

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики и кибернетики
Кафедра технической кибернетики

Отчет по лабораторной работе №5

Дисциплина: «Развертывание и жизненный цикл программного обеспечения»

Тема: «**Monitoring - Prometheus Grafana**»

Выполнил: Дубман Л.Б.

Группа: 6133-010402D

Самара 2023

Задание

Шаги:

1. Deploy sample app
2. Install Prometheus+Grafana with helmchart (optional) use values.yaml from repo for Exporter Grafana login password: admin prom-operator
3. Setup sample app monitoring

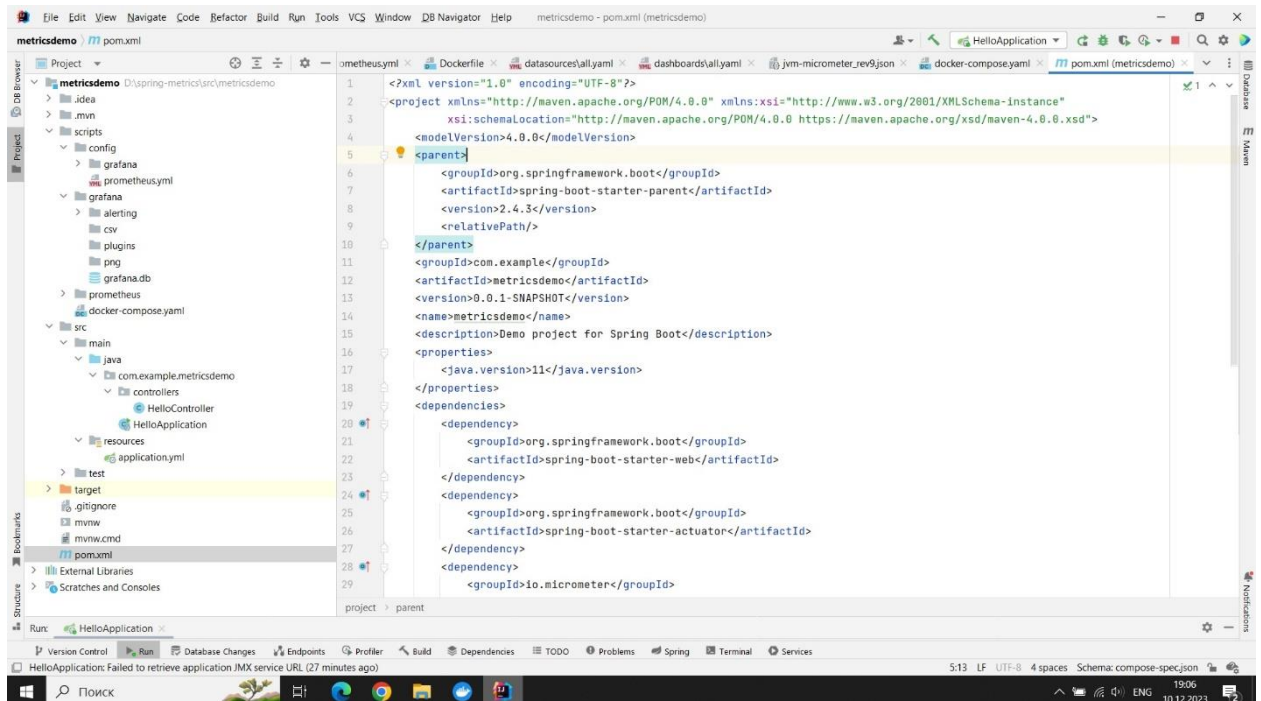
Ход работы

Ограничения данного решения:

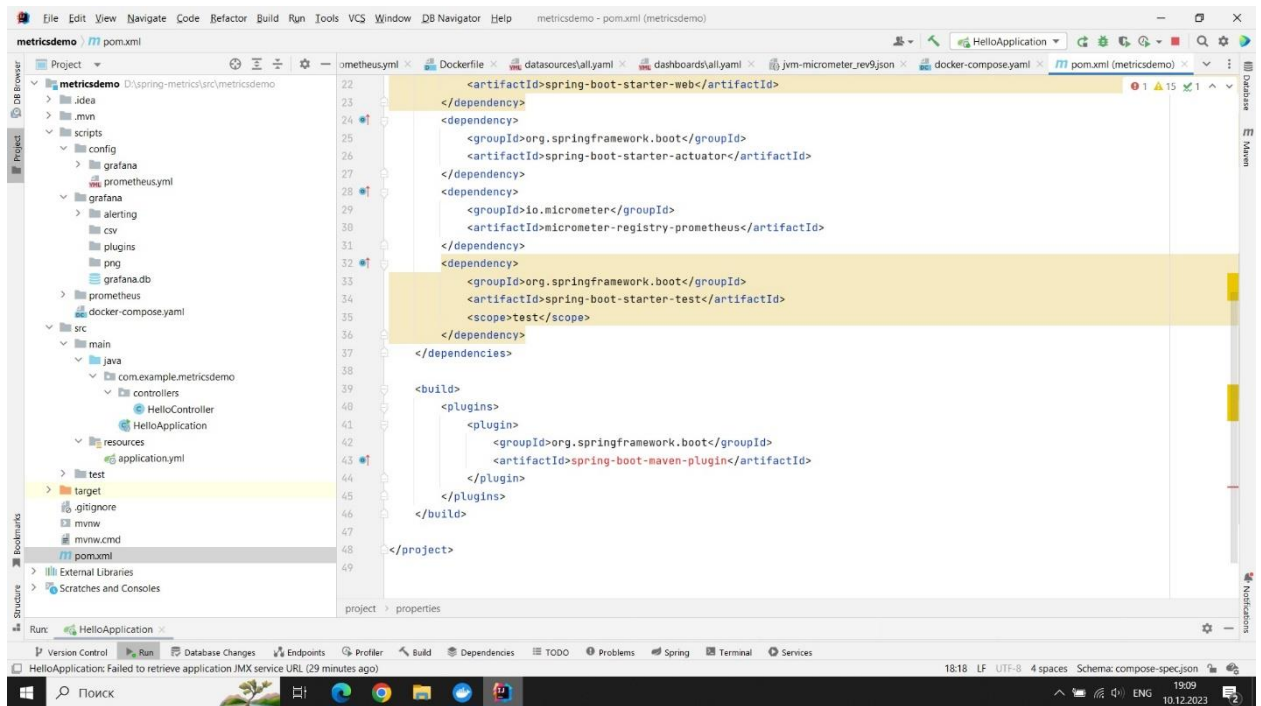
- Maven – в качестве сборщика
- Для запуска Prometheus и Grafana использую docker, docker-compose
- Spring boot

1) Создание приложения Spring Boot

Создание приложения с зависимостью spring-boot-starter-web, ниже создание actuator для предоставления метаданных в приложение.

