

**Руководство администратора ПО
«Система автоматизированной доставки
внутрикорпоративных сообщений в персональные чаты
Телеграм»**

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 автоматизированной доставки внутрикорпоративных сообщений в персональные чаты Телеграм» (далее - «Система»)

1.2. Перечень выполняемых функций администратора/оператора

В перечень выполняемых функций администратора Системы входят:

- Установка и настройка Системы
- Реализация планов устранения сбоев и нетиповых нестандартных ситуаций
- Выполнение сбора и предоставление в вышестоящую линию технической поддержки информации для воспроизведения технических проблем и выработки решений по их разрешению
- Реализация рекомендаций по устранению нестандартных ситуаций, полученных с вышестоящей линии поддержки
- Восстановление работоспособности Системы при сбоях в работе функциональных модулей
- Разработка решения по устранению технических проблем в работе функциональных модулей

1.3. Уровень подготовки администратора/оператора

Администратор/оператор (далее по тексту Администратор) Системы должен уметь пользоваться и настраивать среду функционирования контейнеров или систему оркестрации, используемую на предприятии.

Рекомендуемая численность персонала для эксплуатации Системы — 1 штатная единица.

Администраторы Системы должны пройти обязательную общую и специальную подготовку для работы с Системой.

Общая подготовка должна включать в себя получение знаний и навыков работы с Системой в качестве администратора.

Специальная подготовка должна включать в себя получение знаний и навыков в объеме, необходимом для выполнения своих должностных обязанностей

1.4. Перечень документации

В состав документации, с которой необходимо ознакомиться администратору Системы входят:

- описание функциональных характеристик Системы
- описание процессов, обеспечивающих поддержание жизненного цикла программного обеспечения.

2. Установка Системы

В данном разделе будет описана установка Системы на Debian Linux. Предполагается, что были предварительно установлены также Docker, Docker Compose, RabbitMQ, PostgreSQL.

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 RabbitMQ
- Подключение к СУБД PostgreSQL
- 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)
- RabbitMQ (Открытая лицензия Mozilla Public License)
- Grafana Loki 2.6.1 (Открытая лицензия GNU)
- Grafana 9.2.2 (Открытая лицензия GNU)
- PostgreSQL 14 (Открытая лицензия PostgreSQL license)

2.1.3. Языки программирования

При разработке Системы был использован язык программирования GoLang 1.20 (открытая лицензия BSD)

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. Выполните скрипт install.sql, находящийся в папке /home/app, на вашем сервере PostgreSQL. Обратите внимание, что установка и настройка сервера PostgreSQL, а также создание базы данных находятся вне компетенции этого документа и не будут тут описаны.
5. Смените текущую папку на /home/app и выполните команды
sudo chown 10001:10001 ./volumes/loki
sudo chown 472:472 ./volumes/grafana
6. Отредактируйте файл docker-compose.yml, в соответствии с пунктами 3.2 и 3.3 данного документа
7. Создайте и отредактируйте файлы настроек для обоих модулей, в соответствии с пунктами 3.2.2, 3.2.3 и 3.3.2 данного документа
8. Смените текущую папку на /home/app и выполните в ней команду
docker compose -up -d --build
9. 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, СУБД, сервера очередей RabbitMQ, а также возможная настройка использования систем оркестрации, находятся вне компетенции этого документа и не будут тут описаны.

3.2. Модуль приема запросов

3.2.1. Конфигурируемые параметры

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

- HOST - адрес сервиса, который будет слушать модуль. На этот адрес следует пробросить внешний порт или настроить проксирующий сервер для поддержки протокола https.
- METRICS_PORT - порт к подсистеме проверки работоспособности.
- AMQP_HOST — имя хоста или IP сервера RabbitMQ, с указанием порта и учетной записи. Пример смотрите ниже.
- AMQP_EXCHANGE — точка обмена на сервере RabbitMQ
- 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 логирования. Поддерживаемые значения:
 - error
 - warn
 - info
 - debug
 - trace

