ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ «АВТОТРЕК»

(ПО «Автотрек»)

Руководство по сборке ПО из исходных кодов

Листов 14

Москва, 2025

Содержание

1 Инструкция по сборке программного обеспечения «Автотрек»

Сборку ПО серверных компонентов рекомендуется выполнять с использованием Docker. Исходный код серверных компонентов содержит Dockerfile, необходимый для создания образов.

Приложения сторонних разработчиков, необходимые для работы Программы:

- PostgreSQL версии 15 и выше с установленными расширениями:
  - PostGIS версии 3.3.2 и выше;
  - uuid-ossp версии 1.1 и выше;
  - PG Partman версии 4.7.2 и выше (рекомендовано, но необязательно);
- S3 хранилище, например, Minio;
- RabbitMQ версии 4 и выше;
- Redis версии 7.0 и выше;
- сервер Rocket Chat версии 7.0 и выше:
  - Rocket Chat требует нереляционную базу данных MongoDB версии 7.0 и выше.
- OSRM сервер с загруженным графом для Московской области;
- Front-web сервер: рекомендуется Nginx версии 1.27 и выше.

1.1 Компоненты Программы

1.1.1 Frontend

Frontend — это клиентская часть для работы через web-browser.

После сборки проекта необходимо настроить Front-web-server для доступа к получившимся статическим файлам. Пример конфигурации содержится в самом проекте в файле `docker/nginx.conf`. Также в проекте есть Dockerfile, создающий контейнер с легковесным web-сервером, позволяющим обеспечить доступность статичных файлов (шаблонов страниц, JS-скриптов, изображений, CSS-стилей и прочего) по HTTP. Данная конфигурация направлена на простоту и скорость взаимодействия компонентов программы. Связанные с безопасностью настройки (такие, как SSL-шифрование) рекомендуется выполнять на Front-web сервере.

Front-web должен проксировать запросы от пользователей на статику данного компонента.

Пример конфигурации Nginx в качестве Front-web сервера:

```
upstream web {
  server 127.0.0.1:8080; # адрес контейнера со статикой
}

  location ~ ^/next(/.*|$) {
    proxy_pass http://web/codd$1;
  }
```

Для корректной работы Frontend необходимо, чтобы Front-web сервер проксировал запросы к статике через алиас next.

1.1.2 User Management

User Management сервис, обеспечивающих сквозную авторизацию и аутентификацию пользователей.

Для корректной работы приложения требуется .NET Runtime версии 8.x. Для сборки исполняемого проекта из исходного кода — .NET SDK версии 8.x. По результатам сборки проекта создаётся исполняемый бинарный файл.

В состав исходного кода также входит Dockerfile для создания образа с сервером приложения.

Для конфигурации сервиса необходимо определить следующие параметры среды:

- `ConnectionStrings__ApplicationContext` — параметры для доступа к Postgresql-серверу;
- `ConnectionStrings__Redis` — параметры для доступа к Redis;
- `Kestrel__Endpoints__gRPC__Url` — адрес для обслуживания запросов gRPC;
- `Kestrel__Endpoints__Http__Url` — адрес для обслуживания HTTP запросов;
- `TokenManager__PrivateKey` — секретная случайная строка для обеспечения шифрования временной стойкости для токенов доступа пользователей (ключи периодически меняются, гарантированная стойкость не требуется).

Пример конфигурации:

```
ASPNETCORE_ENVIRONMENT=Production
ConnectionStrings__ApplicationContext=Host=127.0.0.1;Port=5432;Database
=mps_production;Username=user;Password=password;Timeout=5;CommandTimeou
t=10
ConnectionStrings__Redis=127.0.0.1:6379
Kestrel__Endpoints__gRPC__Url=http://0.0.0.0:50051
Kestrel__Endpoints__Http__Url=http://0.0.0.0:9200

TokenManager__PrivateKey=GWYy3ddNjLOVowuNR0feKvtJTr8VvKNf2dyjwU9gCdWKx5
SirRI1XfHGKwR0L7NUTCVNN2gghlbktzPsIFkhKvAkg52RzAqgLrofB9Cn7nldWSerFyvRz
ZNxz0LsbEOv
```

Для обслуживания запросов пользователей рекомендуется проксировать запросы средствами Front-web сервера. Для взаимодействия между серверными компонентами программы (кроме Frontend) рекомендуется использовать прямое подключение.