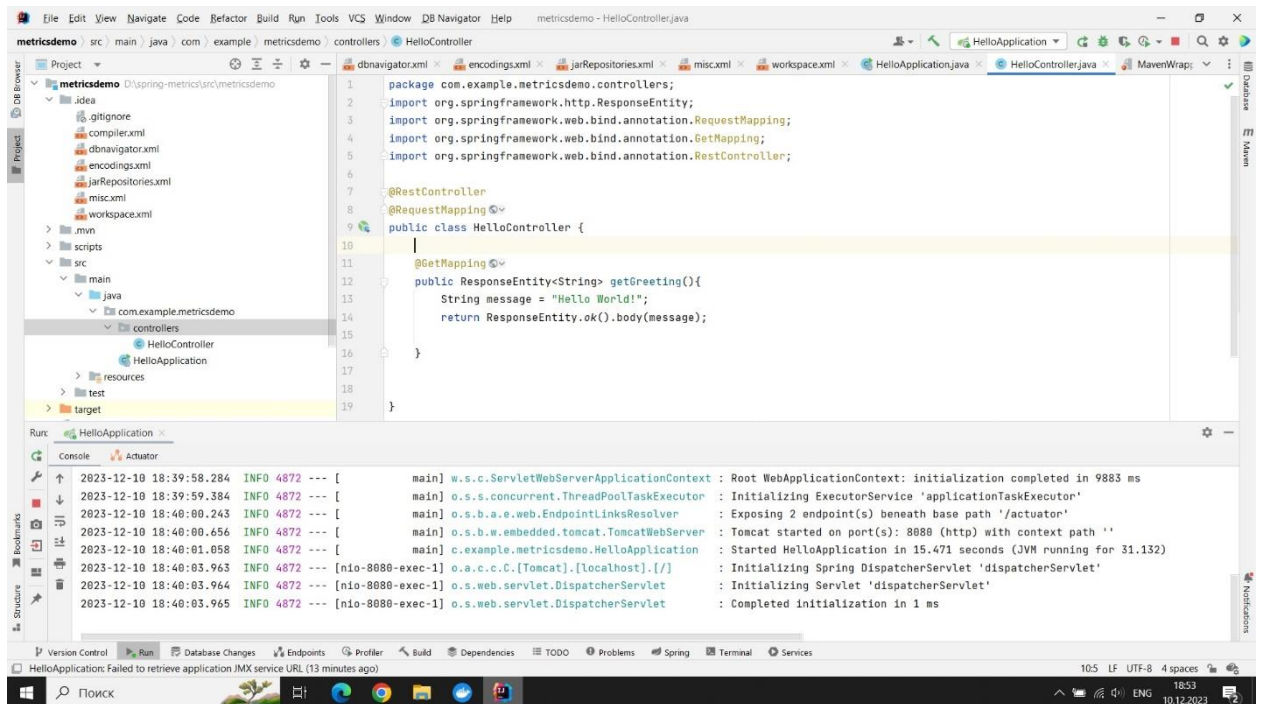


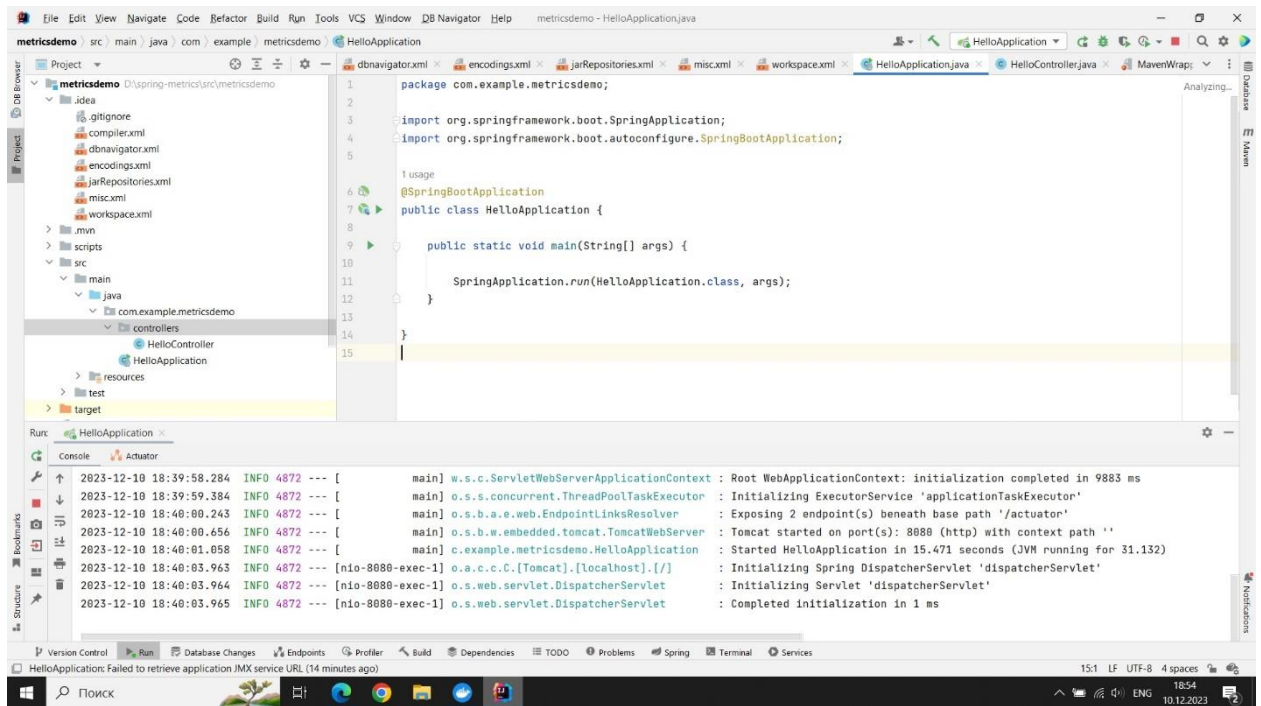
Создание registry для Prometheus, чтобы микрометр мог выгружать для него данные в этом формате.



2) Создание контроллера (привет мир)

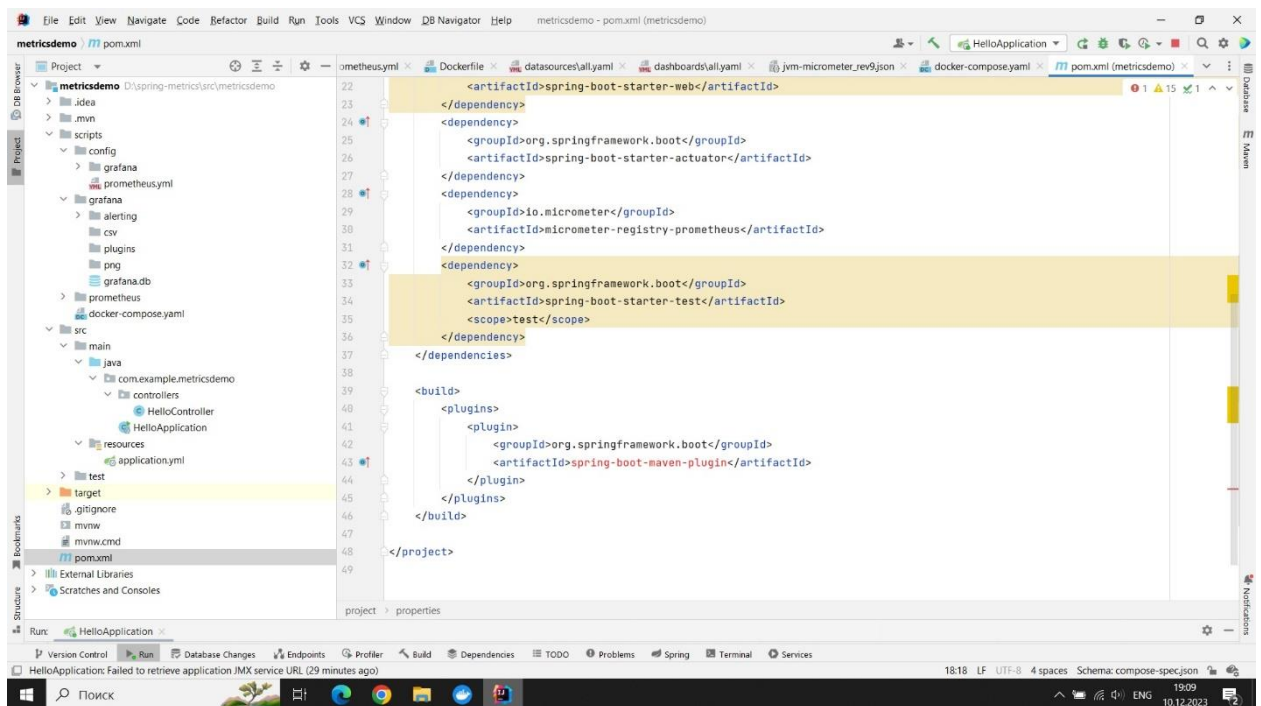
ResponseEntity предоставляет возможность задать body в ответе. Чтобы добавить строку в message body, вам нужно создать объект String.





3) Конфигурация приложения

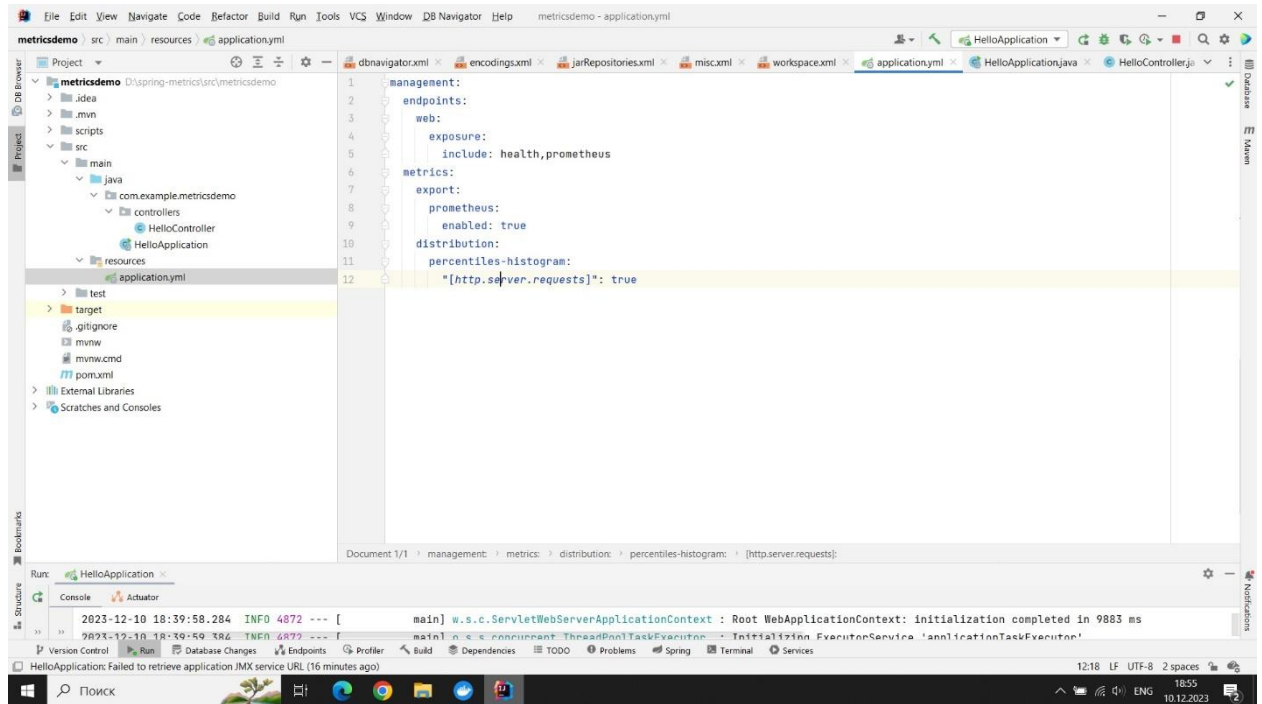
Так как микрометр включен в пакет Spring Boot, то необходимо лишь добавить registry для Prometheus