Пример настройки модуля:

```
httpin:  
build:  
  context: ./http-in/  
restart: always  
ports:  
  - '80:80'  
environment:  
  HOST: :80  
  LOG_LEVEL: trace  
  AMQP_HOST: amqp://rabbit:Q3ij6X4DZnb9uPT@host.docker.internal:5672/  
  AMQP_EXCHANGE: test  
  DB_HOST: host.docker.internal  
  DB_PORT: 5432
```

```
DB_USER: postgres
DB_PASSWORD: LDP8brN7B8ZLzf5Q879QXuJYXCwm9nP
DB_NAME: messages_db
extra_hosts:
- "host.docker.internal:host-gateway"
```

3.3. Модуль маршрутизации

3.3.1. Конфигурируемые параметры

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

- METRICS_PORT - порт к подсистеме проверки работоспособности.
- AMQP_HOST — имя хоста или IP сервера RabbitMQ, с указанием порта и учетной записи. Пример смотрите ниже.
- IN_AMQP_EXCHANGE — точка обмена на сервере RabbitMQ для входящих сообщений
- OUT_AMQP_EXCHANGE — точка обмена на сервере RabbitMQ для исходящих сообщений
- 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 логирования. Поддерживаемые значения:
 - error
 - warn
 - info
 - debug
 - trace

Пример настройки модуля:

```
router:
  build:
    context: ./router/
  restart: always
  environment:
    LOG_LEVEL: trace
    AMQP_HOST: amqp://rabbit:Q3ij6X4DZnb9uPT@host.docker.internal:5672/
    IN_AMQP_EXCHANGE: test
    OUT_AMQP_EXCHANGE: out_test
    DB_HOST: host.docker.internal
    DB_PORT: 5432
    DB_USER: postgres
    DB_PASSWORD: LDP8brN7B8ZLzf5Q879QXuJYXCwm9nP
    DB_NAME: messages_db
  extra_hosts:
    - "host.docker.internal:host-gateway"
```

3.4. Модуль доставки сообщений в Телеграм

3.4.1. Конфигурируемые параметры

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

- TELEGRAM_HASH — токен для доступа к апи Телеграм. Для работы с Телеграм, необходимо создать бота. Для этого специальному боту @botfather отправляется сообщение /newbot. Далее, следуя подсказкам бота, потребуется ввести имя бота, его идентификатор (имя пользователя бота), после чего, бот вернет токен для доступа API Телеграм. Этот токен должен быть указан тут.
- METRICS_PORT - порт к подсистеме проверки работоспособности.
- AMQP_HOST — имя хоста или IP сервера RabbitMQ, с указанием порта и учетной записи. Пример смотрите ниже.
- AMQP_EXCHANGE — точка обмена на сервере RabbitMQ
- 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
- TEMPLATES_PATH — путь к папке с шаблонами сообщений. По умолчанию /files
- LOG_LEVEL - уровень логирования. Поддерживаемые значения:
 - error

- warn
- info
- debug
- trace

Пример настройки модуля:

```
tgout:
  build:
    context: ./telegram-out/
  restart: always
  environment:
    LOG_LEVEL: trace
    AMQP_HOST: amqp://rabbit:Q3ij6X4DZnb9uPT@host.docker.internal:5672/
    IN_AMQP_EXCHANGE: out_test
    OUT_AMQP_EXCHANGE: delivery
    DB_HOST: host.docker.internal
    DB_PORT: 5432
    DB_USER: postgres
    DB_PASSWORD: LDP8brN7B8ZLzf5Q879QXuJYXCwm9nP
    DB_NAME: messages_db
    TELEGRAM_HASH: 6751305935:AAEnKA1yUVQnaqprkS9SFOfuKb2E29GsBIQ
  volumes:
    - ./volumes/files:/files
  extra_hosts:
    - "host.docker.internal:host-gateway"
```

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. А 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:

русском языке. Если соответствующий шаблон сообщения не будет найден — сообщение не будет доставлено. Шаблоны сообщений хранятся в папке, которую использует Модуль доставки сообщений в Телеграм. Конкретное их расположение в файловой системе модуля регулируется переменной окружения `TEMPLATES_PATH`.

