Департамент образования и науки города Москвы
Государственное автономное образовательное учреждение высшего образования города Москвы
«Московский городской педагогический университет»
Институт цифрового образования
Департамент информатики, управления и технологий

Макарова Екатерина Павловна

ЛАБОРАТОРНАЯ РАБОТА 3.1.

Docker Compose для мультиконтейнерных приложений

Интеграция и развертывание программного обеспечения с помощью контейнеров

Направление подготовки
38.03.05 Бизнес-информатика
Профиль подготовки
Аналитика данных и эффективное управление

Курс обучения: 4
Форма обучения: очная
Преподаватель: кандидат технических наук,
доцент Босенко Тимур Муртазович

Москва
2025

Цель работы: освоить использование Docker Compose для управления многоконтейнерными приложениями.

Задачи:

1. Создать файл docker-compose.yml для указанного многоконтейнерного приложения.

2. Запустить приложение с помощью Docker Compose.

3. Проверить работоспособность приложения и взаимодействие между контейнерами.

4. Выполнить индивидуальное задание.

Вариант 8 (st_95):

1. Создать файл docker-compose.yml для системы управления проектами (Ruby on Rails + MySQL).
2. Запустить приложение и проверить создание задач.
3. Реализовать расчет процента выполнения проектов.

Ход работы

Общее задание:

Постановка задачи:

1. Генерирует синтетические данные о продажах в реальном времени
2. Сохраняет данные в Redis
3. Визуализирует статистику с помощью Grafana
4. Обспечивает мониторинг системы через Prometheus
5. Разворачивается через Docker Compose

Технический стек:

Backend: Node.js + Express.js

База данных Redis

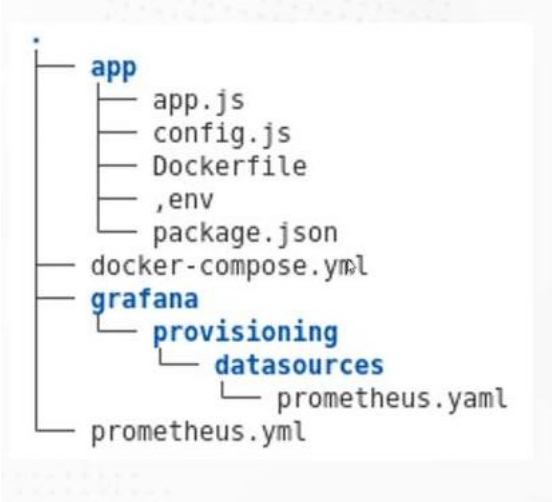
Визуализация: Grafana

Мониторинг: Prometheus

Экспортер метрик: Redis Exporter

Контейниризация: Docker + Docker Compose

Дерево проекта:



Функциональные требования:

1. Генерация данных:

   Создание записей о продажах каждые 5 секунд

   Случайный выбор товара из предопределённого списка

Генерация случайного количества и стоимости

2. Хранение данных:

   Сохранение всех транзакций в Redis

   Ограничение хранения последних 1000 записей

   Структурированное хранение в формате JSON

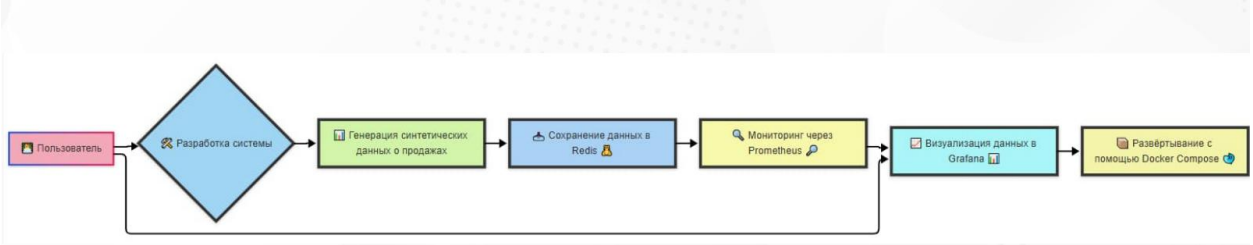
3. Визуализация:

   Графики продаж по времени

   Статистика по товарам

   Средний чек

   Общая выручка



Индивидуальное задание:

Постановка задач:

1. Генерирует синтетические данные о проектах
2. Сохраняет данные в MySQL
3. Визуализирует статистику с помощью Grafana
4. Обспечивает мониторинг системы через Prometheus
5. Разворачивается через Docker Compose