4) Настройка endpoints, которые мы хотим, чтобы отдавал actuator

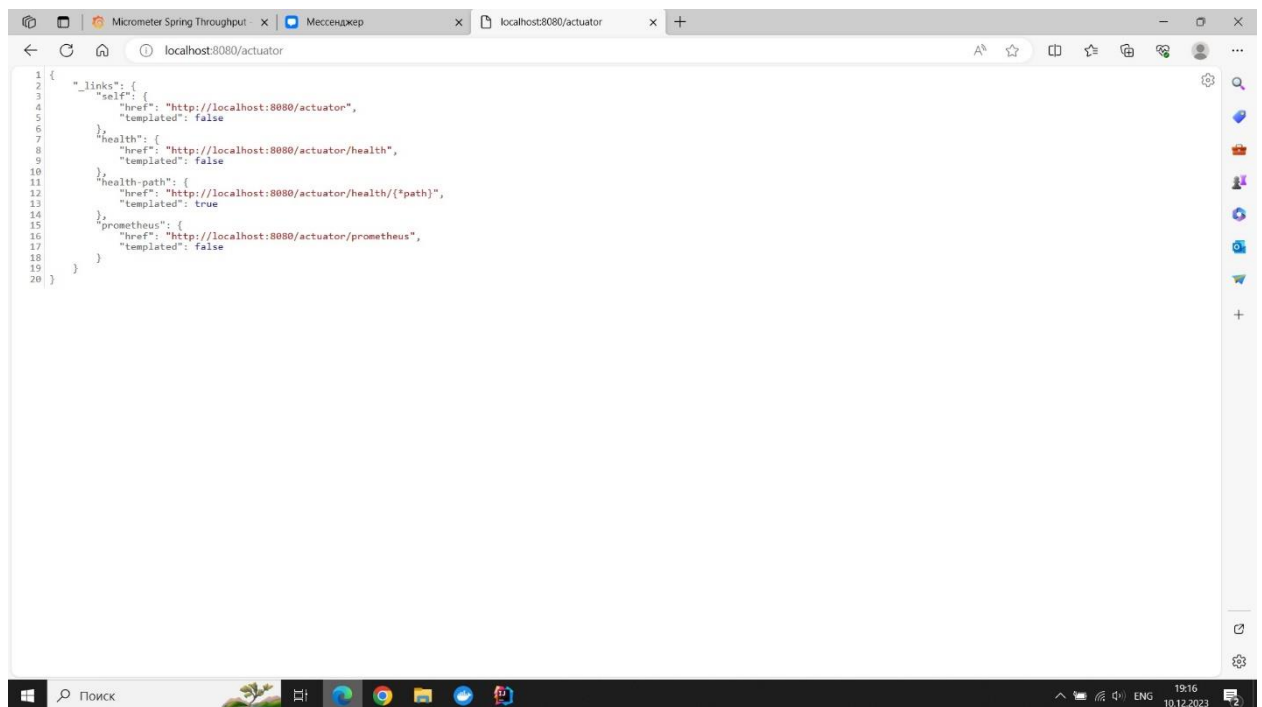
Сначала - health, Prometheus, так как под будут отдаваться метрики.

Далее настраиваю все метрики для jvm, включаем export метрик через Prometheus и, наконец, percentiles-histogram для времени выполнения запросов (с его помощью можно построить диаграммы выполнения).



Запускаем приложение и смотрим actuator (`localhost:8080/actuator`).

Последний endpoint возвращает все метрики.



5) Метрики Prometheus

По умолчанию включены jvm метрики и percentiles-histogram (включили).

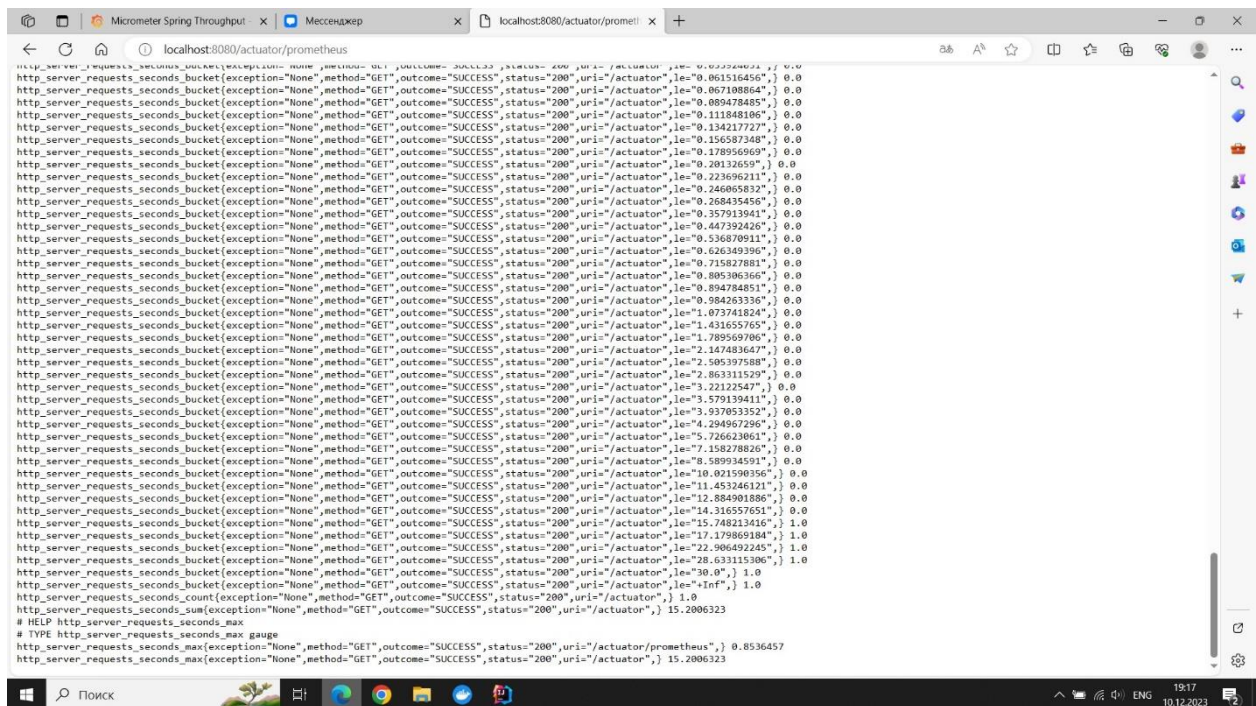
Обращения к actuator также включаются в список.

```
localhost:8080/actuator/prometheus

# HELP process_uptime_seconds The uptime of the Java virtual machine
# TYPE process_uptime_seconds gauge
process_uptime_seconds 2217.305
# HELP process_start_time_seconds Start time of the process since unix epoch.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.702219172193E9
# HELP jvm_threads_live_threads The current number of live threads including both daemon and non-daemon threads
# TYPE jvm_threads_live_threads gauge
jvm_threads_live_threads 25.0
# HELP jvm_threads_states_threads The current number of threads having NEW state
# TYPE jvm_threads_states_threads gauge
jvm_threads_states_threads{state="runnable",} 11.0
jvm_threads_states_threads{state="blocked",} 0.0
jvm_threads_states_threads{state="waiting",} 11.0
jvm_threads_states_threads{state="timed-waiting",} 3.0
jvm_threads_states_threads{state="new",} 0.0
jvm_threads_states_threads{state="terminated",} 0.0
# HELP system_cpu_usage The "recent cpu usage" for the whole system
# TYPE system_cpu_usage gauge
system_cpu_usage 0.09325162876181592
# HELP jvm_gc_memory_promoted_bytes_total Count of positive increases in the size of the old generation memory pool before GC to after GC
# TYPE jvm_gc_memory_promoted_bytes_total counter
jvm_gc_memory_promoted_bytes_total 9237504.0
# HELP tomcat_sessions_created_sessions_total
# TYPE tomcat_sessions_created_sessions_total counter
tomcat_sessions_created_sessions_total 0.0
# HELP jvm_buffer_memory_used_bytes An estimate of the memory that the Java virtual machine is using for this buffer pool
# TYPE jvm_buffer_memory_used_bytes gauge
jvm_buffer_memory_used_bytes{id="mapped" - 'non-volatile memory',} 0.0
jvm_buffer_memory_used_bytes{id="mapped",} 0.0
jvm_buffer_memory_used_bytes{id="direct",} 90112.0
# HELP logback_events_total Number of error level events that made it to the logs
# TYPE logback_events_total counter
logback_events_total{level="warn",} 0.0
logback_events_total{level="debug",} 0.0
logback_events_total{level="error",} 0.0
logback_events_total{level="trace",} 0.0
logback_events_total{level="info",} 7.0
# HELP tomcat_sessions_alive_max_seconds
# TYPE tomcat_sessions_alive_max_seconds gauge
tomcat_sessions_alive_max_seconds 0.0
# HELP jvm_threads_daemon_threads The current number of live daemon threads
# TYPE jvm_threads_daemon_threads gauge
jvm_threads_daemon_threads 21.0
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for the Java virtual machine to use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="heap",id="G1 Survivor Space",} 1048576.0
jvm_memory_committed_bytes{area="heap",id="G1 Old Gen",} 2.4177248E7
```

```
localhost:8080/actuator/prometheus

# HELP system_cpu_count The number of processors available to the Java virtual machine
# TYPE system_cpu_count gauge
system_cpu_count 8.0
# HELP http_server_requests_seconds
# TYPE http_server_requests_seconds histogram
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.001",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.00148576",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.001398101",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.001747626",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.002097151",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.002446676",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.002796201",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.003145726",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.003495251",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.003844776",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.004194304",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.005592405",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.006990906",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.008388607",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.009786708",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.011184809",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.01258291",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.013981011",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.015379112",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.016777216",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.022369621",) 0.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.027962026",) 13.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.033554431",) 230.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.039146836",) 328.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.044739241",) 342.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.050331646",) 351.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.055924051",) 355.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.061516456",) 360.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.067108864",) 367.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.009478495",) 373.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.111848106",) 386.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.134217727",) 392.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.156587348",) 399.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.178956969",) 404.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.20132659",) 407.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.223696211",) 409.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.246065832",) 412.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.268435456",) 412.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.357913941",) 418.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.447392426",) 422.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.536870911",) 426.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.626349396",) 429.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.715027881",) 431.0
http_server_requests_seconds_bucket(exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",le="0.805306366",) 434.0
```



