

ИНФРАСТРУКТУРА ЭЛЕКТРОННОГО ПРАВИТЕЛЬСТВА

**ВЫПОЛНЕНИЕ РАБОТ ПО РАЗВИТИЮ ТИПОВОГО ТИРАЖИРУЕМОГО  
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ВИТРИН ДАННЫХ**

Витрина данных НСУД

Руководство по установке ПО «Витрина данных НСУД»

Версия 1.13.0

Листов 89

Москва, 2024

## СОДЕРЖАНИЕ

<b>1 Общие сведения о программе .....</b>	<b>6</b>
1.1 Назначение программы.....	6
1.2 Возможности программы .....	6
1.3 Компоненты системы.....	7
<b>2 Подготовка к установке .....</b>	<b>9</b>
2.1 Предварительные действия .....	9
2.1.1 Установка операционной системы .....	9
2.1.2 Настройка межсетевого экрана .....	10
2.1.3 Отключение SELinux (только для CentOS) .....	10
2.1.4 Выбор часового пояса .....	11
2.1.5 Установка сервиса синхронизации времени .....	11
2.1.6 Настройка имен хостов (FQDN) на серверах .....	11
2.1.7 Установка Java SE Development Kit 17.0.7 .....	11
2.1.8 Установка компонента сбора данных запросов и ответов Витрины данных.....	11
2.1.8.1 Процесс установки .....	12
2.1.9 Настройка сервиса журналирования .....	15
2.1.10 Настройка подсистемы мониторинга.....	19
2.1.10.1 Настройка конфигурации для OpenShift .....	21
2.1.10.2 Grafana: графики мониторинга.....	22
<b>3 Настройка на состав технических средств .....</b>	<b>23</b>
3.1 Общие требования .....	23
3.2 Требования к серверам .....	24
3.2.1 Требования к серверу ProStore .....	24
3.2.2 Требования к серверу Arenadata Cluster Manager (ADCM) .....	24
3.2.3 Требования к кластеру серверов Arenadata Streaming (ADS).....	24
3.2.4 Требования к серверу «СМЭВ3-адаптер» .....	25
3.2.5 Требования к серверу «Агент ПОДД» .....	25
3.2.6 Требования к серверу «ETL» .....	26
3.2.7 Требования к серверу «REST-адаптер» .....	27
<b>4 Установка программы .....</b>	<b>28</b>
4.1 Порядок установки.....	28
4.2 Установка ПО ProStore.....	29
4.2.2 Запуск службы Prostore-query-execution-core.....	29
4.3 Установка СМЭВ QL Сервера.....	31

4.4 Установка СМЭВ3-адаптера .....	31
4.5 Установка ПОДД-адаптера - Модуль исполнения запросов .....	33
4.6 Установка ПОДД-адаптер – Модуль MPPR .....	34
4.7 Установка ПОДД-адаптер-Модуль MPPW .....	34
4.8 Установка ПОДД-адаптер – Модуль импорта данных табличных параметров .....	35
4.9 Установка ПОДД-адаптер – Модуль группировки данных табличных параметров .....	36
4.10 Установка ПОДД-адаптер – Wrapper .....	37
4.11 Установка модуля группировки чанков репликации .....	38
4.12 Установка DATA-uploader – Модуль исполнения асинхронных заданий .....	38
4.13 Установка REST-uploader – Модуль асинхронной загрузки данных из сторонних источников .....	39
4.14 Установка ПОДД-адаптер – Модуль подписки .....	39
4.15 Установка BLOB-адаптер .....	40
4.16 Установка Сервиса формирования документов .....	41
4.17 Загрузка и удаление данных через ETL .....	42
4.17.1 Общее описание .....	42
4.17.1.1 Общие положения .....	42
4.17.2 Особенности реализации .....	42
4.17.2.1 Основные требования к исходным файлам .....	42
4.17.2.2 Особенности реализации ETL .....	45
4.17.2.3 Получение токена Proxy API .....	45
4.17.3 Загрузка / удаление данных .....	48
4.17.3.1 Загрузка/ удаление данных .....	48
4.17.4 Проверка статусной информации по загрузке .....	53
4.17.4.1 Проверка статусной информации по загрузке / удалению данных (Endpoint – status) .....	53
4.17.5 Работа с вложениями через S3 .....	54
4.17.5.1 Работа с вложениями через S3 .....	54
4.17.6 Маппинг данных .....	56
4.17.6.1 Маппинг данных (Endpoint – generateMapping) .....	56
4.17.7 Валидация данных .....	57
4.17.7.1 Валидация данных .....	57
4.17.8 Установка Apache Airflow .....	58
4.17.9 Установка Apache Spark .....	60
4.17.10 Установка Apache Hadoop .....	61
4.17.11 Установка Tarantool(Vinyl) .....	62
4.18 Установка CSV-Uploader .....	63
4.19 Установка REST-адаптера .....	64
4.19.1 Установка docker-образов .....	64

