

**Инструкция по развертыванию экземпляра  
«Система сбора данных с различных источников»**

# 1. Введение

## 1.1. Область применения

Настоящий документ предназначен для сотрудников эксплуатирующей организации и отражает основные функциональные возможности и порядок действий при выполнении операций, связанных с администрированием программного обеспечения «Система сбора данных с различных источников» (далее - «Система»)

## 2. Установка Системы

В данном разделе будет описана установка Системы на Debian Linux. Предполагается, что были предварительно установлены также Docker, Docker Compose, Apache Kafka, а также, используемая СУБД: PostgreSQL или MySQL. Система может быть настроена на использование одной из указанных СУБД.

### 2.1. Системные требования к ПО

Минимальные аппаратные требования:

- Операционная система, способная запускать контейнеры. Предпочтительно Linux.
- Система управления контейнерной виртуализацией. Предпочтительно Docker Swarm или Kubernetes.
- Подключение к серверу очередей Apache Kafka
- Количество логических ядер процессора: 4
- Семейство процессоров: x86
- Частота процессора: 3.0. ГГц
- Объем установленной памяти: 16 Гб

#### 2.1.2. Минимальные требования к сторонним компонентам и/или системам, необходимым для установки и работы ПО

- Debian 11 (Открытая лицензия GNU)
- Docker 24.0.2 (open-source community edition)
- Apache Kafka 2.13-2.8.1 (Открытая лицензия Apache License)
- Grafana Loki 2.6.1 (Открытая лицензия GNU)
- Grafana 9.2.2 (Открытая лицензия GNU)
- PostgreSQL 14 (Открытая лицензия PostgreSQL license)
- MySQL 8.0 (open-source community edition Открытая лицензия GNU)
- Golang 1.20 (Открытая лицензия GNU)

## 2.2. Порядок установки

1. Создайте папку `/home/app`
2. Смонтируйте диск с дистрибутивом в папку `/mnt`
3. Скопируйте из дистрибутива исходники из папки `/mnt` в папку `/home/app`
4. Смените текущую папку на `/home/app` и выполните команды  
`sudo chown 10001:10001 ./volumes/loki`  
`sudo chown 472:472 ./volumes/grafana`
5. Отредактируйте файл `docker-compose.yml`, в соответствии с пунктами 3.2.1 и 3.3.1 данного документа
6. Создайте и отредактируйте файлы настроек для обоих модулей, в соответствии с пунктами 3.2.2, 3.2.3 и 3.3.2 данного документа
7. Смените текущую папку на `/home/app` и выполните в ней команду  
`docker compose -up -d --build`
8. Войдите браузером на ваш сервер на порт 3000 в систему мониторинга с пользователем `admin` и паролем `admin`. Измените пароль на безопасный.

## 3. Настройка Системы

### 3.1. Общие сведения

В данном документе приводятся примеры настройки Системы с использованием среды Docker Compose. Настройка операционной системы, сервера очередей Apache Kafka, настройка СУБД, а также возможная настройка использования систем оркестрации, находятся вне компетенции этого документа и не будут тут описаны.

Система может быть сконфигурирована на сохранение в одну из баз данных: PostgreSQL или MySQL. В зависимости от того, какая СУБД используется, должен быть запущен тот или иной модуль.

## 3.2. Модуль приема сообщений

### 3.2.1 Конфигурируемые параметры

Для корректной работы модуля приема сообщений, необходимо настроить для него следующие переменные окружения:

- `SETTINGS_FILE` - путь к файлу с настройками запросов. Файл подключается к контейнеру с помощью `volume`. В данной переменной окружения указывается локальный путь внутри контейнера, к которому был примонтирован `volume`.
- `METRICS_PORT` - порт к подсистеме проверки работоспособности.
- `HOST` - адрес сервиса, который будет слушать модуль. На этот адрес следует пробросить внешний порт или настроить проксирующий сервер для поддержки протокола `https`.
- `KAFKA_HOST` — имя хоста или IP сервера Kafka
- `KAFKA_TOPIC` - топик на сервере Kafka
- `KAFKA_PARTITION` — имя разделения (`partition`) на сервере Kafka
- `KAFKA_TIMEOUT` - таймаут запросов к серверу Kafka
- `LOG_LEVEL` - уровень логгирования. Поддерживаемые значения:
  - `error`
  - `warn`
  - `info`
  - `debug`
  - `trace`