Внутри этой папки должны располагаться папки с кодами поддерживаемых языков. Например `ru` или `en`, внутри которых уже располагаются файлы шаблонов.

Шаблон представляет собой текстовый файл с любым именем, внутри которого находится сообщение, которое будет отправлено пользователю. Сообщение может содержать именованные параметры, окруженные двойными круглыми скобками. Например: `{{test}}`. При получении сообщения, Система будет заменять все параметры в шаблоне на пришедшие в сообщении параметры.

5.2.2. Создание типа сообщений

Для создания типа сообщений, надо отправить сообщение `subjects.create`. Например:

```
{
  "jsonrpc": "2.0",
  "method": "subjects.create",
  "id": 1244560000,
  "params": {
    "subject": "testing",
    "params": {
      "telegram_template": "tgtest.tpl"
    }
  }
}
```

В качестве параметров сообщения отправляется объект, содержащий параметры:

- `subject` — тип сообщения или его «тема». Уникальный идентификатор, по которому Система идентифицирует типы сообщений.
- `params` — параметры типа сообщения. Объект, содержащий необходимые для отправки сообщения настройки.
 - `telegram_template` - имя файла шаблона сообщения.

В качестве ответа, Система возвращает уникальный идентификатор созданного типа сообщений. Этот идентификатор, в дальнейшем, будет использоваться для управления этим типом сообщений. Например:

```
{
  "jsonrpc": "2.0",
  "result": "241b5680-4696-4e70-a8b6-14adcd68971d",
  "id": 1244560000
}
```

5.2.3. Чтение типа сообщений

Для чтения используется метод `subjects.read` принимающий идентификатор типа сообщений в качестве параметра. Например:

```
{
  "jsonrpc": "2.0",
  "method": "subjects.read",
  "id": 1244560000,
  "params": "241b5680-4696-4e70-a8b6-14adcd68971d"
}
```

В ответе Система возвращает свойства указанного типа сообщений. Например:

```
{
  "jsonrpc": "2.0",
  "result": {
    "subject": "testing",
    "params": {
      "telegram_template": "tgtest.tpl"
    }
  },
  "id": 1244560000
}
```

5.2.4. Чтение списка типов сообщений в Системе

Метод `subjects.list`, не принимающий параметров, возвращает список всех имеющихся в системе типов сообщений. Например:

```
{
  "jsonrpc": "2.0",
  "result": [
    {
      "id": "fa791266-d945-406c-9be5-a74ff95f7e60",
      "subject": "test",
      "params": {
        "telegram_template": "tgtest.tpl"
      }
    },
    {
      "id": "241b5680-4696-4e70-a8b6-14adcd68971d",
      "subject": "testing",
      "params": {
        "telegram_template": "tgtest.tpl"
      }
    }
  ]
}
```

```
    }  
  ],  
  "id": 1244560000  
}
```

5.2.5. Изменение типа сообщений

Для изменения используется метод `subjects.update`, принимающий параметры:

- `id` — идентификатор типа сообщения
- `subject` — тип сообщения или его «тема». Уникальный идентификатор, по которому Система идентифицирует типы сообщений.
- `params` — параметры типа сообщения. Объект, содержащий необходимые для отправки сообщения настройки.
 - `telegram_template` - имя файла шаблона сообщения.

Например:

```
{  
  "jsonrpc": "2.0",  
  "method": "subjects.update",  
  "id": 1244560000,  
  "params": {  
    "id": "241b5680-4696-4e70-a8b6-14adcd68971d",  
    "subject": "test1",  
    "params": {  
      "telegram_template": "tgtest1.tpl"  
    }  
  }  
}
```

5.2.6. Удаление типа сообщений

Для удаления используется метод `subjects.delete` принимающий идентификатор типа сообщений в качестве параметра. Например:

```
{  
  "jsonrpc": "2.0",  
  "method": "subjects.delete",  
  "id": 1244560000,  
  "params": "241b5680-4696-4e70-a8b6-14adcd68971d"  
}
```

5.3. Управление получателями сообщений

Система рассылает сообщения только, заранее настроенным, получателям. Для того, чтобы Система могла доставить сообщение, получателя необходимо внести его в базу данных Системы и направить ему ссылку для авторизации в Системе.