4.19.2 Подготовка конфигурации .....	64
4.19.3 Процесс установки .....	65
4.20 Установка Counter-provider.....	65
4.21 Установка коннектора Kafka-Postgres .....	66
4.22 Установка Arenadata Cluster Manager (ADCM) .....	67
4.23 Установка Arenadata Streaming (ADS).....	69
<b>5 Проверка программы .....</b>	<b>74</b>
5.1 Проверка Arenadata Cluster Manager (ADCM).....	74
5.2 Проверка Arenadata Streaming (ADS) .....	74
5.2.1 Проверка сервиса Zookeeper.....	74
5.2.2 Проверка сервиса Apache Kafka.....	74
5.3 Проверка ProStore .....	74
5.4 Проверка СМЭВ QL Сервера.....	75
5.4.1 Проверки и валидации.....	75
5.5 Проверка СМЭВ3-адаптера.....	76
5.6 Проверка ПОДД-адаптера - Модуль исполнения запросов.....	76
5.7 Проверка ПОДД-адаптер – Модуль MPPR .....	76
5.8 Проверка ПОДД-адаптер-Модуль MPPW .....	77
5.9 Проверка ПОДД-адаптер – Модуль импорта данных табличных параметров.....	77
5.10 Проверка ПОДД-адаптер – Модуль группировки данных табличных параметров .....	77
5.11 Проверка ПОДД-адаптер – ПОДД-адаптер – Wrapper.....	78
5.12 Проверка модуля группировки чанков репликации.....	78
5.13 Проверка DATA-uploader – Модуль исполнения асинхронных заданий .....	78
5.14 Проверка REST-uploader – Модуль асинхронной загрузки данных из сторонних источников .....	79
5.15 Проверка ПОДД-адаптер – Модуль подписки.....	79
5.16 Проверка BLOB-адаптер .....	79
5.17 Проверка Сервиса формирования документов.....	80
5.18 Проверка ETL .....	80
5.18.1 Проверка Apache Airflow .....	80
5.18.2 Проверка Apache Spark.....	81
5.18.3 Проверка Apache Hadoop .....	81
5.18.4 Проверка Tarantool(Vinyl) .....	82
5.19 Проверка REST-адаптер .....	82
5.20 Проверка Counter-provider - Сервиса генерации уникального номера .....	82
<b>6 Термины и определения .....</b>	<b>83</b>

## **Аннотация**

Настоящий документ является руководством по установке модернизированного программного обеспечения «Витрина данных НСУД» (далее – программа).

Перед началом работы необходимо ознакомиться с документом 83219291.62.01.11.A003.РСП.01.1 «Руководство системного программиста» на ПО «Витрина данных НСУД», разработанным в рамках выполнения государственного контракта № 0173100007520000024\_144316 от 13 ноября 2020 года, на выполнение работ по модернизации федеральной государственной информационной системы «Единая информационная платформа Национальной системы управления данными» в части выделения программного обеспечения компонента «Витрина данных» и его доработки, необходимой для подготовки к публикации на условиях открытой лицензии.

В разделе «Общие сведения» указаны назначение и возможности Программы.

В разделе «Подготовка к установке» описаны предварительные действия на аппаратных и программных средствах перед установкой Программы.

В разделе «Настройка на состав технических средств» приведены требования к серверам программы, перечень сетевых портов, используемых аппаратными средствами.

В разделе «Установка программы» описаны порядок установки программных компонентов Программы и действия по установке Программы.

В разделе «Проверка программы» описаны действия по проверке корректной работы Программы.

Оформление программного документа «Руководство по установке» произведено по требованиям ЕСПД (ГОСТ 19.101-77, ГОСТ 19.103-77, ГОСТ 19.104-78, ГОСТ 19.105-78, ГОСТ 19.106-78, ГОСТ 19.503-79, ГОСТ 19.604-78).

# 1 ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

## 1.1 Назначение программы

Национальная система управления данными (далее – НСУД) представляет собой систему, состоящую из взаимосвязанных элементов информационно-технологического, организационного, методологического, кадрового и нормативно-правового характера и обеспечивающую достижение целей и выполнение задач, обозначенных в Концепции Национальной системы управления данными, утвержденной распоряжением Правительства Российской Федерации от 3 июня 2019 года № 1189-р.

НСУД предназначена для управления информацией, содержащейся в информационных системах органов и организаций государственного сектора, а также в информационных ресурсах, созданных в целях реализации полномочий органов и организаций государственного сектора (далее – государственные данные) и для осуществления информационного обмена между Поставщиками и Получателями данных, присоединившимися к НСУД (далее – Участники НСУД).

Управление процессами информационного обмена между Участниками НСУД осуществляется средствами федеральной государственной информационной системы «Единая информационная платформа Национальной системы управления данными» (далее – ФГИС «ЕИП НСУД»).

Для передачи данных между Участниками НСУД используется среда взаимодействия НСУД, состоящая из Системы межведомственного электронного взаимодействия 3.0 (далее – СМЭВ) и (или) подсистемы обеспечения доступа к данным СМЭВ (далее – ПОДД СМЭВ) (СМЭВ 4.0), обеспечивающих транспорт и процессинг данных, а также агентов ПОДД СМЭВ, устанавливаемых на стороне Участников НСУД.

Для формирования и (или) для получения данных с использованием среды взаимодействия НСУД необходим комплекс программных и технических средств в составе информационно-телекоммуникационной инфраструктуры участника НСУД, описываемое в данном документе «Витрина данных НСУД», но возможно и применение «Витрина данных НСУД»). Данный документ описывает применение именно ПО среды взаимодействия НСУД.

Программа «Витрина данных НСУД» является частью НСУД и предназначена для загрузки публикуемых данных в отдельную БД на стороне Поставщика данных. Программа представляет собой типовое программное обеспечение, устанавливаемое на стороне поставщиков/потребителей данных.

## 1.2 Возможности программы

Программа обеспечивает выполнение следующих задач:

- описание логической модели данных;
- настройка программы и структуры таблиц в ее БД для хранения публикуемых данных;
- загрузка и хранение публикуемых данных в БД программы;
- извлечение данных из внешних систем (внешних ИС по отношению к Витрине данных НСУД);

- выполнение запросов в соответствии с протоколом ПОДД через механизмы ПОДД СМЭВ:
- поддержка протокола коммуникации агента [ПОДД](#).
- предоставление публикуемых данных (в т. ч. BLOB-объектов и/или с использованием табличных параметров);
- генерацию формируемых документов на основании публикуемых данных;
- репликацию публикуемых данных (в качестве витрины-источника);
- получение реплицируемых данных (в качестве витрины-получателя).
- обмен в соответствии с протоколом СМЭВ3:
- подключение к СМЭВ3 как информационной системы участника взаимодействия;
- обработку запросов на предоставление публикуемых данных (видов сведений), в т. ч. BLOB-объектов;
- инициативная рассылка оповещений об обновлении публикуемых данных.
- обработка запросов с использованием стандарта [JDBC](#);
- публикация конечных точек [API](#) для обработки запросов с использованием спецификации [OpenAPI](#) версии 3;
- предоставление публикуемых данных информационным системам с использованием интерфейса JDBC и/или REST-запросов;
- восстановление данных в непротиворечивое состояние после сбоев;
- поддержка языка [SQL](#);
- журналирование событий функциональных блоков;
- мониторинг информации о работоспособности экземпляра Программы.