#### Пример настройки модуля:

```
httpin:  
  build:  
    context: ./http-in/  
  restart: always  
  ports:  
    - '80:80'  
  volumes:  
    - ./http-in/config.json:/app/config.json:ro  
  environment:  
    HOST: :80  
    KAFKA_HOST: host.docker.internal:9092  
    KAFKA_TOPIC: test  
    KAFKA_PARTITION: 0  
    KAFKA_TIMEOUT: 10  
    LOG_LEVEL: trace  
    SETTINGS_FILE: /app/config.json  
  extra_hosts:  
    - "host.docker.internal:host-gateway"
```

### 3.2.2. Настройка маршрутизации

Для того, чтобы сервер обрабатывал конкретный URL, его следует прописать в файле, подключенном к модулю и указанном в переменной окружения `SETTINGS_FILE`. Этот файл содержит в текстовом формате `json` — объект, который в свою очередь, содержит еще два объекта:

- `requests` — содержит описание маршрутов и обработчики поступающих данных

Ключами объекта `requests` являются описатели маршрутов. Они формируются путем конкатенации HTTP-метода, разделителя и относительного пути.

Префиксом названия команды является HTTP-метод. Поддерживаемые методы: `GET`, `POST`, `PUT`, `DELETE`. После префикса должен находиться разделитель `|`. Затем должен быть указан маршрут. Маршрут может включать в себя как параметры, так и символ `*`, позволяющий обрабатывать различные маршруты одной командой. Примеры:

- `GET|/users`
- `GET|/users/:id`
- `GET|/users/files/*`
- `POST|/users/:id`

Значениями объекта `requests` являются строки, содержащие в себе JavaScript-код.

- `responses` — содержит обработчики ответов, отправляемых модулем клиенту.

Ключами объекта `responses` являются описатели маршрутов, описанные выше.

Значениями объекта `responses` являются строки, содержащие в себе JavaScript-код.

#### Пример настройки маршрутизации:

```
{
  "requests": { "GET|/telemetry/:id/:value": " " },
  "responses": { "GET|/telemetry/:id/:value": " " }
}
```

### 3.2.3. Обработка запросов

Для каждого поступающего запроса может быть написан JavaScript-код, который позволит модифицировать как входящие параметры, так и ответ на запрос. В силу ограничений формата json, в коде не поддерживаются переносы строк и кавычки. Эти символы могут быть вставлены в код с помощью символа экранирования, и соответственно, будут выглядеть как `\n` и `\"`.

Для обработки входящего запроса, в JavaScript передаются параметры:

- `raw_data` - RAW-содержимое запроса. Представляет собой json-закодированную строку. Передается в JavaScript как массив байт.
- `body` - тело запроса. Передается в JavaScript как массив байт.
- `cookies` - массив cookie
- `headers` - массив с заголовками запроса
- `method` - HTTP-метод
- `params` - параметры запроса в URL
- `url` - URL запроса

JavaScript-код может проанализировать эти параметры и должен вернуть параметр `raw_data`, который будет отправлен модулю доставки сообщений.

Для обработки ответа на запрос, в JavaScript передается параметр `raw_data` - RAW-содержимое ответа.

В JavaScript дополнительно передаются внешние функции:

- `atob` - преобразует строку в base64
- `btoa` - преобразует base64 в строку

Если в разделе `responses` не указать соответствующий запросу маршрут, то модуль ответит кодом 200 Ok с пустым телом ответа.