Технический стек:

Backend: Ruby on Rails

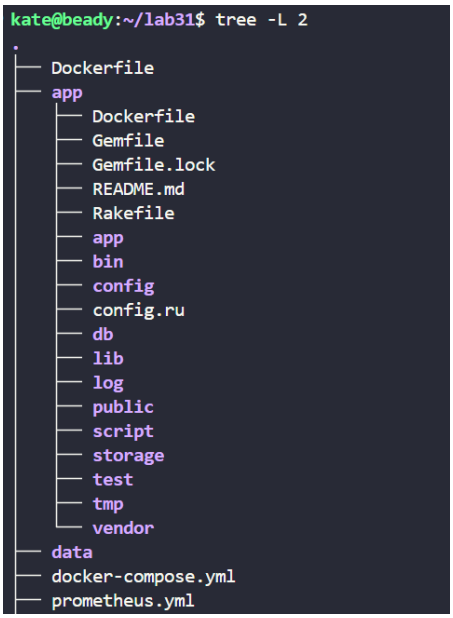
База данных: MySQL

Фоновые задачи: Sidekiq + Redis

Контейнеризация: Docker + Docker Compose

Мониторинг: Prometheus + Grafana

Дерево проекта:



Функциональные требования:

4. Генерация данных:

Создание записей о продажах каждые 5 секунд

Случайный выбор товара из предопределённого списка

Генерация случайного количества и стоимости

5. Хранение данных:

   Сохранение всех транзакций

Ход работы:

1. **Создание директории и переход в неё (Рис. 1).**

```
kate@beady:~$ mkdir LAB_3_1
kate@beady:~$ cd LAB_3_1
kate@beady:~/LAB_3_1$
```

Рис. 1

2. **Далее создан файл docker-compose.yml (Рис. 2)**

```
docker-compose.yml
Run All Services
1  services:
2
   Run Service
3    bootstrap:
4      image: ruby
5      volumes:
6        - ./data:/home
```

Рис. 2

3. **Для формирования ruby окружения был создан контейнер (Рис. 3)**

```
Run Service
bootstrap:
  image: ruby
  user: 1000:1000
  volumes:
    - ./data:/home
Run Service
```

Рис. 3

Далее контейнер был запущен с помощью команды (Рис. 4).

```
kate@beady:~/lab31$ docker compose run --rm -it bootstrap bash
root@be695070f658:/# gem install rails
Fetching thor-1.3.2.gem
Fetching rack-3.1.11.gem
Fetching zeitwerk-2.7.2.gem
Fetching rackup-2.2.1.gem
Fetching concurrent-ruby-1.3.5.gem
Fetching tzinfo-2.0.6.gem
Fetching i18n-1.14.7.gem
Fetching connection_pool-2.5.0.gem
```

Рис. 4

Создание структуры проекта (Рис. 5)

```
root@be695070f658:/# rails new app --api
Based on the specified options, the following options will also be activated:

  --skip-javascript [due to --api]
  --skip-hotwire [due to --skip-javascript]
  --skip-asset-pipeline [due to --api]

      create
      create  README.md
      create  Rakefile
      create  .ruby-version
      create  config.ru
      create  .gitignore
      create  .gitattributes
      create  Gemfile
         run  git init -b main from "."
Initialized empty Git repository in /app/.git/
```

Рис. 5

Созданная структура в директории (Рис. 6).