### 1.3 Компоненты системы

Перечень состава компонентов программы версии 1.13.0 приведен в таблице ниже (см. [Таблица 1.1](#))

Таблица 1.1 Состав компонентов в дистрибутиве программы

Наименование компонента	Версия	Техническое наименование
ПОДД Агент	3.11.0	ПОДД-Агент:3.11.0
СМЭВ3-адаптер	1.13.0	smev3-adapter:1.13.0
printable-form-service	1.13.0	printable-form-service:1.13.0
rest-adapter	1.13.0	rest-adapter:1.13.0
rest-uploader	1.13.0	rest-uploader:1.13.0
data-uploader	1.13.0	data-uploader:1.13.0
blob-adapter	1.13.0	blob-adapter:1.13.0
podd-adapter-query	1.13.0	podd-adapter-query:1.13.0
podd-adapter-replicator	1.13.0	podd-adapter-replicator:1.13.0

Наименование компонента	Версия	Техническое наименование
podd-adapter-group-repl	1.13.0	podd-adapter-group-repl:1.13.0
podd-adapter-mppr	1.13.0	podd-adapter-mppr:1.13.0
podd-adapter-mppw	1.13.0	podd-adapter-mppw:1.13.0
podd-adapter-group-tp	1.13.0	podd-adapter-group-tp:1.13.0
podd-adapter-import-tp	1.13.0	podd-adapter-import-tp:1.13.0
podd-avro-defragmentator	1.13.0	podd-avro-defragmentator:1.13.0
smevql-server	1.13.0	smevql-server:1.13.0
csv-uploader	1.13.0	csv-uploader:1.13.0
counter-provider	1.13.0	counter-provider:1.13.0
backup-manager	1.13.0	backup-manager:1.13.0
query-execution	6.8.1	query-execution:6.8.1
kafka	2.13	kafka:2.13
zookeeper	3.5.7	zookeeper:3.5.7
redis	7.0.11	redis:7.0.11
fdw	0.10.2	fdw:0.10.2
pxf	1.0	pxf:1.0



## 2 ПОДГОТОВКА К УСТАНОВКЕ

### 2.1 Предварительные действия

#### Примечание:

Установка Программы производится в закрытом контуре (без необходимости доступа к сети Интернет).

Перед установкой Программы необходимо выполнить следующие предварительные действия:

1. Установить на серверы одну из поддерживаемых операционных систем (см. раздел [Установка операционной системы](#)).
2. Проверить настройки Firewall и отключить при необходимости (см. раздел [Настройка межсетевого экрана](#)).
3. Выключить SELinux (см. раздел [Отключение SELinux \(только для CentOS\)](#)).
4. Указать соответствующий местоположению сервера часовой пояс (см. раздел [Выбор часового пояса](#)).
5. Проверить, что на всех серверах установлен сервис синхронизации времени (см. раздел [Установка сервиса синхронизации времени](#)).
6. Проверить, что имена хостов (FQDN) серверов могут получать IP по имени со всех машин (см. раздел [Настройка имен хостов \(FQDN\) на серверах](#));
7. Установить/обновить Java SE Development Kit 17.0.7 (см. раздел [Установка Java SE Development Kit 17.0.7](#)).

Дополнительно можно установить:

- компонент сбора данных запросов и ответов Витрины (см. раздел [Установка компонента сбора данных запросов и ответов Витрины данных](#));
- сервис журналирования (см. раздел [Настройка сервиса журналирования](#));
- подсистему мониторинга (см. раздел [Настройка подсистемы мониторинга](#)).

#### 2.1.1 Установка операционной системы

Программа может работать на одной из операционных систем:

- Centos 7.9;
- Astra Linux Special Edition 1.7 (уровень защищенности «Воронеж»);
- Alt 8 SP Server;
- РЕД ОС, версии 7.2.

Подробная инструкция по установке операционной системы Centos 7.9 приведена на официальном сайте разработчика: <https://docs.centos.org/en-US/centos/install-guide/>.

Подробная инструкция по установке операционной системы Astra Linux Special Edition

1.7 приведена на официальном сайте разработчика: <https://astralinux.ru/products/astra-linux-special-edition/documents-astra-se/>

Подробная инструкция по установке операционной системы Alt 8 SP Server приведена на официальном сайте разработчика: <https://www.basealt.ru/alt-8-sp-sertifikat-fstehk/docs>

Подробная инструкция по установке операционной системы РЕД ОС, версии 7.2 приведена в документе «Руководство администратора по РЕД ОС 7.2»

### 2.1.2 Настройка межсетевого экрана

Для корректной установки потребуется отключить службу **Firewalld** операционной системы CentOS.

Что просмотреть текущий статус работы приложения используйте команду **firewall-cmd**:

```
sudo firewall-cmd --state
```

В случае, если служба **Firewalld** запущена, команда выше выведет следующее сообщение:

```
running
```

Вы можете временно остановить службу **Firewalld** для этого выполните следующую команду:

```
sudo systemctl stop firewalld
```

Следует учитывать, что данная команда только временно отключит службу, при последующей перезагрузке служба **Firewalld** снова будет запущена. Чтобы полностью отключить службу выполните следующую команду:

Остановите службу:

```
sudo systemctl stop firewalld
```

Отключите автоматический запуск службы **Firewalld** при загрузке операционной системы:

```
sudo systemctl disable firewalld
```

После отключения проверьте, что статус службы изменился на **not running**, для этого выполните команду

```
sudo firewall-cmd --state  
not running
```

### 2.1.3 Отключение SELinux (только для CentOS)

Для корректной установки CentOS необходимо отключить SELinux, для этого выполните следующие действия:

1. Проверьте параметры запуска **SELinux** при загрузке системы. Для этого выполните следующую команду:

```
cat /etc/selinux/config
```

2. Если параметр **SELINUX** имеет значение **enforcing**, отключите запуск **SELinux** при загрузке системы. Для этого следует в файле **/etc/selinux/config** указать значение **SELINUX=disabled** и перезагрузите сервер. **SELinux** будет отключен.