## 3.3. Модуль сохранения в PostgreSQL

### 3.3.1. Конфигурируемые параметры

- `SETTINGS_FILE` - путь к файлу с настройками запросов. Файл подключается к контейнеру с помощью `volume`. В данной переменной окружения указывается локальный путь внутри контейнера, к которому был примонтирован `volume`.
- `METRICS_PORT` - порт к подсистеме проверки работоспособности.
- `KAFKA_HOST` - имя хоста или IP адрес сервера Kafka
- `KAFKA_TOPIC` - топик на сервере Kafka
- `KAFKA_TIMEOUT` - таймаут запросов к серверу Kafka
- `KAFKA_GROUP` - группа подписчиков сервера Kafka
- `DB_HOST` — адрес сервера PostgreSQL
- `DB_PORT`— порт сервера PostgreSQL
- `DB_USER` — пользователь базы данных
- `DB_PASSWORD` — пароль пользователя базы данных
- `DB_NAME` — имя базы данных
- `DB_CONN_MAX_TIME` — устанавливает максимальное время перед переиспользованием соединения
- `DB_MAX_OPEN_CONNECTIONS` — максимальное количество соединений с базой данных
- `DB_MAX_IDLE_CONNECTIONS` — максимальное количество неиспользуемых соединений с базой данных
- `LOG_LEVEL` - уровень логгирования. Поддерживаемые значения:
  - `error`
  - `warn`
  - `info`
  - `debug`
  - `trace`

### Пример настройки модуля:

```
outpg:
  build:
    context: ./out-pg/
  restart: always
  volumes:
    - ./out-pg/config.json:/app/config.json:ro
  environment:
    KAFKA_HOST: host.docker.internal:9092
    KAFKA_TOPIC: test
    KAFKA_GROUP: consumer-group-id-1
    KAFKA_TIMEOUT: 10
    LOG_LEVEL: trace
    SETTINGS_FILE: /app/config.json
    DB_HOST: host.docker.internal
    DB_PORT: 5432
    DB_USER: postgres
    DB_PASSWORD: bG68r6229J855A7Nm1u5
    DB_NAME: test_db
  extra_hosts:
    - "host.docker.internal:host-gateway"
```

### 3.3.2. Обработка поступающих запросов

Каждый запрос, полученный от модуля приема сообщений, приходит с теми же настройками маршрутизации, которые были описаны выше в разделе 3.2.2.

Принимаемые сообщения могут быть обработаны с помощью JavaScript-кода, который позволит модифицировать входящие параметры и сформировать запрос к базе данных.

Для того, чтобы модуль обработал конкретный поступающий запрос, его следует прописать в файле, подключенном к модулю и указанном в переменной окружения `SETTINGS_FILE`. Этот файл содержит в текстовом формате `json` — объект, который в свою очередь, содержит еще объект:

- `requests` — содержит описание маршрутов и обработчики поступающих данных

Ключами объекта `requests` являются описатели маршрутов. Они должны совпадать с аналогичными маршрутами, указанными у модуля приема сообщений.

Значениями объекта `requests` являются строки, содержащие в себе JavaScript-код.

Для каждой команды должен быть написан JavaScript-код, который должен вернуть SQL-запрос к базе данных в параметре `raw_data`.



Для обработки входящей команды, в JavaScript передаются параметры:

- raw\_data - RAW-содержимое запроса. Представляет собой json-закодированную строку. Передается в JavaScript как массив байт.
- body - тело запроса. Передается в JavaScript как массив байт.
- cookies - массив cookie
- headers - массив с заголовками запроса
- method - HTTP-метод
- params - параметры запроса в URL
- url - URL запроса JavaScript-код может проанализировать и изменить эти параметры.

В JavaScript дополнительно передаются внешние функции:

- atob - преобразует строку в base64
- btoa - преобразует base64 в строку

Пример файла настройки обработчика запросов:

```
{
  "requests": {
    "GET|/telemetry/:id/:value": "str = String.fromCharCode.apply(null, raw_data); js =
JSON.parse(str); raw_data='insert into public.telemetry (sensor, value) values (' + js.params.id + ', '
+ js.params.value + ');';"
  }
}
```