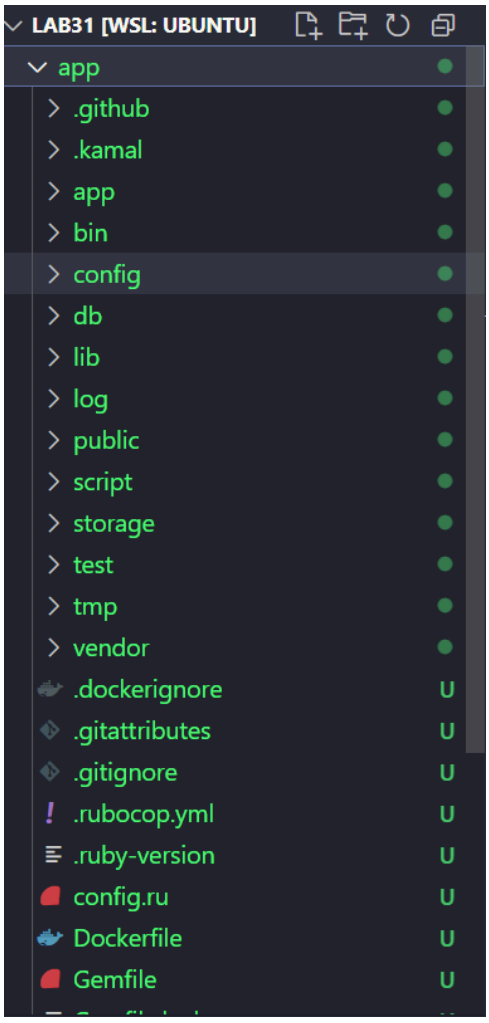


Рис. 6

4. **Далее необходимо установить зависимости:**

bundler add mysql2

bundler add prometheus_exporter

bundle add sidekiq

Установленные зависимости отобразятся в файлах Genifile

app > Gemfile

```
50
51    gem "mysql2", "~> 0.5.6"
52
53    gem "prometheus_exporter", "~> 2.2"
54
55    gem "sidekiq", "~> 8.0"
56
```

```
kate@beady:~/Lab_3_1/individual/project$ docker build -f dev.Dockerfile --tag test .
[+] Building 142.6s (12/12) FINISHED                                                    docker:default
 => [internal] load build definition from dev.Dockerfile                                         0.0s
 => => transferring dockerfile: 660B                                                             0.0s
 => [internal] load metadata for docker.io/library/ruby:3                                        1.2s
 => [auth] library/ruby:pull token for registry-1.docker.io                                      0.0s
 => [internal] load .dockerignore                                                                0.0s
 => => transferring context: 2B                                                                  0.0s
 => [1/7] FROM docker.io/library/ruby:3@sha256:d780a9b44dc5e57a4968d3e4b086bbdd9b595f06d747da10b29e568386fe1dd9   0.0s
 => => resolve docker.io/library/ruby:3@sha256:d780a9b44dc5e57a4968d3e4b086bbdd9b595f06d747da10b29e568386fe1dd9   0.0s
 => CACHED [2/7] RUN apt-get update -qq && apt-get install -y build-essential                    0.0s
 => [3/7] RUN gem install rails                                                                 17.2s
 => [4/7] RUN mkdir /myapp                                                                       0.4s
```

```
kate@beady:~/Lab_3_1/individual/project$ docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
test                      latest    c2ad0745247d   27 seconds ago   1.8GB
main-app                  latest    81f77b906819   2 hours ago      200MB
oliver006/redis_exporter  latest    e7e96895407a   4 hours ago      14.3MB
prom/prometheus           latest    6927e0919a14   9 days ago       415MB
rust-app                  latest    3d5a96d366b1   11 days ago      140MB
flask-app                 latest    3732af5fd667   12 days ago      206MB
grafana/grafana           latest    8b37a2f028f1   2 weeks ago      744MB
postgres                  latest    0321e2252ebf   2 weeks ago      621MB
hello-world               latest    e0b569a5163a   6 weeks ago      20.4kB
redis                     latest    6aafb7f25fc9   8 weeks ago      173MB
mongo                     latest    aaad67f2dca9   3 months ago     1.18GB
```

```
kate@beady:~/Lab_3_1/individual/project$ docker run --rm test
=> Booting Puma
=> Rails 8.0.1 application starting in development
=> Run `bin/rails server --help` for more startup options
Puma starting in single mode...
* Puma version: 6.6.0 ("Return to Forever")
* Ruby version: ruby 3.4.2 (2025-02-15 revision d2930f8e7a) +YJIT +PRISM [x86_64-linux]
*  Min threads: 3
*  Max threads: 3
*  Environment: development
*          PID: 1
* Listening on http://0.0.0.0:3000
```

Внутри контейнера был запущен prometheus_exporter
bundle exec prometheus_exporter -b 0.0.0.0
Листинг Prometheus.yml

prometheus.yml

```yaml
global:
  scrape_interval: 15s # when Prometheus is pulling data from exporters etc
  evaluation_interval: 30s # time between each evaluation of Prometheus' alerting rules

scrape_configs:
  - job_name: web # your project name
    static_configs:
      - targets:
          - web:9394
```

Листинг Dockerfile

Dockerfile > ...