Открыть и отредактировать файл **/etc/selinux/config** можно с помощью редактора **vi**,

для этого выполните команду:

```
sudo vi /etc/selinux/config
```

3. Проверьте, что служба отключена. Для этого выполните команду:

```
sestatus
```

В ответ вы должны получить:

```
SELinux status: disabled
```

#### 2.1.4 Выбор часового пояса

Проверьте, что установлен нужный часовой пояс. В нашем случае, на команду `timedatectl`, должна выводиться строка `Time zone: Europe/Moscow (MSK, +0300)`.

Пример команды:

```
timedatectl
```

Пример ответа:

```
Local time: Mon 2021-12-20 12:06:39 MSK
Universal time: Mon 2021-12-20 09:06:39 UTC
RTC time: Mon 2021-12-20 09:06:49
Time zone: Europe/Moscow (MSK, +0300)
NTP enabled: n/a
NTP synchronized: no
RTC in local TZ: no
DST active: n/a
```

Если результат отличается, укажите соответствующий местоположению сервера часовой пояс.

Пример команды для московского часового пояса:

```
sudo timedatectl set-timezone Europe/Moscow
```

#### 2.1.5 Установка сервиса синхронизации времени

Для корректной установки программы необходимо убедиться, что на всех серверах установлен сервис синхронизации времени.

#### 2.1.6 Настройка имен хостов (FQDN) на серверах

Для корректной установки программы необходимо проверить, что имена хостов (FQDN) серверов могут взаимно получать IP по имени со всех машин. Имена хостов меняются согласно документации установленной ОС.

#### 2.1.7 Установка Java SE Development Kit 17.0.7

Установка Java SE Development Kit 17.0.7 осуществляется согласно официальной документации: <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

#### 2.1.8 Установка компонента сбора данных запросов и ответов Витрины данных

Компонент сбора данных запросов и ответов Витрины данных реализован с целью проведения бизнес-мониторинга ИЭП процессов обработки запросов типовым ПО витрины данных, как в целом, так и в части функционирования отдельных витрин для последующей передачи данных в СЦЛ.

### 2.1.8.1 Процесс установки

Общий процесс установки состоит из следующих действий:

1. Настройка логирования приложений
2. Установка и настройка Vector.
3. Установка и настройка HaProxy.
4. Установка и настройка fluentbit.
5. Установка ClickHouse.

#### 2.1.8.1.1 Настройка логирования приложений

На стороне приложений **ПОДД-адаптер**, **Модуль MPPR**, **BLOB-адаптер** и сервиса формирования документов необходимо настроить формирование логов в формате JSON. Для этого необходимо в файле `logback.xml` включить `net.logstash.logback.encoder.LogstashEncoder`.

Пример `logback.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>logs/application.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <!-- daily rollover -->
      <fileNamePattern>logs/application.%d{yyyy-MM-dd}.log</fileNamePattern>
      <!-- keep 30 days' worth of history capped at 3GB total size -->
      <maxHistory>30</maxHistory>
      <totalSizeCap>3GB</totalSizeCap>
    </rollingPolicy>
    <encoder class="net.logstash.logback.encoder.LogstashEncoder" />
  </appender>

  <root level="INFO">
    <appender-ref ref="FILE" />
  </root>
</configuration>
```

[Подробная информация об encoder](#)

#### 2.1.8.1.2 Установка и настройка Vector

Установка производится по [официальной документации Vector](#)

Настройка Vector

Пример настройки source:

```
json_source:
  type: fluent
  address: 0.0.0.0:24226
```

Пример фильтрации сообщений, имеющих флаг `scl`:

```
scl_tags_filter:
  type: filter
  inputs:
    - json_source
  condition:
    type: "vrl"
    source: |-
      exists(.tags) && includes(array!(.tags), "TYPE_SCL")
```

Пример парсинга scl-сообщений:

```
scl_message_remap:
  type: remap
  inputs:
  - scl_tags_filter
  source: |-
    . = parse_json!(.message)
```

Пример отправки scl-сообщений в **Kafka**:

```
podd_agent_sink:
  type: kafka
  inputs:
  - scl_message_remap
  bootstrap_servers: kafka:9092
  topic: "<префикс>.scl.signal"
  acknowledgements: true
  compression: "gzip"
  encoding:
    codec: json
  healthcheck: true
```

### 2.1.8.1.3 Установка и настройка HaProxy

Установка производится по [официальной документации HaProxy](#)

Для настройки HaProxy в секции **backend** нужно перечислить список установленных инстансов **Vector**.

Пример файла **haproxy.cfg**:

```
global
  log 127.0.0.1 local2

  chroot /var/lib/haproxy
  pidfile /var/run/haproxy.pid
  maxconn 4000
  user haproxy
  group haproxy
  daemon

  stats socket /var/lib/haproxy/stats

defaults
  mode tcp
  log global
  retries 3

  maxconn 3000

listen stats
  bind          0.0.0.0:1936
  mode          http
  stats         enable
  stats         uri /

frontend services
  bind 0.0.0.0:24226
  default_backend services
  mode tcp

backend services
  balance roundrobin
```

```
mode tcp
server vector01 vector-01:24226
server vector02 vector-02:24226
```

#### 2.1.8.1.4 Установка и настройка FluentBit

Установка производится по [официальной документации FluentBit](#).

Далее необходимо настроить fluentbit на чтение файлов с логами приложений.

Пример файла конфигурации **fluent-bit.conf**:

```
[SERVICE]
  flush          5
  daemon         off
  log_level      info
  parsers_file   parsers.conf
[INPUT]
  name tail
  path <путь до лог файла приложения>
  tag *
  parser json
[OUTPUT]
  name forward
  match *
  host haproxy
  port 24226
```

Пример файла **parsers.conf**:

```
[PARSER]
  Name      json
  Format    json
```

На этом настройка fluentbit завершена.

#### 2.1.8.1.5 Включение / выключение отправки сообщений в СЦЛ

Отправка логов в СЦЛ осуществляется автоматически после корректной настройки компонента.

Для выключения отправки логов можно закомментировать блок **podd\_agent\_sink** отправки сообщений в kafka в настройках Vector.

#### 2.1.8.1.6 Установка и настройка ClickHouse

Установка производится по [официальной документации ClickHouse](#)

Пример задания конфигурационных настроек:

```
clickhouse_default_config:
clickhouse:
  logger:
    level: trace
    log: /var/log/clickhouse-server/clickhouse-server.log
    errorlog: /var/log/clickhouse-server/clickhouse-server.err.log
    size: 1000M
    count: 10
  http_port: 8123
  tcp_port: 9000
  listen_host: 0.0.0.0
  max_connections: 4096
  keep_alive_timeout: 3
  user_directories:
    users_xml:
      path: users.xml
    local_directory:
      path: "{{ clickhouse_root_data_folder }}/access/"
  path: "{{ clickhouse_root_data_folder | add_slash }}"
```

### 2.1.9 Настройка сервиса журналирования

Сервис журналирования позволяет работать с логами прикладных модулей запущенных в средах WildFly и Kubernetes/OpenShift.

Настройка сервиса журналирования должна быть применена к каждому модулю.

Ниже указаны шаги, позволяющие обеспечить интеграцию сервиса с системой журналирования Платформы ГосТех.

1. Добавить зависимости в проект

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.3.4.RELEASE</version>
  <relativePath/> <!-- Lookup parent from repository -->
</parent>
<modelVersion>4.0.0</modelVersion>

<artifactId>logger-test</artifactId>

<properties>
  <java.version>11</java.version>
  <spring-cloud.version>Hoxton.SR5</spring-cloud.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
  </dependency>

  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
```

```

</dependency>

<dependency>
  <groupId>ch.qos.logback.contrib</groupId>
  <artifactId>logback-json-classic</artifactId>
  <version>0.1.5</version>
</dependency>

<dependency>
  <groupId>ch.qos.logback.contrib</groupId>
  <artifactId>logback-jackson</artifactId>
  <version>0.1.5</version>
</dependency>
<dependency>
  <groupId>net.logstash.logback</groupId>
  <artifactId>logstash-logback-encoder</artifactId>
  <version>6.3</version>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.12</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-boot-starter</artifactId>
  <version>3.0.0</version>
</dependency>
<dependency>
  <groupId>org.codehaus.janino</groupId>
  <artifactId>janino</artifactId>
  <version>3.0.6</version>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
  <version>2.5.9.RELEASE</version>
</dependency>
</dependencies>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

```

2. Подключить Fluentbit к приложению как отдельный контейнер в OpenShift



```
containers:
- name: fluent-bit
  image: fluent/fluent-bit:latest
  volumeMounts: #перенос конфигураций, и лог файла из проекта в контейнер
  - name: logger
    mountPath: /fluent-bit/logger/
  - name: fluent-bit-config
    mountPath: /fluent-bit/etc/
  terminationMessagePolicy: File
  envFrom:
  - configMapRef:
    name: logger-fluent-bit-config-env
```

3. Создать файла **logback.xml** для логирования приложения ( подробнее см. [документацию](#))

#### Внимание:

Обязательно в appender FILE\_FLUENT прописать путь до конфигураций fluent-bit, иначе в контейнер логи не подтянутся. Например, **<file>name-project/docker/fluentbit/conf/log.log</file>**.

```
<configuration debug="true">
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <layout class="ch.qos.logback.classic.PatternLayout">
      <pattern>
        <Pattern>
          %d{yyyy-MM-dd HH:mm:ss}%-5level %logger{36} - %msg%n
        </Pattern>
      </pattern>
    </layout>
  </appender>

  <appender name="FILE_FLUENT" class="ch.qos.logback.core.FileAppender">
    <file>docker/fluentbit/conf/log.log</file>
    <append>false</append>
    <layout class="ch.qos.logback.classic.PatternLayout">
      <pattern>
        <Pattern>
          x-b3-traceid=%X{X-B3-TraceId:-} x-b3-spanid=%X{X-B3-SpanId:-} x-b3-
          parentsSpanid=%X{X-B3-ParentSpanId:-} x-b3-sampled=%X{X-B3-Sampled:-} x-b3-flags=%X{X-
          B3-Flags:-} caller_file_name=%file serverEventDatetime="%d" logLevel=%level
          threadName=%thread message="%replace(%replace(%m){'¥n','¥u2028'}){'¥','¥'}"
          exception="%replace(%replace(%ex){'¥n','¥u2028'}){'¥n','¥u2028'}%nopex" callerLine=%L
          callerMethod="%replace(%caller){'¥n','¥u2028'}" loggerName="%10.10logger"
          callerClass=%logger{40} levelStr="%level" levelInt="%level" mdc= ¥n
        </Pattern>
      </pattern>
    </layout>
  </appender>

  <root level="debug" additivity="false">
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="FILE_FLUENT"/>
  </root>
</configuration>
```