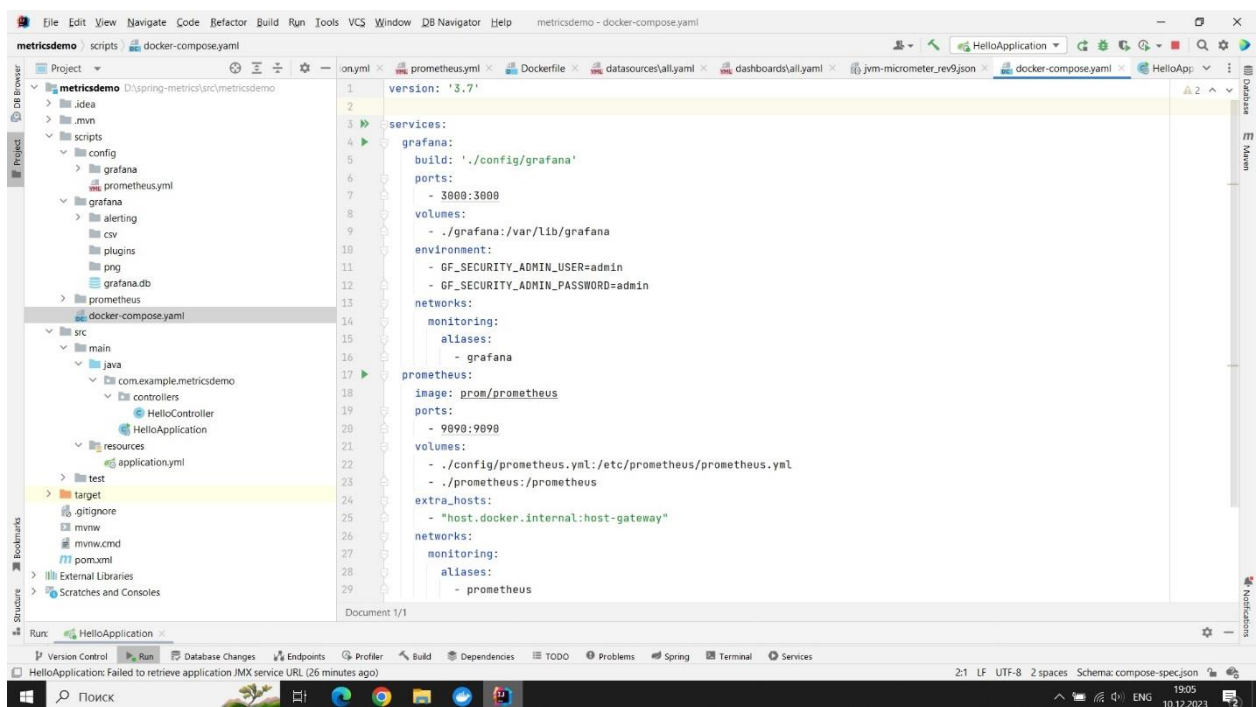
6) Настройка окружения для сбора и хранения метрик

(через докер) и с помощью docker-compose подняты и Prometheus и Grafana.

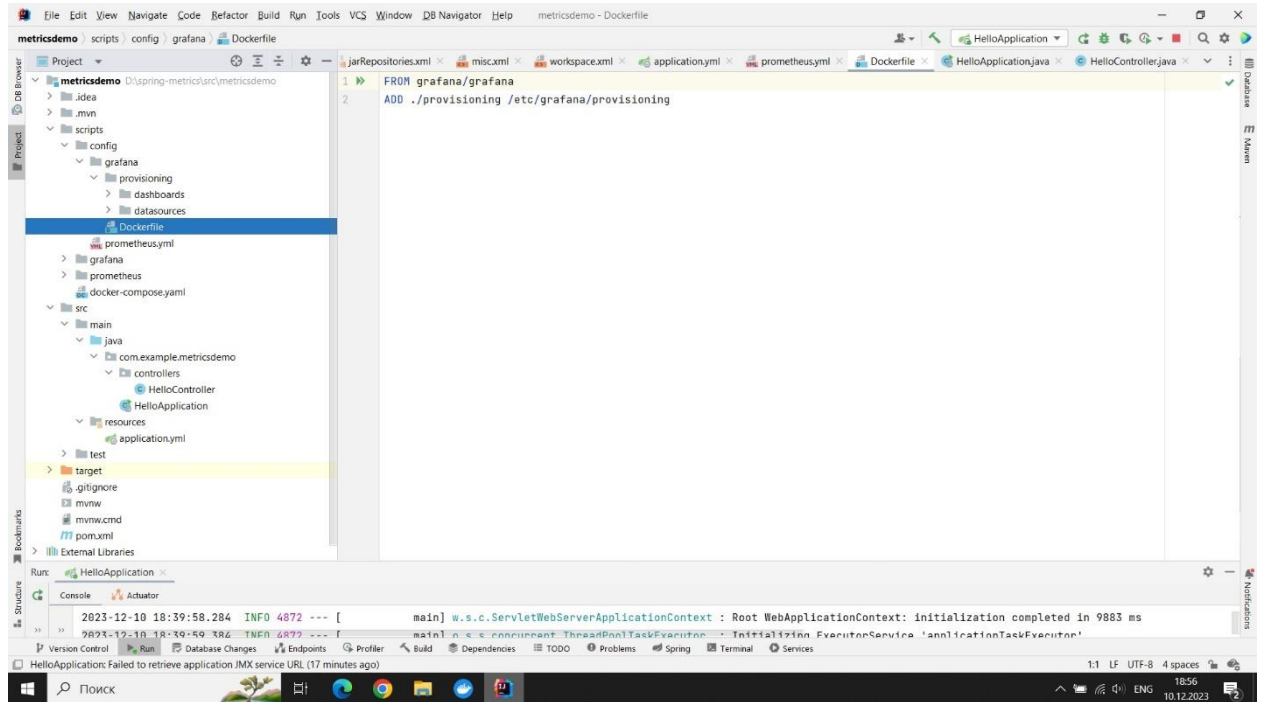
Образ Grafana собирается, чтобы в Grafana добавлять dashboards.

Также заданы переменные окружения для админа и создан volume для хранения, также делаем привязку к порту.

Для Prometheus используется готовый образ, привязка к портам и настроен Prometheus config.