5.3.1. Авторизация получателя сообщений в Системе

Для работы с Телеграм, необходимо создать бота. Для этого специальному боту @botfather отправляется сообщение /newbot. Далее, следуя подсказкам бота, потребуется ввести имя бота, его идентификатор (имя пользователя бота), после чего, бот вернет токен для доступа API Телеграм. Имя бота используется для авторизации пользователя в Телеграм. Запомните его.

Также, при создании получателя сообщений, Система возвращает его уникальный идентификатор. Он также используется для авторизации в Системе.

Вам необходимо сформировать ссылку вида
`https://telegram.me/YOURBOTNAME?start=unique_code`

Где YOURBOTNAME — это имя, созданного вами бота, а unique_code — идентификатор получателя сообщений. Эту ссылку необходимо передать получателю сообщений. После перехода по этой ссылке, он авторизуется в Системе, а в его настройки будет добавлен параметр «идентификатор чата Телеграм»

5.2.2. Создание получателя сообщений

Для создания получателя сообщений, надо отправить сообщение recipients.create. Например:

```
{
  "jsonrpc": "2.0",
  "method": "recipients.create",
  "id": 1244560000,
  "params": {
    "caption": "test",
    "params": {
      "test": "test"
    }
  }
}
```

В качестве параметров сообщения отправляется объект, содержащий параметры:

- caption — наименование получателя сообщений. Используется только для того, чтобы администратор мог легко отличать получателей сообщений между собой
- params — параметры получателя сообщений. Этот объект может содержать любые настройки как самого получателя сообщений, так и настройки типов или маршрутов сообщений. При отправке сообщения, формируется консолидированный json-объект,

в котором настройки получателя сообщений имеют приоритет над настройками типа сообщений или маршрута. Это позволяет, например, задать персональный шаблон сообщений для конкретного пользователя.

В качестве ответа, Система возвращает уникальный идентификатор созданного получателя сообщений. Этот идентификатор, в дальнейшем, будет использоваться для управления этим получателем сообщений, а также для его авторизации в Системе. Например:

```
{
  "jsonrpc": "2.0",
  "result": "241b5680-4696-4e70-a8b6-14adcd68971d",
  "id": 1244560000
}
```

5.3.3. Чтение свойств получателя сообщений

Для чтения используется метод `recipients.read` принимающий идентификатор получателя сообщений в качестве параметра. Например:

```
{
  "jsonrpc": "2.0",
  "method": "recipients.read",
  "id": 1244560000,
  "params": "241b5680-4696-4e70-a8b6-14adcd68971d"
}
```

В ответе Система возвращает свойства указанного типа сообщений. Например:

```
{
  "jsonrpc": "2.0",
  "result": {
    "caption": "test",
    "params": {
      "test": "test"
    }
  },
  "id": 1244560000
}
```

5.3.4. Чтение списка получателей сообщений в Системе

Метод `recipients.list`, не принимающий параметров, возвращает список всех имеющихся в системе получателей сообщений. Например:

```
{
  "jsonrpc": "2.0",
  "result": [
    {
```

```

    "id": "09d49abe-cc01-494e-b69e-99cb9aab7eb4",
    "caption": "user",
    "params": {
      "lang": "ru",
      "telegram_chat_id": 200575401
    }
  },
  {
    "id": "241b5680-4696-4e70-a8b6-14adcd68971d",
    "caption": "test",
    "params": {
      "test": "test"
    }
  }
],
"id": 1244560000
}

```

5.3.5. Изменение получателя сообщений

Для изменения используется метод `recipients.update`, принимающий параметры:

- `id` — идентификатор получателя сообщения
- `caption` — наименование получателя сообщений. Используется только для того, чтобы администратор мог легко отличать получателей сообщений между собой
- `params` — параметры получателя сообщений.