4. Добавить описание конфигурации для Fluent-bit (подробнее см [документацию](#) )

```
[SERVICE]
  Flush      1
  Log_Level  info
```

```

    Daemon      off
    Parsers_File /fluent-bit/etc/parsers.conf
[INPUT]
    Name        tail
    Path         /fluent-bit/logger/log.log
    Tag         kafka-efs
    Buffer_Chunk_Size 400k
    Buffer_Max_Size 6MB
    Mem_Buf_Limit 6MB
    Parser       logfmt
    Refresh_Interval 20

[FILTER]
    Name record_modifier
    Match      kafka-efs
    Record     subsystem ${SUBSYSTEM}
    Record     distribVersion ${DISTRIBVERSION}
    Record     deploymentUnit ${DEPLOYMENTUNIT}
    Record     hostName ${HOSTNAME}
    Record     ipAddress ${IPADDRESS}

[FILTER]
    Name        modify
    Match      kafka-efs
    Hard_copy   callerClass className

[FILTER]
    Name        record_modifier
    Match      kafka-efs
    Whitelist_key serverEventDatetime
    Whitelist_key subsystem
    Whitelist_key distribVersion
    Whitelist_key deploymentUnit
    Whitelist_key hostName
    Whitelist_key ipAddress
    Whitelist_key logLevel
    Whitelist_key className
    Whitelist_key threadName
    Whitelist_key message
    Whitelist_key x-b3-traceid
    Whitelist_key x-b3-spanid
[FILTER]
    Name        lua
    Match      kafka-efs
    script     convert_date.lua
    call       convert_date_efs

[OUTPUT]
    Name        stdout
    Match      kafka-efs
    Format       json
    json_date_key time

[OUTPUT]
    Name        http
    Match      kafka-efs
    Host        logstash-service-gt-tatarstan-test-efs.apps.ocp-public.sbercloud.ru
    Port        80
    Format       json
    json_date_key time

```

Для правильной работы необходимо подключить **parsers.conf**.

[PARSER]		
Name		logfmt
Format		logfmt

## 5. Добавить форматирование даты

```
function convert_date_efs(tag, timestamp, record)
  local pattern = "(%d+)-(%d+)-(%d+) (%d+):(%d+):(%d+),(%d+)"
  dt_str = record["serverEventDatetime"]
  local year, month, day, hour, minute, seconds, milliseconds = dt_str:match(pattern)
  ts = os.time{year = year, month = month, day = day, hour = hour, min = minute, sec =
seconds }
  ts = (ts * 1000) + milliseconds
  record["serverEventDatetime"] = ts
  return 2, timestamp, record
end

function convert_date_pprb(tag, timestamp, record)
  local pattern = "(%d+)-(%d+)-(%d+) (%d+):(%d+):(%d+),(%d+)"
  dt_str = record["serverEventDatetime"]
  local year, month, day, hour, minute, seconds, milliseconds = dt_str:match(pattern)
  ts = os.time{year = year, month = month, day = day, hour = hour, min = minute, sec =
seconds }
  ts = (ts * 1000) + milliseconds
  record["timestamp"] = ts
  return 2, timestamp, record
end

function convert_date_logstash(tag, timestamp, record)
  local pattern = "(%d+)-(%d+)-(%d+) (%d+):(%d+):(%d+),(%d+)"
  dt_str = record["serverEventDatetime"]
  local year, month, day, hour, minute, seconds, milliseconds = dt_str:match(pattern)
  ts = os.time{year = year, month = month, day = day, hour = hour, min = minute, sec =
seconds }
  ts = (ts * 1000) + milliseconds
  record["time"] = ts
  return 2, timestamp, record
end
```

## 6. Добавить параметры модуля

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: logger-fluent-bit-config-env
data:
  MODULEID: 1.0.0
  MODULEVERSION: 1.0.0
  NODEID: 12345
  HOSTADDRESS: 0.0.0.0
  SUBSYSTEM: LOGGER-TEST
  DISTRIBVERSION: 1.0.0
  DEPLOYMENTUNIT: TEST-UNIT
  IPADDRESS: 0.0.0.1
```

### 2.1.10 Настройка подсистемы мониторинга

Подсистема мониторинга предназначена для регистрации отладочной информации прикладных модулей в едином журнале.

Настройка подсистемы мониторинга должна быть применена к каждому модулю.

Ниже указана последовательность действий для выставления метрик в формате Prometheus из прикладного приложения, написанного на Spring Boot.

1. Добавить зависимостей на Spring Boot Actuator и Micrometer в maven репозиторий.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-core</artifactId>
</dependency>
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

2. Указать порт **Actuator**, фильтр включенных конечных точек и системные теги, которые автоматически будут добавлены к прикладным метрикам (файл application.yml)

```
server:
port: 8080

management:
endpoint:
  health.show-details: always
endpoints:
web:
exposure:
  include: '*'
metrics:
tags:
application: ${spring.application.name}
namespace: ${POD_NAMESPACE:local}
pod: ${POD_NAME:local}
node_name: ${NODE_NAME:local}
```

3. Создать прикладные метрики в проекте (с помощью micrometer)

```
@RestController
public class CounterController {

    private static final String COUNTER_WITH_TAG_NAME = "simple.counterWithTags";
    private static final String COUNTER_WITH_TAG_DESCRIPTION = "Just a simple counter
with tags";
    private static final String TAG_NAME_1 = "terbank";
    private static final String TAG_NAME_2 = "vsp";

    private final MeterRegistry meterRegistry;

    private Counter simpleCounter;
    private Counter simpleCounterWithTags;
    private Counter simpleCounterWithTags2;
```

```

public CounterController(MeterRegistry meterRegistry) {
    this.meterRegistry = meterRegistry;
}