Пример конфигурации Nginx в качестве Front-web сервера:

```
upstream user_management {
  server 127.0.0.1:9200; # адрес контейнера с сервером приложения
}

  location ~ ^/user-management/(.+) {
    proxy_pass http://user_management/$1$is_args$args;
  }
```

Для корректной работы Frontend необходимо, чтобы Front-web сервер проксировал запросы к сервису через алиас user-management.

1.1.3 Telemetry receiver

Telemetry receiver — сервис приёма телеметрии.

Для корректной работы приложения требуется интерпретатор Ruby версии 3.4.1+ и Ruby gems. В состав исходного кода также входить Dockerfile, позволяющий создать контейнер с рабочим сервером сервиса приема телеметрии.

Для запуска проекта необходим выполнить команду:

```
bundle exec puma -p [Номер порта для приёма телеметрии]
```

Для конфигурации сервиса необходимо определить следующие параметры среды:

- `SETTINGS__RABBITMQ__EXCHANGE_NAME` — название обменника сообщениями для передачи полученных пакетов телеметрии;
- `SETTINGS__RABBITMQ__ROUTING_KEY` — ключ маршрутизации для передачи полученных пакетов телеметрии;
- `SETTINGS__RABBITMQ__CURRENT__HOST` — ip адрес сервера RabbitMQ;
- `SETTINGS__RABBITMQ__CURRENT__USER` — имя пользователя для обращений к RabbitMQ;
- `SETTINGS__RABBITMQ__CURRENT__PASSWORD` — пароль для обращений к RabbitMQ;
- `SETTINGS__RABBITMQ__CURRENT__VHOST` — название внутреннего виртуального хоста RabbitMQ.

Пример конфигурации:

```
SETTINGS__RABBITMQ__EXCHANGE_NAME=mps.core.telemetry
SETTINGS__RABBITMQ__ROUTING_KEY=telemetry.data
SETTINGS__RABBITMQ__CURRENT__HOST=127.0.0.1
SETTINGS__RABBITMQ__CURRENT__USER=user
SETTINGS__RABBITMQ__CURRENT__PASSWORD=password
SETTINGS__RABBITMQ__CURRENT__VHOST=mps.production
```

Для корректного взаимодействия сервисов важно чтобы название очереди соответствовало шаблону `[prefix]core.telemetry` где `[prefix]` соответствует параметру `SETTINGS__RABBITMQ__CURRENT__PREFIX` основного backend сервера (см. далее пп. 1.1.4 Core service) а параметр `SETTINGS__RABBITMQ__ROUTING_KEY` должен быть `telemetry.data`.

Для приема пакетов телеметрии необходимо проксировать запросы средствами Front-web сервера.

Пример конфигурации Nginx в качестве Front-web сервера:

```
upstream telemetry-receiver {
  server 127.0.0.1:8081; # адрес контейнера с приемщиком телеметрии
```

```
}

  location ~ ^/telemetry-receiver {
    proxy_pass http://telemetry-receiver;
  }
```

1.1.4 Core service

Core service — основной сервис бизнес-логики.

Для корректной работы приложения требуется интерпретатор Ruby версии 3.4.1+ и Ruby gems. В состав исходного кода также входить Dockerfile, позволяющий создать контейнер с рабочим сервером сервиса приема телеметрии.

Для запуска проекта необходим запускать три инстанса приложения:

- сервер приложения `bundle exec puma -p [Номер порта для обработки запросов пользователей]`;
- сервер выполнения отложенных и регулярных команд `bundle exec sidekiq`;
- сервер асинхронного выполнения команд и обработки внутренних событий `bundle exec rake sneakers:run`.

Все три инстанса используют одинаковые настройки.

