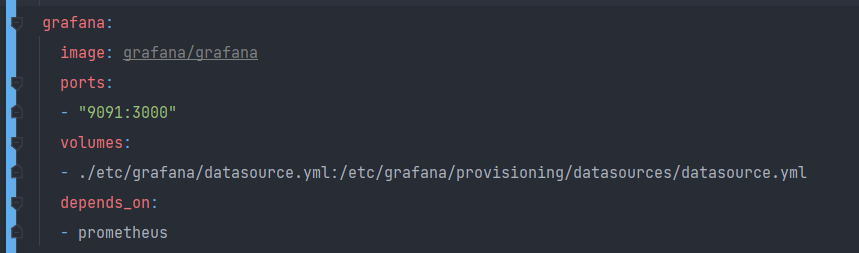


Рисунок – контейнер grafana

Включим извлечение метрик из Prometheus в grafana:

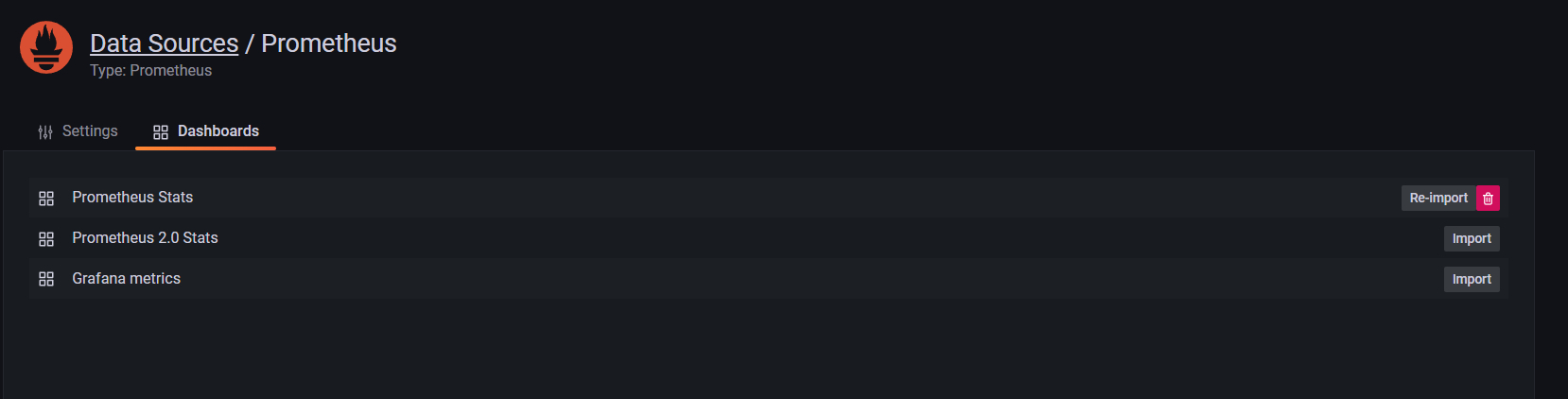


Рисунок – настройка Prometheus

Покажем визуализацию статистики из Prometheus

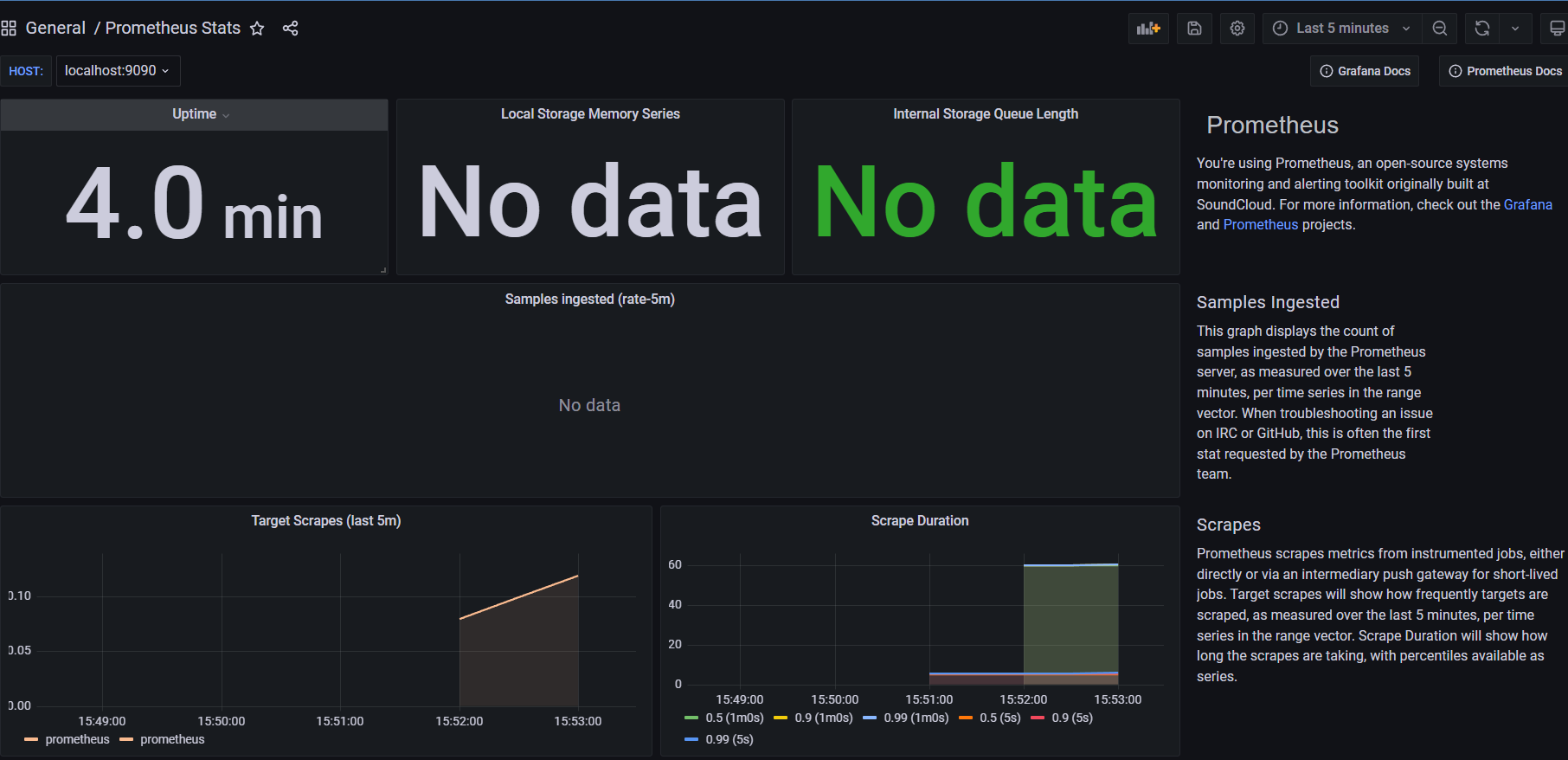


Рисунок – дашборд Prometheus stats