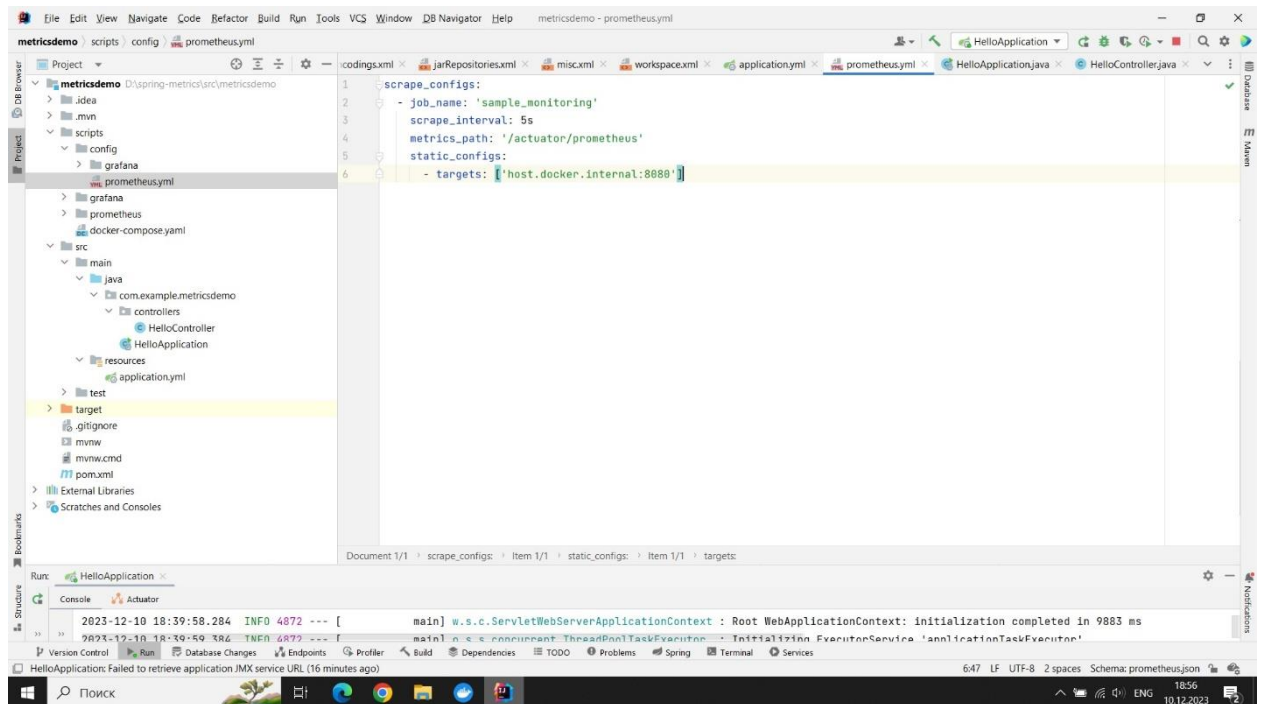


Включены по умолчанию. Это dashboard с jvm и с процессами выполнения.



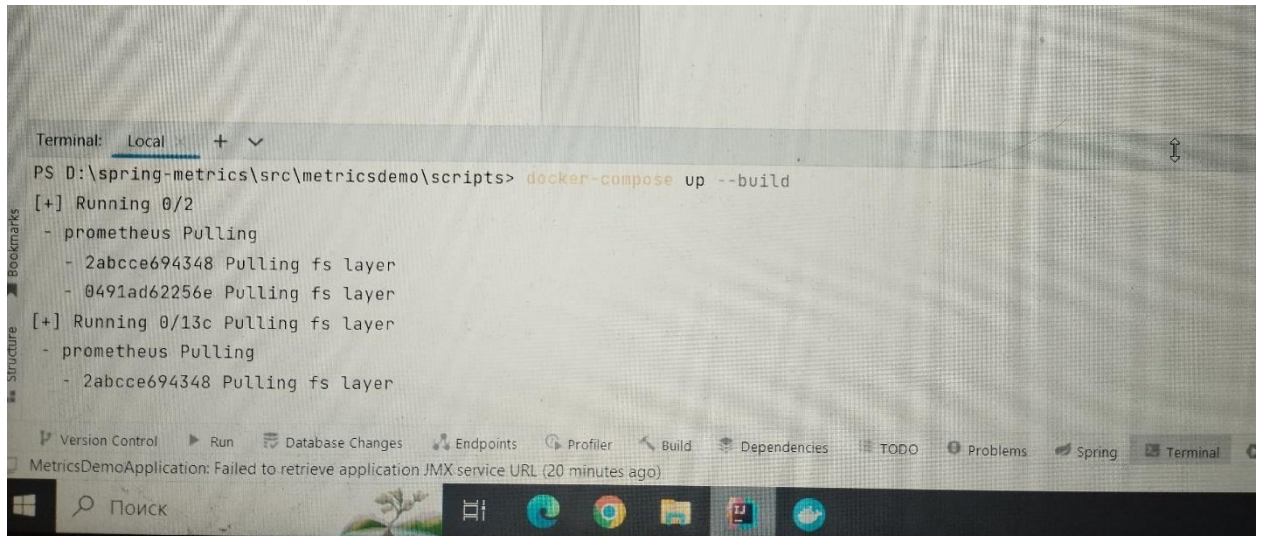
Prometheus config:

Ходит за метриками на actuator каждые 5 секунд. Маршрут – host.docker.internal – внешний адрес машины (адрес привязки к самой машине. Для того чтобы из контейнера достучаться до хоста).

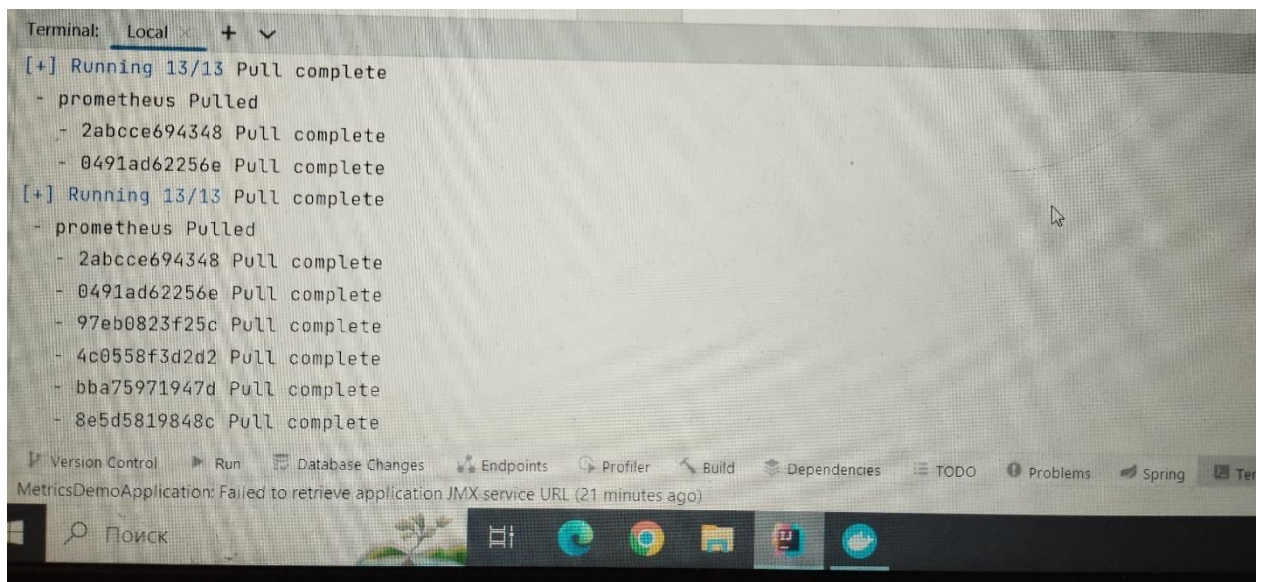


7) Запуск инфраструктуры

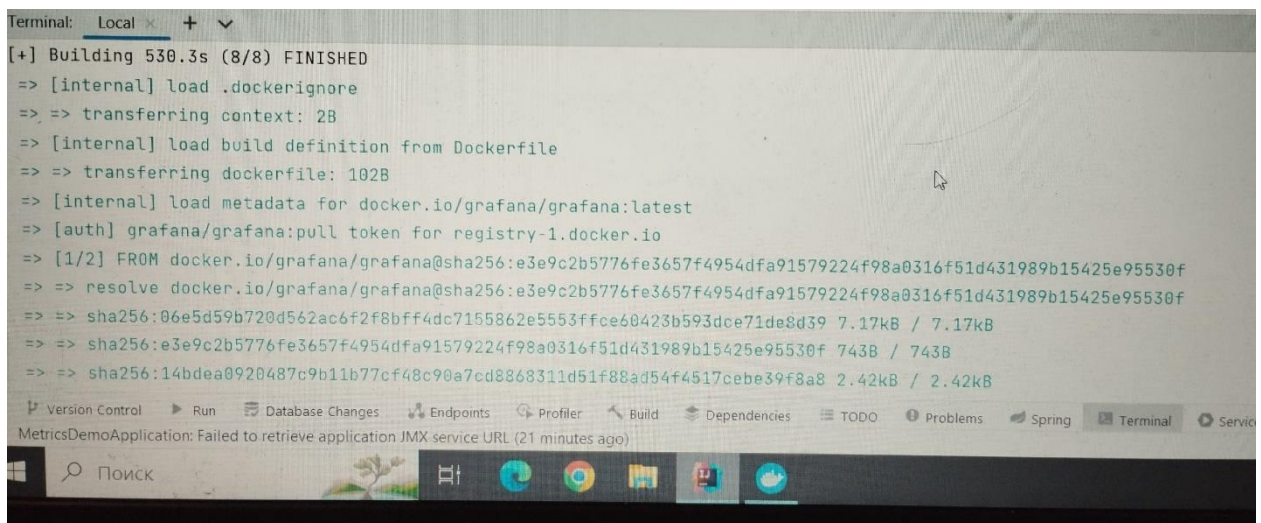
Запускаем docker-compose. Build – для явной сборки с Grafana.