Для конфигурации сервиса необходимо определить следующие параметры среды:

- `SETTINGS__SYSTEM__ENABLE_REDIS_CACHE_STORE` — если задан, то для кеширования результатов некоторых часто повторяемых запросов будет использовано in-memory хранилище.
- `SETTINGS__RAILS__SECRET_KEY_BASE` — секретный ключ для шифрования некоторых данных на стороне сервера.
- настройки взаимодействия с RabbitMQ. Количество предзагружаемых из очереди сообщений, количество разделяемых тредов, общее количество тредов и количество копий процесса:
  - `SETTINGS__SNEAKERS__PREFETCH`.
  - `SETTINGS__SNEAKERS__SHARE_THREADS`.
  - `SETTINGS__SNEAKERS__THREADS`.
  - `SETTINGS__SNEAKERS__WORKERS`.
- Параметры для доступа к FTP-серверу оператора электронных путевых листов:
  - `SETTINGS__EASSIGNMENT_FTP__HOST`.

- SETTINGS__EASSIGNMENT_FTP__USER.
- SETTINGS__EASSIGNMENT_FTP__PASSWORD.

- Параметры доступа к S3 сервису для хранения загружаемых файлов:
  - SETTINGS__MINIO__ENDPOINT.
  - SETTINGS__MINIO__SECRET_ACCESS_KEY.
  - SETTINGS__MINIO__ACCESS_KEY_ID.
  - SETTINGS__MINIO__BUCKET.
- Параметры URL сервиса, по которому доступен Сервис Управления Пользователями и секретный (системный) токен:
  - SETTINGS__USER_MANAGEMENT__URL.
  - SETTINGS__USER_MANAGEMENT__TOKEN.
- Параметры доступа к базе данных:
  - SETTINGS__POSTGRESQL__ADAPTER.
  - SETTINGS__POSTGRESQL__ENCODING.
  - SETTINGS__POSTGRESQL__HOST.
  - SETTINGS__POSTGRESQL__PORT.
  - SETTINGS__POSTGRESQL__USER.
  - SETTINGS__POSTGRESQL__PASSWORD.
  - SETTINGS__POSTGRESQL__DATABASE.
  - SETTINGS__POSTGRESQL__POOL.
  - SETTINGS__POSTGRESQL__TIMEOUT.
- Параметры доступа к серверу RabbitMQ (также см. описание сервиса приёма телеметрии в пп. 1.1.3 Telemetry receiver):
  - SETTINGS__RABBITMQ__CURRENT__HOST.
  - SETTINGS__RABBITMQ__CURRENT__USER.
  - SETTINGS__RABBITMQ__CURRENT__PASSWORD.
  - SETTINGS__RABBITMQ__CURRENT__VHOST.
  - SETTINGS__RABBITMQ__CURRENT__EXCHANGES__TELEMETRY.
- Параметры доступа к Redis:
  - SETTINGS__REDIS__HOST.
  - SETTINGS__REDIS__PORT.
  - SETTINGS__REDIS__DB.

– Параметры доступа к серверу Rocket.Chat для автоматического создания учётных записей пользователей и бесшовной аутентификации:
  – SETTINGS__ROCKETCHAT__HOST.
  – SETTINGS__ROCKETCHAT__PORT.
  – SETTINGS__ROCKETCHAT__USERNAME.
  – SETTINGS__ROCKETCHAT__PASSWORD.

Для обслуживания запросов пользователей рекомендуется проксировать запросы средствами Front-web сервера.

Пример конфигурации Nginx в качестве Front-web сервера:

```
upstream api {
  server 127.0.0.1:3000;
}

  # API
  location ~
^/(api|rails|photos|esmo|print|directories|mayor|rnis|routing-
service|stream) {
    proxy_pass http://api;
  }

  # WebSocket
  location ~ ^/socket {
    proxy_pass_request_headers on;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forward-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forward-Proto http;
    proxy_set_header X-Nginx-Proxy true;

    proxy_pass http://api;
    proxy_http_version 1.1;
    proxy_read_timeout 180s;
  }
```

Для корректной работы Frontend необходимо, чтобы Front-web сервер проксировал запросы к сервису через алиасы:

- api;
- rails;
- photos;
- esmo;
- print;
- directories;
- routing-service;
- stream;
- socket.