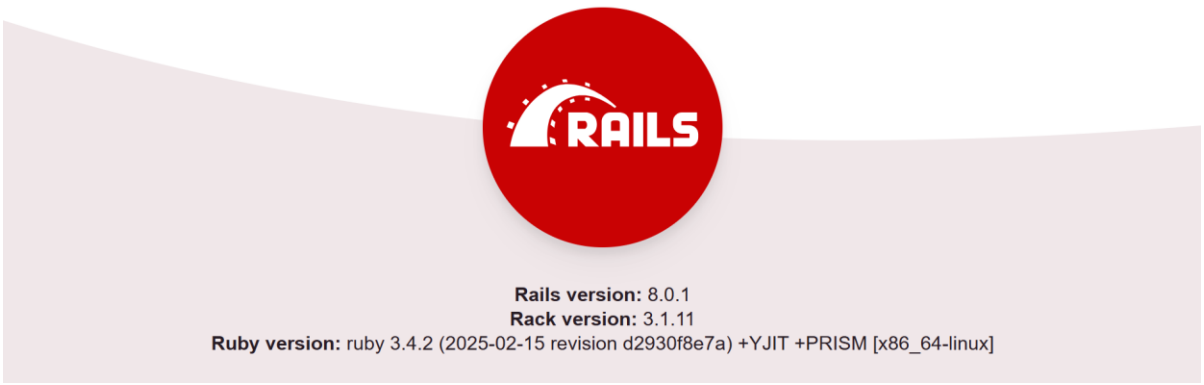
```dockerfile
# Используем официальный образ Ruby
FROM ruby

# Устанавливаем зависимости
# RUN apt-get update -qq && apt-get install -y build-essential

# Создаем директорию для приложения
RUN mkdir /myapp
WORKDIR /myapp

# Копируем Gemfile и Gemfile.lock (сохранение слоёв, чтобы по 100 раз не ставить зависимости с нуля по 100)
COPY app/Gemfile /myapp/Gemfile
COPY app/Gemfile.lock /myapp/Gemfile.lock
RUN bundle install

COPY app /myapp

# Команда для запуска сервера
# CMD ["./start.sh"]
CMD ["rails", "server", "-b", "0.0.0.0"]
```

Листинг docker-compose.yml

docker-compose.yml

```yaml
services:
  web:
    build: .
    command: bundle exec rails s -b '0.0.0.0'
    user: 1000:1000
    ports:
      - "8000:3000"
      - "9394:9394" # prometheus exporter
    depends_on:
      - db
    environment:
      - RAILS_ENV=development
    volumes:
      - ./app:/myapp
  db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: myapp_development
      MYSQL_USER: myapp
      MYSQL_PASSWORD: password
    # volumes:
    #   - db_data:/var/lib/mysql
    ports:
      - "3306:3306"

  grafana:
    hostname: grafana
    image: grafana/grafana
    ports:
      - 3000:3000
  prometheus:
    image: prom/prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
```