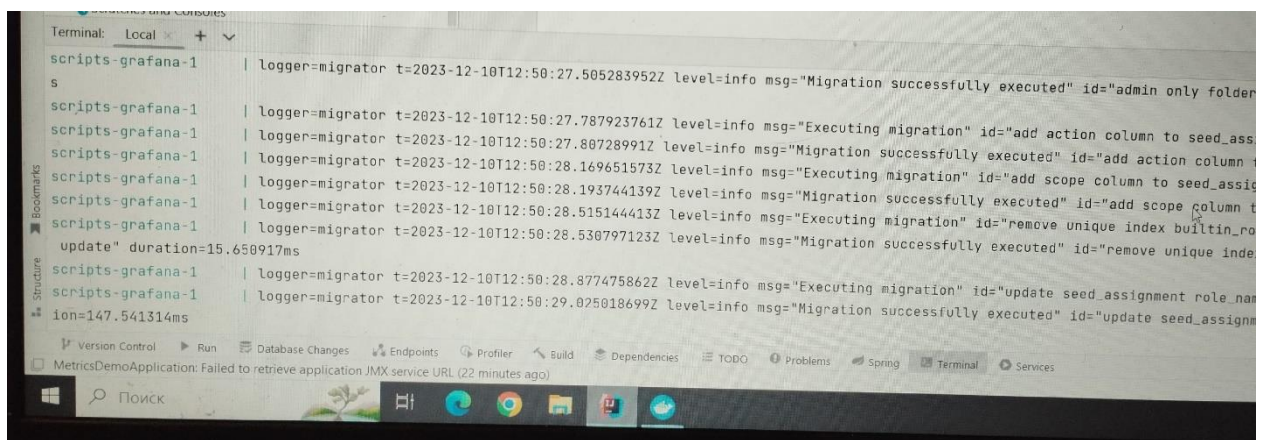
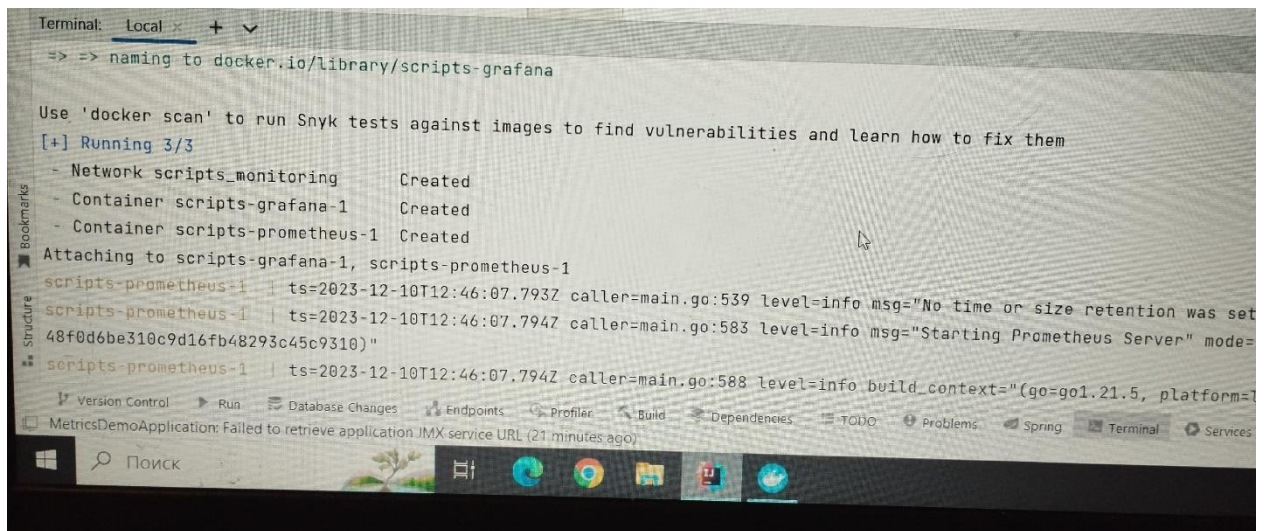
```
Terminal: Local + v
PS D:\spring-metrics\src\metricsdemo\scripts> docker-compose up --build
[+] Running 0/2
- prometheus Pulling
  - 2abcce694348 Pulling fs layer
  - 0491ad62256e Pulling fs layer
[+] Running 0/13c Pulling fs layer
- prometheus Pulling
  - 2abcce694348 Pulling fs layer
```



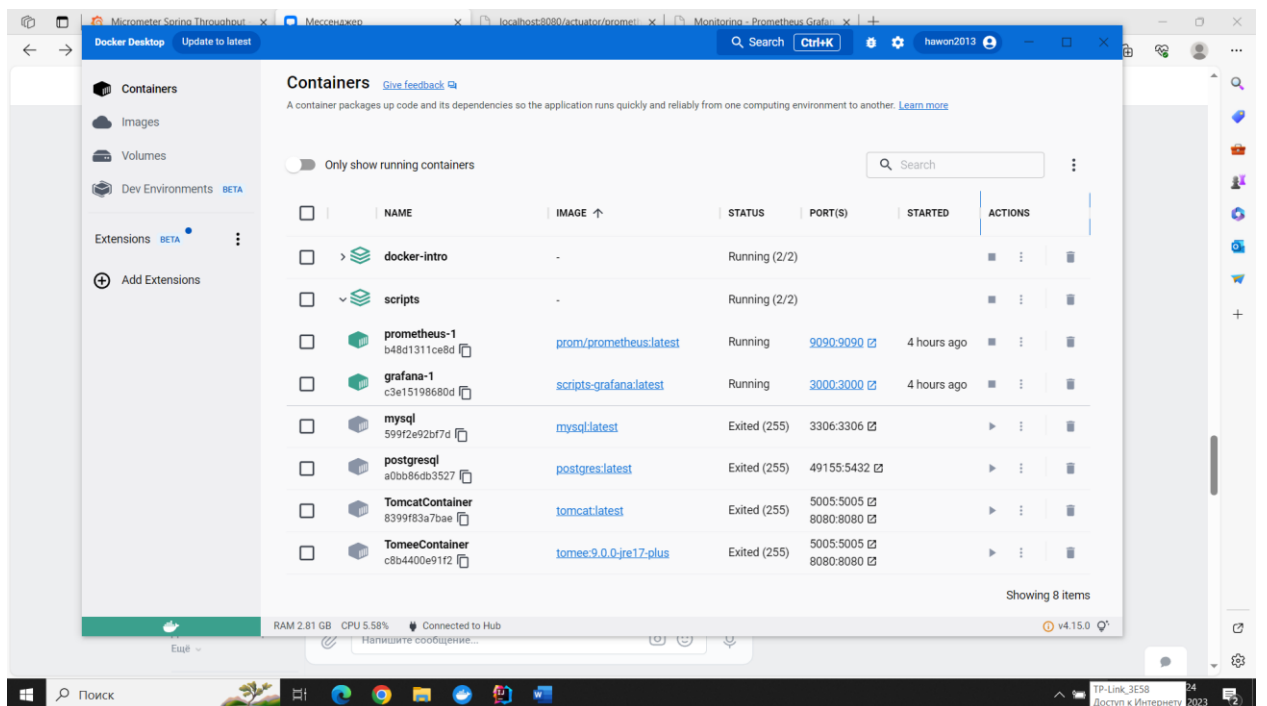
```
Terminal: Local x + v
[+] Running 13/13 Pull complete
- prometheus Pulled
  - 2abcce694348 Pull complete
  - 0491ad62256e Pull complete
[+] Running 13/13 Pull complete
- prometheus Pulled
  - 2abcce694348 Pull complete
  - 0491ad62256e Pull complete
  - 97eb0823f25c Pull complete
  - 4c0558f3d2d2 Pull complete
  - bba75971947d Pull complete
  - 8e5d5819848c Pull complete
```