## 3.4. Модуль сохранения в MySQL

### 3.3.1. Конфигурируемые параметры

- SETTINGS\_FILE - путь к файлу с настройками запросов. Файл подключается к контейнеру с помощью volume. В данной переменной окружения указывается локальный путь внутри контейнера, к которому был примонтирован volume.
- METRICS\_PORT - порт к подсистеме проверки работоспособности.
- KAFKA\_HOST - имя хоста или IP адрес сервера Kafka
- KAFKA\_TOPIC - топик на сервере Kafka
- KAFKA\_TIMEOUT - таймаут запросов к серверу Kafka
- KAFKA\_GROUP - группа подписчиков сервера Kafka
- DB\_HOST — адрес сервера MySQL
- DB\_USER — пользователь базы данных
- DB\_PASSWORD — пароль пользователя базы данных
- DB\_NAME — имя базы данных
- DB\_CONN\_MAX\_TIME — устанавливает максимальное время перед переиспользованием соединения
- DB\_MAX\_OPEN\_CONNECTIONS — максимальное количество соединений с базой данных
- DB\_MAX\_IDLE\_CONNECTIONS — максимальное количество неиспользуемых соединений с базой данных
- LOG\_LEVEL - уровень логгирования. Поддерживаемые значения:
  - error
  - warn
  - info
  - debug
  - trace

### Пример настройки модуля:

```
outmy:
  build:
    context: ./out-my/
  restart: always
  volumes:
    - ./out-my/config.json:/app/config.json:ro
  environment:
    KAFKA_HOST: host.docker.internal:9092
    KAFKA_TOPIC: test
    KAFKA_GROUP: consumer-group-id-1
    KAFKA_TIMEOUT: 10
    LOG_LEVEL: trace
    SETTINGS_FILE: /app/config.json
    DB_HOST: host.docker.internal
    DB_USER: root
    DB_PASSWORD: bG68r6229J855A7Nm1u5
    DB_NAME: test_db
  extra_hosts:
    - "host.docker.internal:host-gateway"
```

### 3.3.2. Обработка поступающих запросов

Каждый запрос, полученный от модуля приема сообщений, приходит с теми же настройками маршрутизации, которые были описаны выше в разделе 3.2.2.

Принимаемые сообщения могут быть обработаны с помощью JavaScript-кода, который позволит модифицировать входящие параметры и сформировать запрос к базе данных.

Для того, чтобы модуль обработал конкретный поступающий запрос, его следует прописать в файле, подключенном к модулю и указанном в переменной окружения `SETTINGS_FILE`. Этот файл содержит в текстовом формате `json` — объект, который в свою очередь, содержит еще объект:

- `requests` — содержит описание маршрутов и обработчики поступающих данных

Ключами объекта `requests` являются описатели маршрутов. Они должны совпадать с аналогичными маршрутами, указанными у модуля приема сообщений.

Значениями объекта `requests` являются строки, содержащие в себе JavaScript-код.

Для каждой команды должен быть написан JavaScript-код, который должен вернуть SQL-запрос к базе данных в параметре `raw_data`.

Для обработки входящей команды, в JavaScript передаются параметры:

- raw\_data - RAW-содержимое запроса. Представляет собой json-закодированную строку. Передается в JavaScript как массив байт.
- body - тело запроса. Передается в JavaScript как массив байт.
- cookies - массив cookie
- headers - массив с заголовками запроса
- method - HTTP-метод
- params - параметры запроса в URL
- url - URL запроса JavaScript-код может проанализировать и изменить эти параметры.

В JavaScript дополнительно передаются внешние функции:

- atob - преобразует строку в base64
- btoa - преобразует base64 в строку

Пример файла настройки обработчика запросов:

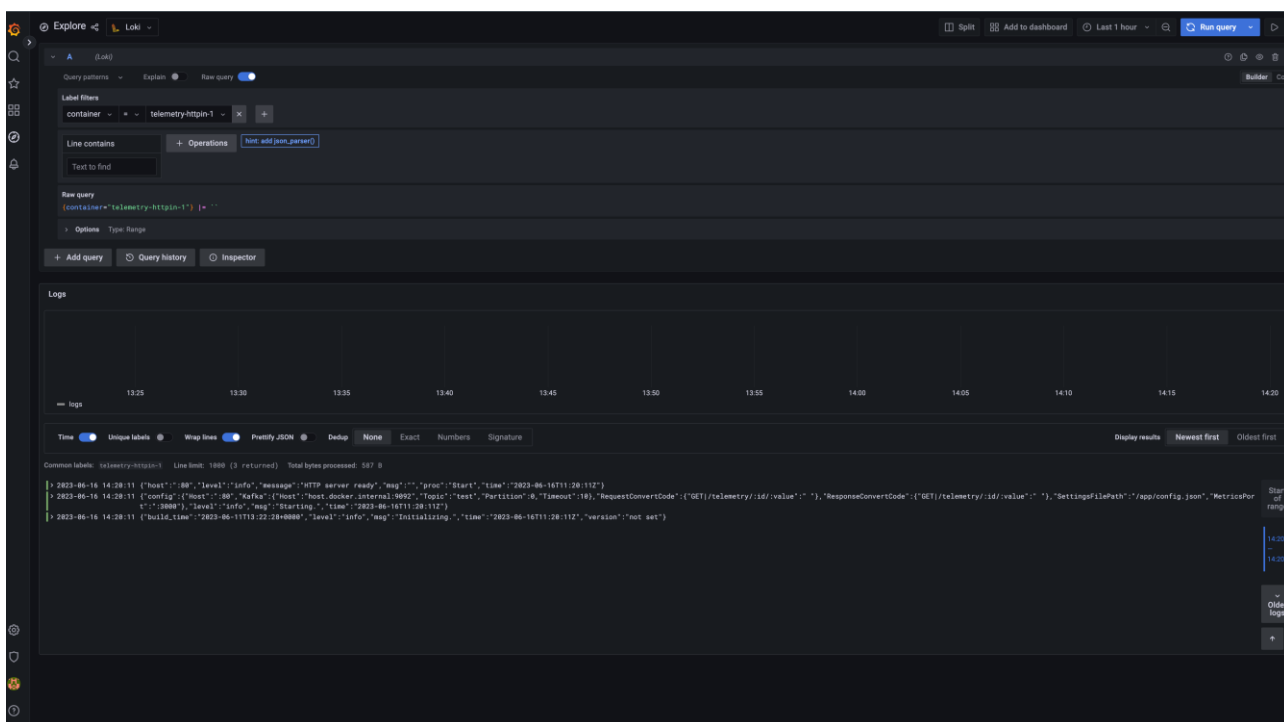
```
{
  "requests": {
    "GET|/telemetry/:id/:value": "str = String.fromCharCode.apply(null, raw_data); js =
JSON.parse(str); raw_data='insert into `telemetry` (`sensor`, `value`) values (' + js.params.id + ', ' +
js.params.value + ');";"
  }
}
```

## 4. Система мониторинга

В качестве системы мониторинга используется Grafana Loki — это набор компонентов для полноценной системы работы с логами. Loki-стек состоит из трёх компонентов: Promtail, Loki, Grafana. Promtail собирает логи, обрабатывает их и отправляет в Loki. Loki их хранит. A Grafana умеет запрашивать данные из Loki и показывать их. Loki можно использовать не только для хранения логов и поиска по ним. Весь стек даёт большие возможности по обработке и анализу поступающих данных

Чтобы открыть интерфейс системы мониторинга, перейдите в браузере на IP Вашего сервера и порт 3000. Если Вы входите туда в первый раз, используйте логин admin и пароль admin. После первого входа система попросит Вас изменить пароль на безопасный.

Интерфейс выглядит так:



Выберите в меню пункт «Explore» - Вы увидите страницу поиска логов.

Сам запрос состоит из двух частей: selector и filter. Selector — это поиск по индексированным метаданным (лейблам), которые присвоены логам, а filter — поисковая строка или регэксп, с помощью которого отфильтровываются записи, определённые селектором.

Выберите в разделе Label filters в ниспадающем списке Label значение container, а в ниспадающем списке value выберите нужный контейнер. Выполните запрос Run query и Вы увидите логи выбранного контейнера.