Проверка подключения к Ruby on Rails



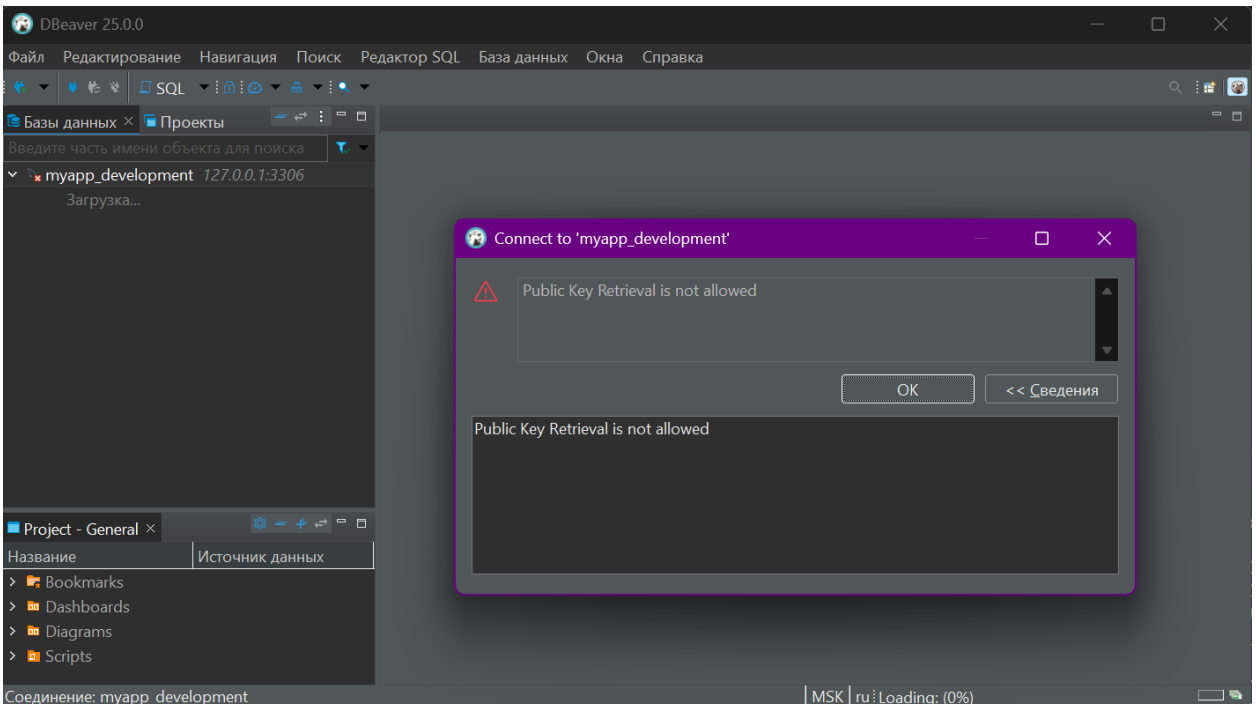
Листинг database.yml

app > config > ! database.yml

```yaml
# SQLite. Versions 3.8.0 and up are supported.
#   gem install sqlite3
#
#   Ensure the SQLite 3 gem is defined in your Gemfile
#   gem "sqlite3"
#
default: &default
  adapter: mysql2
  encoding: utf8mb4
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
  username: myapp
  password: password
  host: db

development:
  <<: *default
  database: myapp_development

test:
  <<: *default
  database: myapp_test

# Store production database in the storage/ directory, which by default
# is mounted as a persistent Docker volume in config/deploy.yml.
production:
  primary:
    <<: *default
    database: storage/production.sqlite3
  cache:
    <<: *default
    database: storage/production_cache.sqlite3
    migrations_paths: db/cache_migrate
  queue:
    <<: *default
    database: storage/production_queue.sqlite3
    migrations_paths: db/queue_migrate
  cable:
    <<: *default
    database: storage/production_cable.sqlite3
    migrations_paths: db/cable_migrate
```

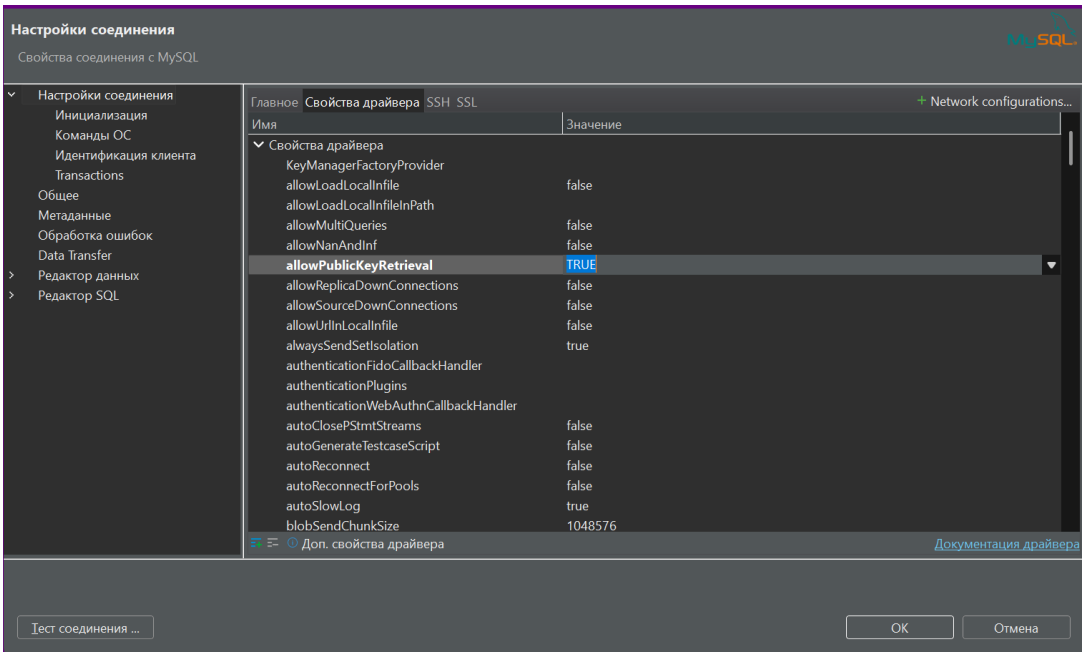
Подключение к MySQL с помощью Dbeaver.