/**
 * Инициализация метрик типа Counter
 */
@PostConstruct
public void init() {
    simpleCounter = Counter.builder("simple.counter")
        .description("Just a simple counter")
        .register(meterRegistry);

    simpleCounterWithTags = Counter.builder(COUNTER_WITH_TAG_NAME)
        .description(COUNTER_WITH_TAG_DESCRIPTION)
        .tag(TAG_NAME_1, "sib")
        .tag(TAG_NAME_2, "111")
        .register(meterRegistry);

    simpleCounterWithTags2 = Counter.builder(COUNTER_WITH_TAG_NAME)
        .description(COUNTER_WITH_TAG_DESCRIPTION)
        .tag(TAG_NAME_1, "msk")
        .tag(TAG_NAME_2, "111")
        .register(meterRegistry);
}

@PutMapping("/counter")
public String incrementCounter() {
    simpleCounter.increment();
    return String.format("Counter has been increases\nCurrent value: %s",
simpleCounter.count());
}

@PutMapping("/counter-with-tags/terbank/sib")
public String incrementCounterTags1() {
    simpleCounterWithTags.increment(1);
    return String.format("Counter has been increases\nCurrent value: %s",
simpleCounterWithTags.count());
}

@PutMapping("/counter-with-tags/terbank/msk")
public String incrementCounterTags2() {
    simpleCounterWithTags2.increment(2);
    return String.format("Counter has been increases\nCurrent value: %s",
simpleCounterWithTags2.count());
}
}

```

С помощью REST API сервиса возможна генерация собственных метрик типа Counter

```

curl -X PUT "MONITOR_URL/counter" -H "accept: */*"
curl -X PUT "MONITOR_URL/counter-with-tags/terbank/msk" -H "accept: */*"

```

В каждом из ответов можно увидеть, что общий счетчик метрик был увеличен на n-ное количество. Пример:

```

Counter has been increases
Current value: 1.0

```

### 2.1.10.1 Настройка конфигурации для OpenShift

Для того, чтобы Prometheus мог идентифицировать сервис и собирать с него

информацию, необходимо создать *Service* и указать путь, на котором располагаются метрики приложения.

```
apiVersion: v1
kind: Service
metadata:
  name: monitoring-rest
annotations:
  description: 'Exposes Prometheus App by Cluster Ip'
  prometheus.io.scrape: 'true'
  prometheus.io.path: '/monitoring-rest/actuator/prometheus'
  prometheus.io.port: '8081'
labels:
  app: monitoring-rest
spec:
  type: LoadBalancer
ports:
  - name: http
    port: 8080
    targetPort: 8080
  - name: http-actuator
    port: 8081
    targetPort: 8081
selector:
  app: monitoring-rest
```

#### 2.1.10.2 Grafana: графики мониторинга

Графики мониторинга метрик можно увидеть в Grafana.

Логин/пароль для входа в Grafana: viewer/viewer.