Например:

```

{
  "jsonrpc": "2.0",
  "method": "recipients.update",
  "id": 1244560000,
  "params": {
    "id": "241b5680-4696-4e70-a8b6-14adcd68971d",
    "caption": "test1",
    "params": {
      "test1": "test1"
    }
  }
}

```

5.3.6. Удаление получателя сообщений

Для удаления используется метод `recipients.delete` принимающий идентификатор получателя сообщений в качестве параметра. Например:

```
{
  "jsonrpc": "2.0",
  "method": "recipients .delete",
  "id": 1244560000,
  "params": "241b5680-4696-4e70-a8b6-14adcd68971d"
}
```

5.4. Управление маршрутами сообщений

Систему можно настроить таким образом, чтобы часть сообщений отправлялась одной группе пользователей, а часть другой. Это соответствие между типом сообщений и пользователем в Системе называется «маршрут».

5.4.1. Создание или модификация маршрута сообщений

Для создания маршрута, надо отправить сообщение `routes.set`. Например:

```
{
  "jsonrpc": "2.0",
  "method": "routes.set",
  "id": 1244560000,
  "params": {
    "subject_id": "241b5680-4696-4e70-a8b6-14adcd68971d",
    "recipient_id": "09d49abe-cc01-494e-b69e-99cb9aab7eb4",
    "params": { }
  }
}
```

В качестве параметров сообщения отправляется объект, содержащий параметры:

- `subject_id` — идентификатор типа сообщений
- `recipient_id` — идентификатор получателя сообщений
- `params` — параметры маршрута сообщений. Этот объект может содержать любые настройки, которыми можно переопределить настройку типа сообщений. Это позволяет, например, задать персональный шаблон сообщений для конкретного пользователя.

5.4.2. Чтение свойств маршрута сообщений

Для чтения используется метод `routes.read` принимающий идентификатор получателя

сообщений в качестве параметра. Например:

```
{
  "jsonrpc": "2.0",
  "method": "routes .read ",
  "id": 1244560000,
  "params": {
    "subject_id": "241b5680-4696-4e70-a8b6-14adcd68971d",
    "recipient_id": "09d49abe-cc01-494e-b69e-99cb9aab7eb4",
  }
}
```

В ответе Система возвращает свойства указанного маршрута сообщений. Например:

```
{
  "jsonrpc": "2.0",
  "result": {},
  "id": 1244560000
}
```

5.4.3. Чтение списка маршрутов сообщений в Системе

Метод `routes.list`, не принимающий параметров, возвращает список всех имеющихся в системе маршрутов сообщений. Например:

```
{
  "jsonrpc": "2.0",
  "result": [
    {
      "subject_id": "fa791266-d945-406c-9be5-a74ff95f7e60",
      "recipient_id": "09d49abe-cc01-494e-b69e-99cb9aab7eb4",
      "params": {}
    }
  ],
  "id": 1244560000
}
```

5.4.4. Удаление маршрута сообщений

Для удаления используется метод `routes.delete` принимающий идентификатор маршрута сообщений в качестве параметра. Например:

```
{
  "jsonrpc": "2.0",
  "method": "routes.delete ",
  "id": 1244560000,
  "params": {
    "subject_id": "241b5680-4696-4e70-a8b6-14adcd68971d",
  }
}
```

```
"recipient_id": "09d49abe-cc01-494e-b69e-99cb9aab7eb4",  
}  
}
```

5.5. Отправка сообщений

После того, как система была настроена, а пользователи авторизованы, можно приступать к отправке сообщений. Для этого надо отправить сообщение `message`. Например:

```
{  
  "jsonrpc": "2.0",  
  "method": "message",  
  "id": 1244560000,  
  "params": {  
    "subject": "test",  
    "message_id": "1",  
    "payload": {  
      "test": "test"  
    }  
  }  
}
```

В качестве параметров сообщения отправляется объект, содержащий параметры:

- `subject` — наименование типа сообщений.
- `message_id` — идентификатор сообщения
- `payload` — параметры сообщения. Внутри этого объекта должны содержаться параметры, которые будут использованы при формировании текста сообщения на основании шаблона сообщения.