Пример конфигурации развёрнутого приложения на базе Docker Compose:

```
x-service: &service
  init: true
  restart: always
  env_file: .env

x-postgres: &postgres
  image: gitlab/mps/postgres:15

x-core: &core
  <<: *service
  image: gitlab/mps/core:latest

services:
  postgresql15:
    <<: *postgres
    restart: always
    container_name: postgres
    shm_size: 1g
    environment:
      POSTGRES_USER: mps
      POSTGRES_PASSWORD: 123456789
      POSTGRES_DB: mps
    volumes:
      - ./volumes/postgresql15/data:/var/lib/postgresql/data
    expose:
```

```
    - 5432

rabbitmq:
  image: rabbitmq:4.0-management
  restart: always
  environment:
    RABBITMQ_DEFAULT_USER: mps
    RABBITMQ_DEFAULT_VHOST: mps.production
    RABBITMQ_DEFAULT_PASS: 123456789
  volumes:
    - ./volumes/rabbitmq/data:/var/lib/rabbitmq
  expose:
    - 4369
    - 5671
    - 5672
    - 15671 # Management Port
    - 15672 # Management Port
    - 15674 # Port of Stomp Plugin
    - 25672


minio:
  image: quay.io/minio/minio:RELEASE.2025-01-20T14-49-07Z
  command: server --address ":9000" --console-address ":9001" /data
  restart: always
  expose:
    - 9000
    - 9001
  volumes:
    - ./volumes/minio/data:/data
  environment:
    MINIO_ROOT_USER: mps
    MINIO_ROOT_PASSWORD: 123456789

redis:
  image: redis:7
  restart: always
  volumes:
    - ./volumes/redis/data:/data
  expose:
    - 6379
```

```
  nginx:
    image: nginx:1.27
    restart: always
    volumes:
      - ./volumes/nginx/nginx.conf:/etc/nginx/nginx.conf:ro
      - ./volumes/nginx/conf.d:/etc/nginx/conf.d:ro
    ports:
      - 80:80

 web:
    image: gitlab/mps/web:latest
    restart: always
    expose:
      - 80

  user-management:
    <<: *service
    image: gitlab/mps/madi-operational-
management/usermanagement:candidate
    expose:
      - 9200
      - 50051

  core:
    <<: *core
    command: bundle exec puma

  core-sneakers:
    <<: *core
    command: bundle exec rake sneakers:run

  core-cron:
    <<: *core
    command: bundle exec sidekiq

  parkon-telemetry-receiver:
    <<: *service
    image: gitlab/mps/telemetry-receiver:latest
    command: bundle exec puma -p 80
    expose:
    - 80
```

И соответствующая конфигурация Nginx:

```
upstream web {
  server web:80;
}

upstream user_management {
  server user-management:9200;
}

upstream telemetry-receiver {
  server parkon-telemetry-receiver:80;
}

upstream api {
  server core:3000;
}

server {
  listen ssl default_server;
  server_name 53.64.98.213;
  large_client_header_buffers 100 5120k;
  client_max_body_size 100M;

  # Default proxy settings
  # ----------------------------------------------------------------------
-------

  proxy_set_header Host $http_host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header X-Forwarded-Proto $scheme;
  proxy_redirect off;

  location ~ ^/next(/.*|$) {
    proxy_pass http://web/codd$1;
  }

  location ~ ^/user-management/(.+) {
    proxy_pass http://user_management/$1$is_args$args;
  }
```

```
  location ~ ^/api/v2/telemetry/data {
    if ($request_method = POST) {
      proxy_pass http://telemetry-receiver;
    }

    if ($request_method = GET) {
      proxy_pass http://api;
    }
  }

  # API
  location ~
^/(api|rails|photos|esmo|print|directories|mayor|rnis|routing-
service|stream) {
    proxy_pass http://api;
  }

  # WebSocket
  location ~ ^/socket {
    proxy_pass_request_headers on;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forward-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forward-Proto http;
    proxy_set_header X-Nginx-Proxy true;

    proxy_pass http://api;
    proxy_http_version 1.1;
    proxy_read_timeout 180s;
  }

}
```