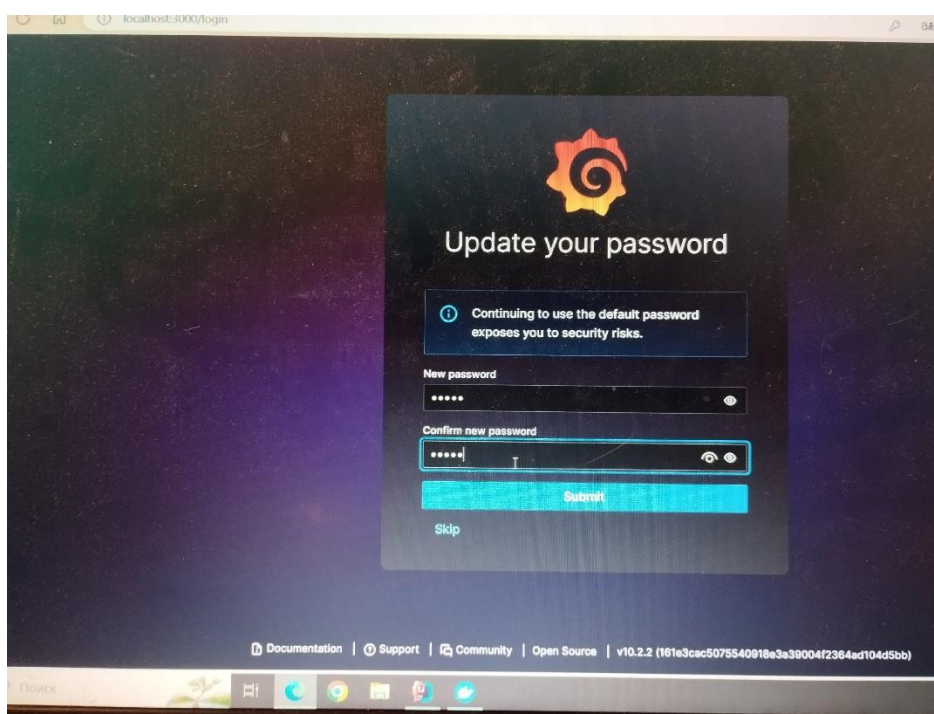
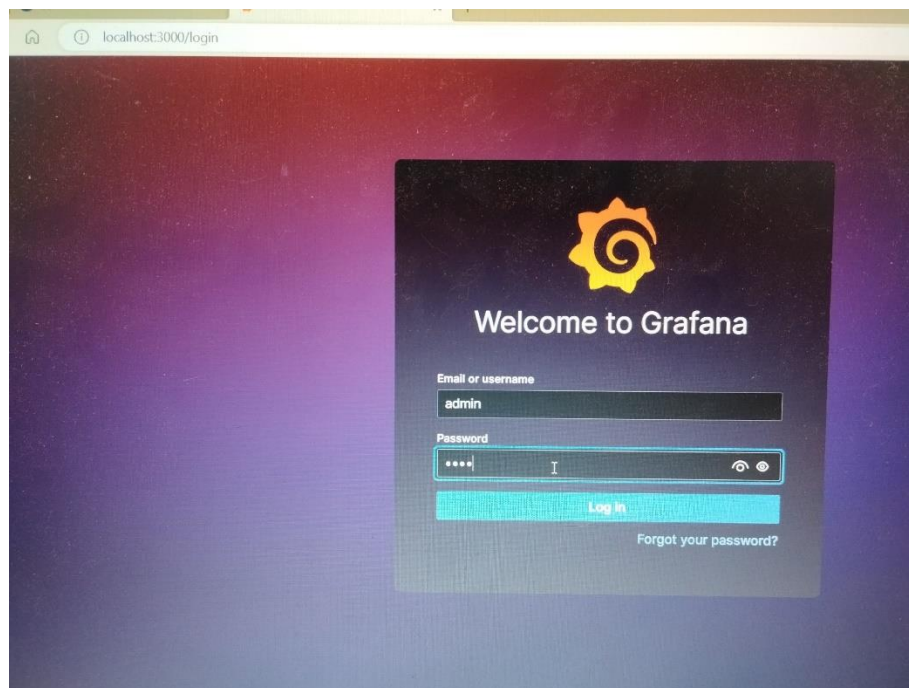
```
Terminal: Local x + v
[+] Building 530.3s (8/8) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 102B
=> [internal] load metadata for docker.io/grafana/grafana:latest
=> [auth] grafana/grafana:pull token for registry-1.docker.io
=> [1/2] FROM docker.io/grafana/grafana@sha256:e3e9c2b5776fe3657f4954dfa91579224f98a0316f51d431989b15425e95530f
=> => resolve docker.io/grafana/grafana@sha256:e3e9c2b5776fe3657f4954dfa91579224f98a0316f51d431989b15425e95530f
=> => sha256:06e5d59b720d562ac6f2f8bffa4dc7155862e5553ffce60423b593dce71de8d39 7.17kB / 7.17kB
=> => sha256:e3e9c2b5776fe3657f4954dfa91579224f98a0316f51d431989b15425e95530f 743B / 743B
=> => sha256:14bdea0920487c9b11b77cf48c90a7cd8868311d51f88ad54f4517cebe39f8a8 2.42kB / 2.42kB
```

Docker:



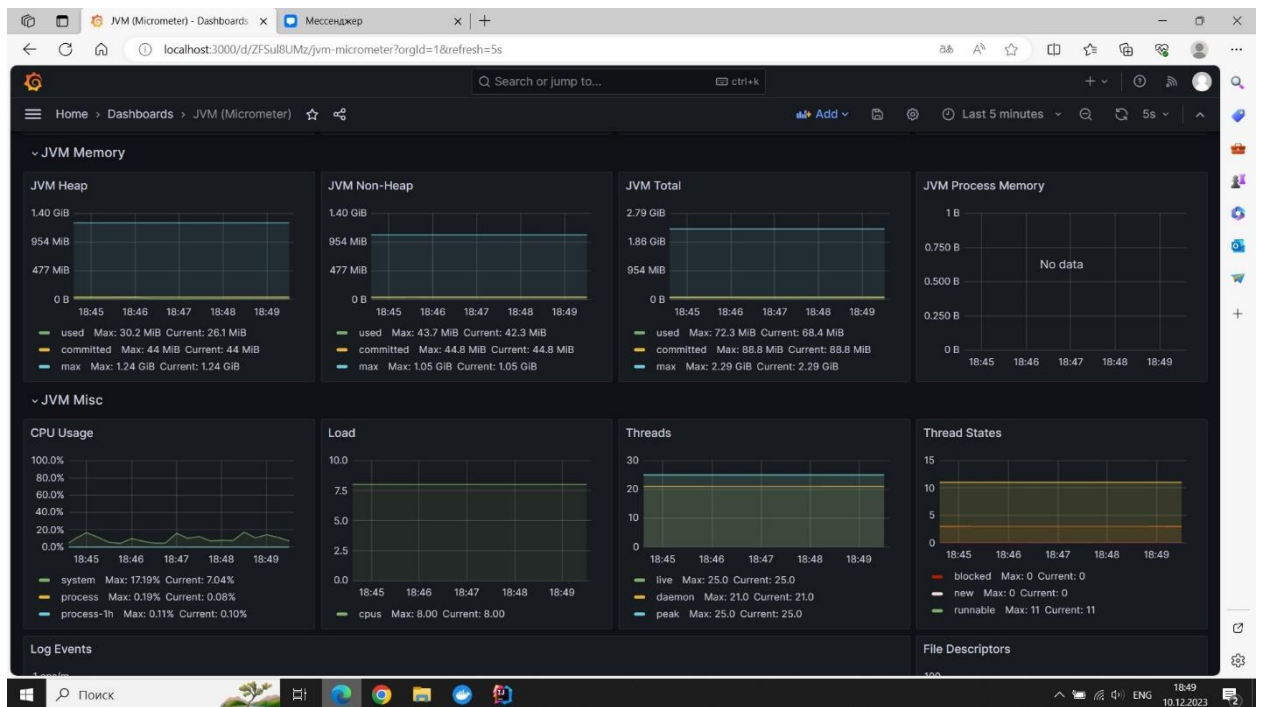
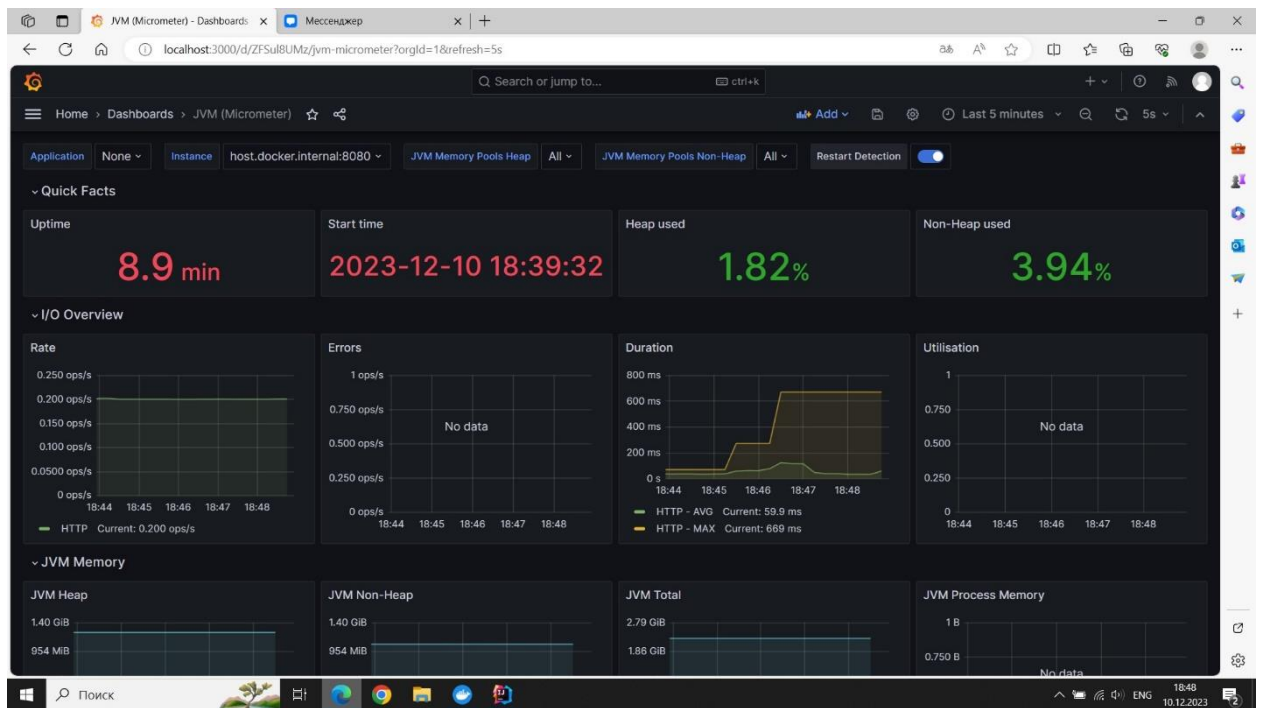
8) Заходим в Grafana (localhost:3000/login)