Откроем панель Adminer и проверим его работу

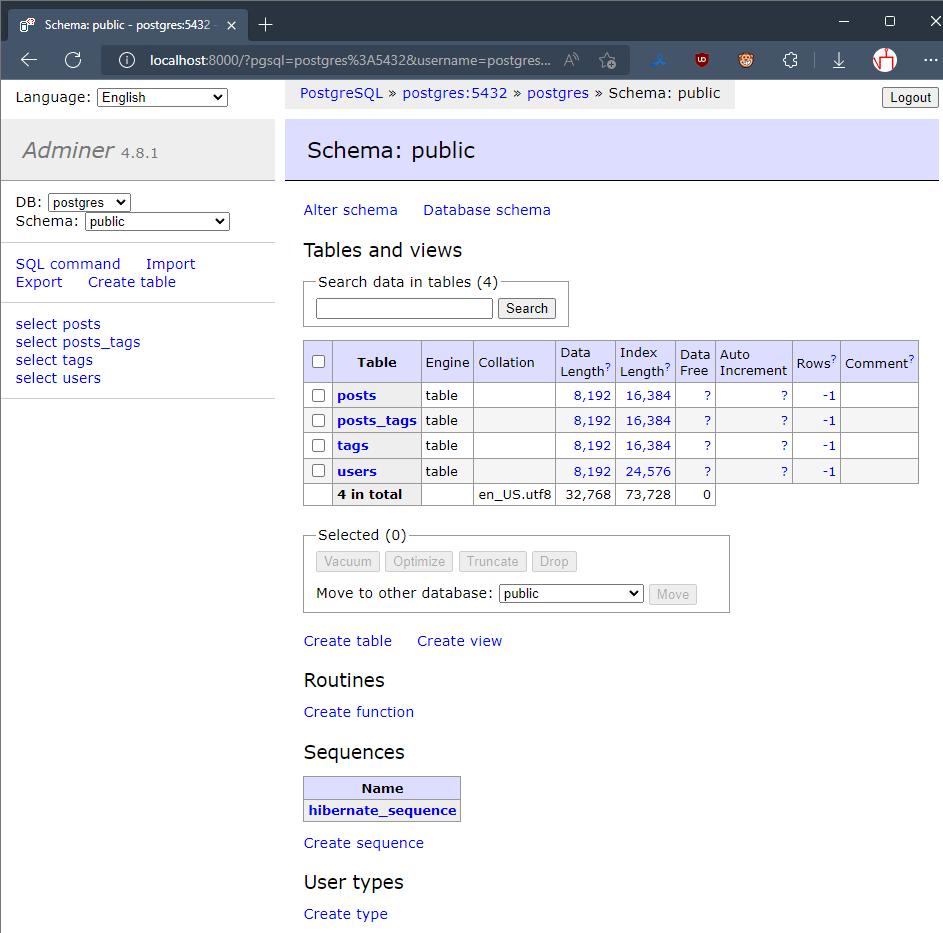


Рисунок – Демонстрация работы панели Adminer

**Вывод**

В результате выполнения данной практической работы были получены навыки работы с docker-compose, Adminer, GrayLog, Prometheus, Grafana, Zabbix, PostgreSQL, Spring Boot.