Возникла ошибка с разрешением публичного ключа

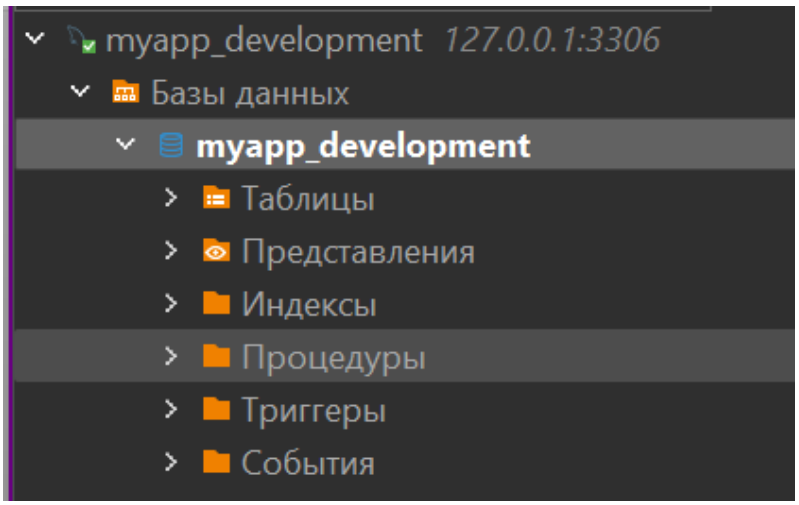


Решение ошибки:

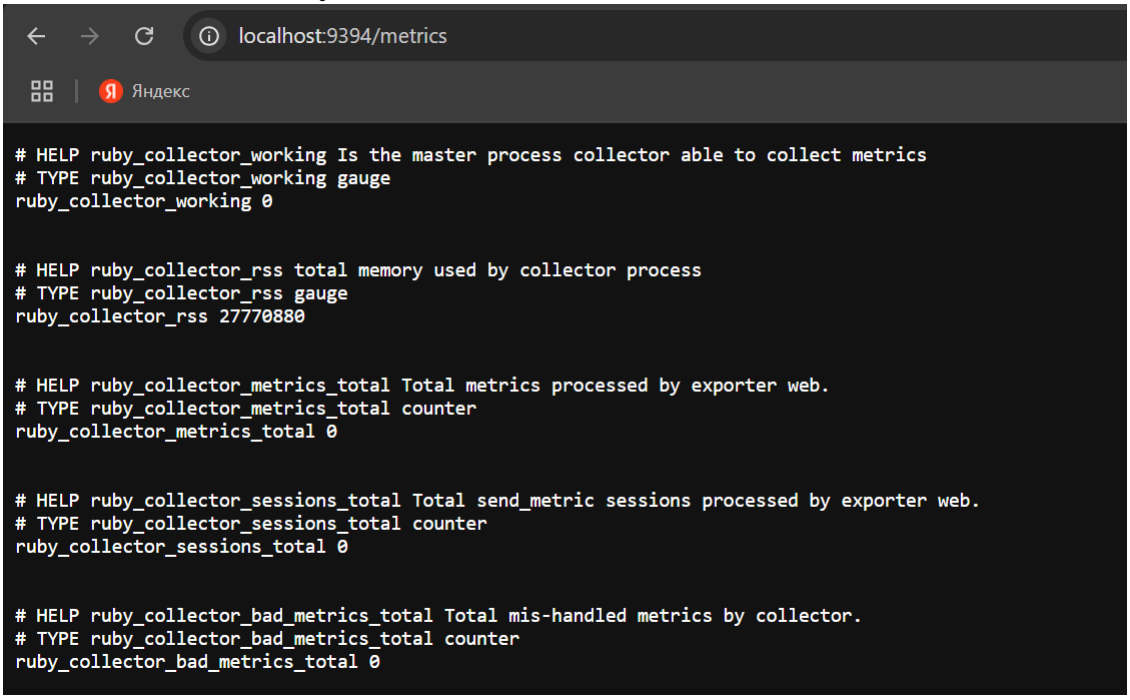
Проставить True в allowPublicKey



Успешное подключение к MySQL

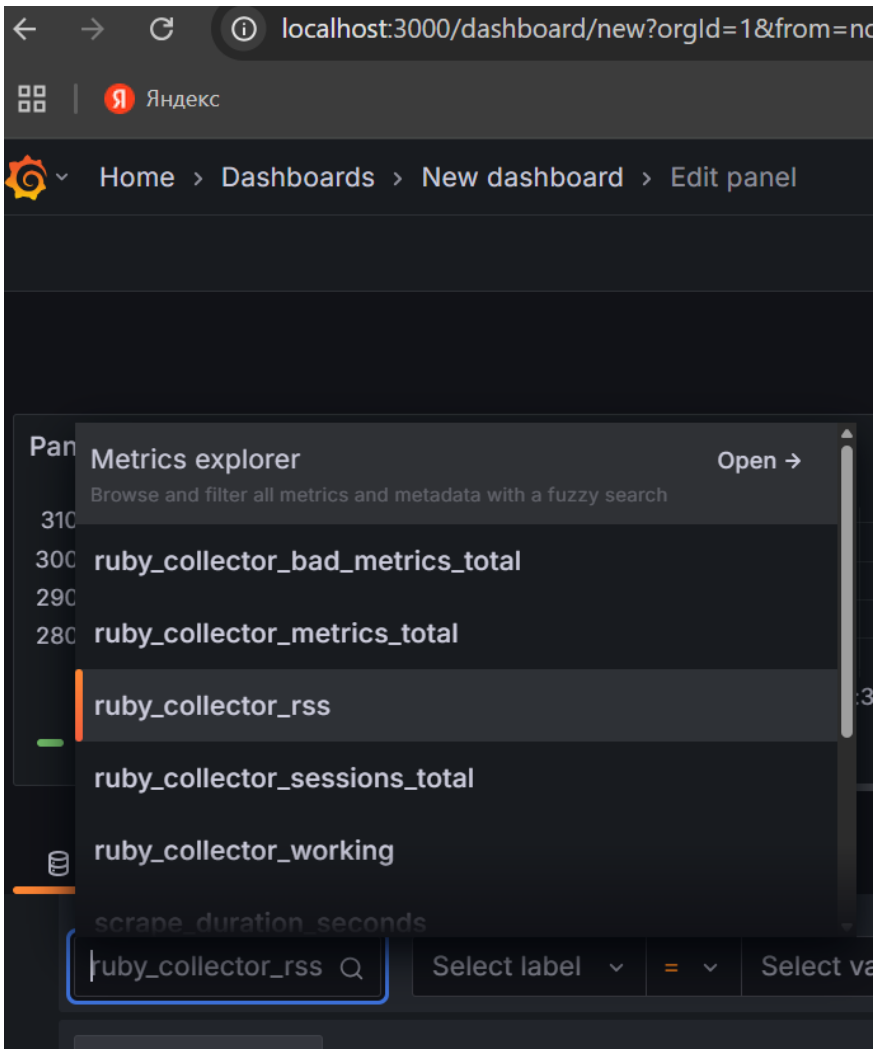


Проверка подключения к Prometheus: получение метрик Prometheus из Ruby

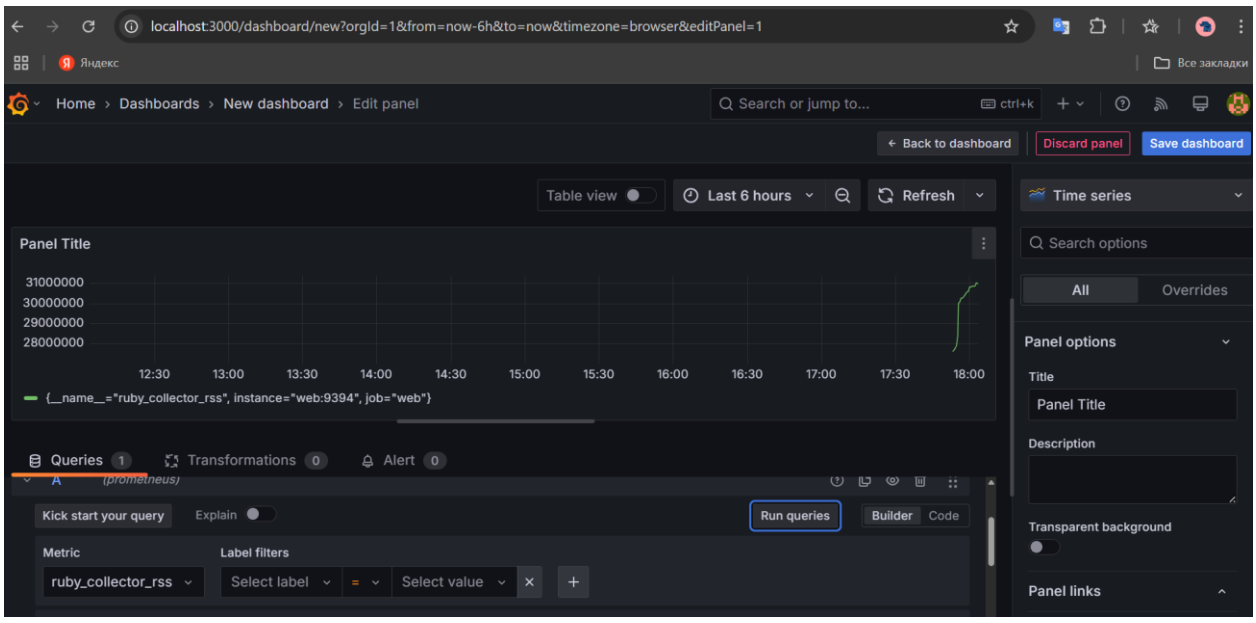


Подключения к Grafana для визуализации метрик

Была добавлена метрика в дашборд



Отображение метрики в дашборде



Контрольные вопросы

1. Что такое Docker Compose и для чего он используется?

Docker Compose — это инструмент, который позволяет управлять многоконтейнерными приложениями. Он используется для одновременного запуска и управления несколькими контейнерами, входящими в состав приложения, что особенно полезно для сложных приложений с несколькими взаимосвязанными сервисами. Docker Compose работает на основе файла конфигурации docker-compose.yml, в котором описываются все необходимые настройки и зависимости между контейнерами.

2. Какие основные преимущества использования Docker Compose для управления многоконтейнерными приложениями?

- Упрощение управления: Docker Compose позволяет управлять несколькими контейнерами через один файл конфигурации (docker-compose.yml), что упрощает настройку и запуск сложных приложений.
- Автоматизация: Compose автоматически создает и запускает все сервисы, указанные в конфигурации, в правильном порядке, учитывая зависимости между контейнерами.
- Изоляция окружения: Каждый сервис (контейнер) работает в изолированном окружении, что уменьшает вероятность конфликтов между компонентами приложения.
- Масштабируемость: Compose поддерживает масштабирование сервисов, что позволяет легко увеличивать количество экземпляров контейнеров.