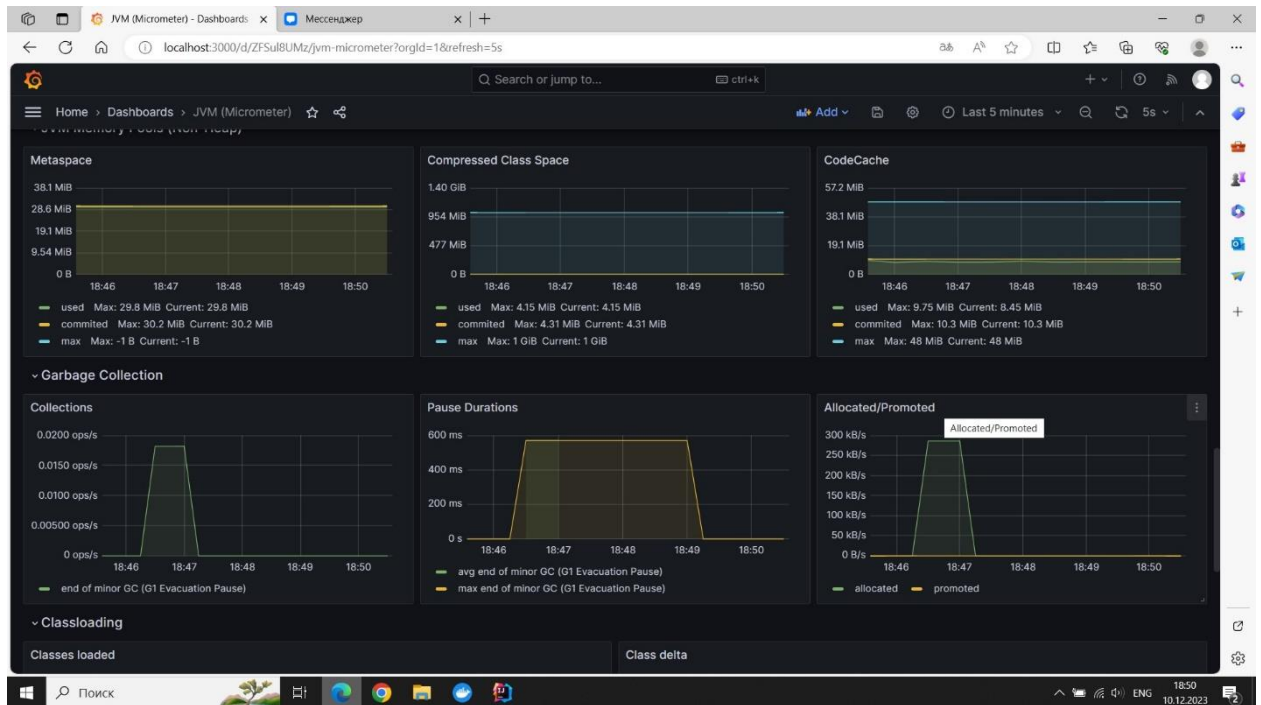
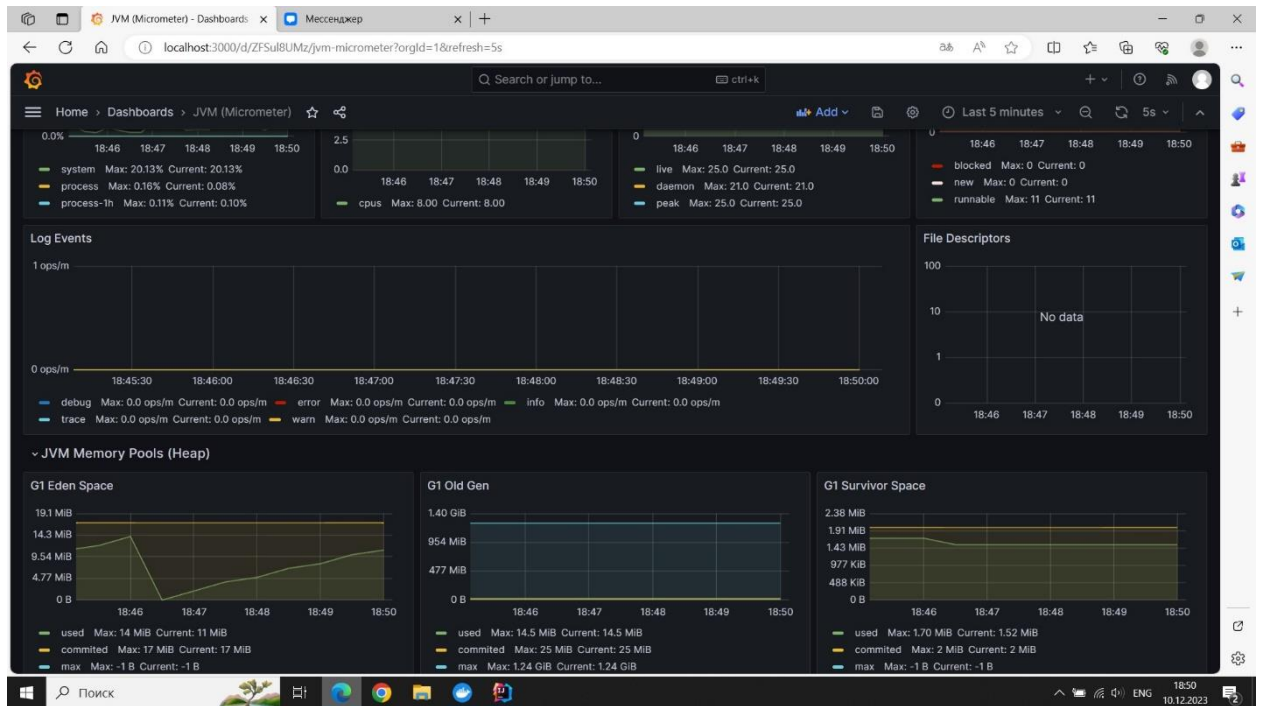


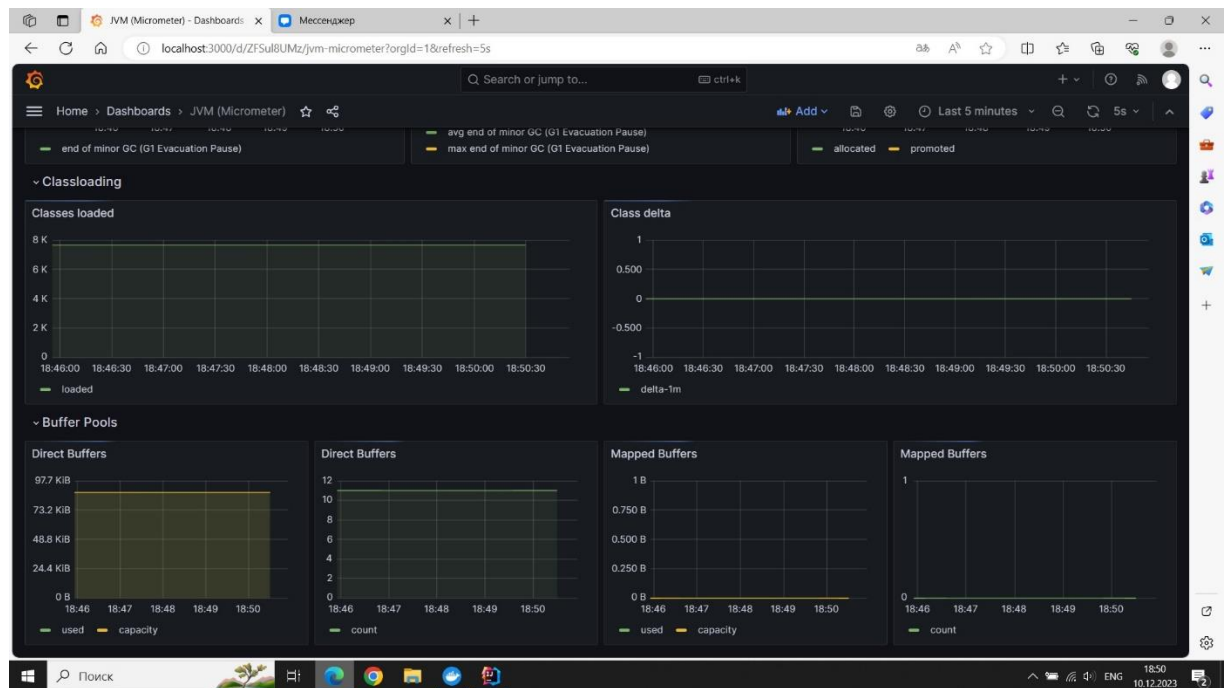
Открываем Dashboard микрометр, сначала запустив приложение.

8) Проверка метрик

Grafana graphs для jvm







Grafana graphs для пропускной способности (response throughput)