**Ответы на контрольные вопросы**

1. Назовите основные различия между Prometheus и Zabbix.

Zabbix и Prometheus могут использоваться в различных сценариях мониторинга, без какой-либо специализации в любом из них. Zabbix старше Prometheus и, вероятно, более стабилен, с более готовыми к использованию решениями.

Zabbix имеет ядро, написанное на C и webUI на основе PHP, также использует "агенты" (клиентские программы), написанные в с. Прометей написан на языке го. Zabbix хранит данные в СУБД (MySQL, PostgreSQL, Oracle, sqlite) пользователя выбор. Prometheus использует собственную базу данных, встроенную в бэкэнд-процесс (это нереляционная база данных, специально разработанная для хранения данных мониторинга таким же образом до OpenTSDBмодель данных).

Zabbix по умолчанию использует модель "pull", когда сервер подключается к агентам на каждой машине мониторинга, агенты периодически собирают информацию и отправляют ее на сервер. Альтернативой является режим" активных проверок", когда агенты устанавливают соединение с сервером и отправляют данные к нему, когда это нужно. Прометей предпочитает "вытягивать" модель, когда сервер собирает информацию с клиентских машин. Но!--13-->Prometheus Push Gateway может использоваться в случаях, когда необходима модель" push".

Prometheus требует, чтобы приложение было инструментировано с клиентской библиотекой Prometheus (доступной на разных языках программирования) для подготовки метрик. Но для мониторинга системы или программного обеспечения, которые не могут быть инструментированы, есть официальная "черный ящик экспортер" это позволяет зондировать конечные точки по целому ряду протоколов; кроме того, широкое распространение сторонние "экспортеры" и инструменты доступны, чтобы помочь выставить метрики для Prometheus (подобно "агенты" для Zabbix). Одним из таких инструментов является telegraf (https://github.com/influxdata/telegraf). Zabbix использует собственный tcp-протокол связи между агентами и сервером. Prometheus использует HTTP с буферами протокола (+текстовый формат для удобства использовать с завитком).

Zabbix предлагает собственный webUI для визуализации. Prometheus предлагает базовый инструмент для изучения собранных данных и визуализации их в простых графиках на своем собственном сервере, а также предлагает минимальный Dashboard builder PromDash. Но Prometheus разработан и поддерживается современными инструментами визуализации, такими как Grafana.

Zabbix поддерживает оповещение в своем ядре. Prometheus предлагает решение для оповещения, которое отделено от его ядро в Alertmanager приложение.

2. Как можно запустить две базы PostgreSQL в одном docker-compose файле чтобы они работали одновременно и таблицы внутри не пересекались?

Запустить два контейнера

3. Назовите виды мониторинга систем.

Nagios, Zabbix, Cacti, OpenNMS, Icinga

4. При помощи чего можно передавать конфигурационные переменные в контейнер?

Задав их в Dockerfile, docker-compose или в аргументах запуска образа.

5. Назовите основные различия Docker Swarm и Docker Compose.

Docker Compose используется для конфигурации нескольких контейнеров на одном и том же хосте, когда Docker Swarm инструмент оркестровки контейнеров.

6. Назовите пример задачи, которую невозможно выполнить, используя один лишь docker-compose без Dockerfile

Использование собственного образа

**Список использованных источников**

1. 50 вопросов по Docker, которые задают на собеседованиях, и ответы на них | Хабр. — Текст: электронный [сайт]. — URL: https://habr.com/ru/company/southbridge/blog/528206/

2. Docker Documentation | Docker Documentation — Текст: электронный [сайт]. — URL: — Текст: электронный [сайт]. — URL: https://docs.docker.com/

3. Zabbix Documentation — Текст: электронный [сайт]. — URL: https://www.zabbix.com/manuals

4. Prometheus Documentation — Текст: электронный [сайт]. — URL: https://prometheus.io/docs/introduction/overview/

5. Grafana Documentation — Текст: электронный [сайт]. — URL: https://grafana.com/docs/

6. GrayLog Documentation — Текст: электронный [сайт]. — URL: https://docs.graylog.org/