- Повторяемость: Конфигурация в docker-compose.yml гарантирует, что приложение будет запускаться одинаково на разных окружениях (разработка, тестирование, production).
- Удобство разработки: Compose упрощает локальную разработку, позволяя быстро запускать и останавливать все компоненты приложения.

3. Какие основные разделы и директивы используются в файле docker-compose.yml?

- version: Указывает версию формата файла (например, '3.8').
- services: Основной раздел, где описываются все сервисы (контейнеры) приложения.
- image: Указывает образ Docker, который будет использоваться для создания контейнера.
- build: Указывает путь к Dockerfile для сборки образа.
- ports: Пробрасывает порты из контейнера на хост.
- volumes: Определяет тома для хранения данных или монтирования директорий.
- environment: Задает переменные окружения.
- depends_on: Указывает зависимости между сервисами (например, база данных должна запускаться перед приложением).
- networks: Подключает сервис к определенной сети.
- command: Переопределяет команду по умолчанию для запуска контейнера.
- networks: Определяет пользовательские сети для связи между контейнерами.
- volumes: Создает именованные тома для хранения данных.

4. Как запустить многоконтейнерное приложение с помощью Docker Compose?

- Создайте файл docker-compose.yml с описанием сервисов.
- Перейдите в директорию, где находится файл docker-compose.yml.
- Выполните команду:
- bash
- Copy
- docker-compose up

5. Как остановить и удалить контейнеры, запущенные с помощью Docker Compose?

Остановка контейнеров: docker-compose down

Эта команда останавливает и удаляет контейнеры, сети и тома, созданные командой docker-compose up.

Остановка без удаления: docker-compose stop