```
select TIME_FLOOR(__time,'PT1M') as "time",
avg("value") as "avg_value",
"labels.app" as "app",
name as name
from "unimon.gostech_task"
WHERE
name like '%counter%'
and "labels.namespace"='gt-sol-test-coreplatform-01'
and ("labels.application"='monitoring-rest'
OR "labels.app"='monitoring-rest'
OR "labels.SUBSYSTEM"='monitoring-rest')
group by TIME_FLOOR(__time,'PT1M'), "labels.app", name
```

Также существуют и другие типы метрик (DistributionSummary, Gauge, Timer), генерация которых реализована в демо-приложении.

## 3 НАСТРОЙКА НА СОСТАВ ТЕХНИЧЕСКИХ СРЕДСТВ

### 3.1 Общие требования

- маршрутизатор может быть логически разделен на 2 функциональных компонента, отвечающих за решение своих задач:
  - Data Plane – уровень данных, через который проходит весь сетевой трафик.
  - Control Plane – уровень построения и обновления таблиц маршрутизации.
- наличие одной или нескольких выделенных и обособленных одноранговых Interconnect-сетей под цели внутренней коммуникации кластера;
- все сервера кластера должны быть подключены к Interconnect-сетям, для серверов должны быть назначены адреса, и должно быть настроено соединение между всеми серверами кластера;
- увеличьте производительность сети с помощью *Jumbo Frames*, для этого установите значение **MTU** - 9000;
- скорость Ethernet от 1 Гб/с (стандартом является 10 Гб/с);
- отсутствие firewalls и другого ПО, блокирующего или замедляющего трафик;
- внутри кластера должна быть открыта коммуникация по всем портам;
- доступность проверки целостности и качества соединений (Ping) для всех серверов (ICMP);
- сервисы firewalld и SELinux не должны контролироваться системой управления конфигурации (при ее наличии), поскольку они должны быть доступны процессу установки ADS для остановки и выключения (в случае использования CentOS 7.9);
- создание новых сервисов не должно контролироваться системой управления конфигурации (при ее наличии), поскольку оно должно быть доступно процессу установки ADS (в случае использования CentOS 7.9).

При установке с подключением к сети Интернет и использовании CentOS 7.9 предъявляются дополнительные требования:

1. Наличие доступа для серверов ADS, предварительно развернутому в сети организации на выделенном сервере.
2. Наличие доступа для сервера [ProStore](#) к официальному репозиторию Arenadata [ProStore](#) или созданному локальному зеркалу.

## 3.2 Требования к серверам

### 3.2.1 Требования к серверу ProStore

Необходимо не менее 20ГБ свободного пространства.

### 3.2.2 Требования к серверу Arenadata Cluster Manager (ADCM)

При условии использования CentOS 7.9

Требуется виртуальная или физическая машина с операционной системой на базе Linux (kernel 3.10 и yum/rpm).

Необходимо не менее 5 ГБ свободного пространства в директории `/home`.

### 3.2.3 Требования к кластеру серверов Arenadata Streaming (ADS)

1. На серверах должен быть настроен протокол NTP;
2. Следующие файлы не должны контролироваться системой управления конфигурации (при ее наличии), поскольку они должны быть доступны процессу установки для создания и модификации:

- `/etc/hosts`
- `/etc/selinux/config`
- `/etc/sysctl.conf`
- файлы директории `/usr/lib/systemd/system/`;

3. В кластере должны быть открыты порты, представленные в таблице ниже (см. [Таблица 3.1](#)).

Таблица 3.1 Перечень сетевых портов, используемых серверами ADS

Номер порта	Описание использования порта
22	SSH
81	(при установке ADS без подключения к сети Интернет) TCP-порт репозитория Arenadata Enterprise Tools
2015	TCP-порт для отправки метрик на сервер мониторинга
2016	UDP-порт для отправки метрик на сервер мониторинга
8000	TCP-порт для отправки статусов компонентов кластера в ADCM
2181	Порт доступа к сервису ZooKeeper
2888	Порт межсерверного взаимодействия для компонентов кворума ZooKeeper
3888	Порт межсерверного взаимодействия для компонентов кворума ZooKeeper
9092	HTTP-порт доступа к сервису Kafka
9093	HTTPS-порт доступа к сервису Kafka
8081	Порт доступа к сервису Schema-Registry
8082	Порт доступа к сервису Kafka REST Proxy
8088	Порт доступа к компоненту KSQL Server
9898	Порт доступа к сервису Kafka-Manager
9997	JMX-порт для доступа к метрикам сервиса Schema-Registry



Номер порта	Описание использования порта
9998	JMX-порт для доступа к метрикам сервиса Kafka REST Proxy
9999	JMX-порт для доступа к метрикам сервиса Kafka

### 3.2.4 Требования к серверу «СМЭВ3-адаптер»

1. На серверах должен быть настроен протокол [NTP](#).
2. Следующие файлы не должны контролироваться системой управления конфигурации (при ее наличии), поскольку они должны быть доступны процессу установки для создания и модификации:
  - `/etc/hosts`;
  - `/etc/selinux/config`;
  - `/etc/sysctl.conf`;
  - `/usr/lib/systemd/system/`;
  - `/etc/sysconfig/iptables*`;
  - `/etc/firewalld/*`;
  - `/etc/docker/*`.
3. Снаружи сервер должен быть доступен по следующим портам:
  - 22 (SSH);
4. Необходим сетевой доступ до сервера СМЭВ. Например, если используется адрес [http://smev3-n0.test.gosuslugi.ru:5000/transport\\_1\\_0\\_2/](http://smev3-n0.test.gosuslugi.ru:5000/transport_1_0_2/), необходимо организовать к нему доступ.
5. Если для шифрования и подписи сообщений используется КриптоПро JCP 2.0.41618.
6. Необходимо так же подготовить ключи (контейнер) для КриптоПро и лицензию на JCP. В случае отсутствия лицензии будет использоваться пробная лицензия.
7. Если для оформления цифровой подписи используется VipNet PKI необходимо обеспечить сетевой доступ до его сервера.

### 3.2.5 Требования к серверу «Агент ПОДД»

1. На серверах должен быть настроен протокол [NTP](#).
2. Следующие файлы не должны контролироваться системой управления конфигурации (при ее наличии), поскольку они должны быть доступны процессу установки для создания и модификации:
  - `/etc/hosts`;
  - `/etc/selinux/config`;

- `/etc/sysctl.conf`;
- `/usr/lib/systemd/system/`;
- `/etc/sysconfig/iptables*`;
- `/etc/firewalld/*`;
- `/etc/docker/*`.

3. Снаружи сервер должен быть доступен по следующим портам:

- 22 (SSH);

4. Необходим сетевой доступ до сервера [ПОДД](#). Например, если используется адрес [http://smev3-n0.test.gosuslugi.ru:5000/transport\\_1\\_0\\_2/](http://smev3-n0.test.gosuslugi.ru:5000/transport_1_0_2/), то необходимо организовать доступ до хоста *smev3-n0.test.gosuslugi.ru* по порту 5000.

### 3.2.6 Требования к серверу «ETL»

1. На серверах должен быть настроен протокол [NTP](#);
2. Следующие файлы не должны контролироваться системой управления конфигурации (при ее наличии), поскольку они должны быть доступны процессу установки для создания и модификации:

- `/etc/hosts`;
- `/etc/selinux/config`;
- `/etc/sysctl.conf`;
- `/usr/lib/systemd/system/`;
- `/etc/sysconfig/iptables*`;
- `/etc/firewalld/*`;
- `/etc/docker/*`.

3. Должны быть открыты порты, представленные в таблице ниже (см. [Таблица 3.2](#)).

Таблица 3.2 Перечень сетевых портов, используемых серверами ETL

Номер порта	Описание использования порта
22	SSH
8080	TCP-порт, WEB - интерфейс для Apache Spark master
6066	TCP-порт, отправка задач в кластер через REST API для Apache Spark master
7077	TCP-порт, отправка задач в кластер регистрация исполнителей Apache Spark worker для spark master
8081	TCP-порт, WEB-интерфейс Apache Spark worker
8080	TCP-порт, WEB – интерфейс Apache Airflow
5555	TCP-порт, мониторинг задач Apache Airflow
5432	TCP-порт, Airflow Postgres для хранения метаданных
6379	TCP-порт, Redis для хранения очередей

### 3.2.7 Требования к серверу «REST-адаптер»

1. На серверах должен быть настроен протокол [NTP](#);
2. Следующие файлы не должны контролироваться системой управления конфигурации (при ее наличии), поскольку они должны быть доступны процессу установки для создания и модификации:
  - `/etc/hosts`;
  - `/etc/selinux/config`;
  - `/etc/sysctl.conf`;
  - файлы директории `/usr/lib/systemd/system/`;
  - `/etc/sysconfig/iptables*`;
  - `/etc/firewalld/*`;
  - `/etc/docker/*`.
3. Снаружи сервер должен быть доступен по следующим портам:
  - 22 (SSH)
  - 8080

## 4 УСТАНОВКА ПРОГРАММЫ

Установка витрины данных производится без необходимости доступа к сети «Интернет» для скачивания компонент.

### 4.1 Порядок установки

1. Проверить соответствие серверов техническим характеристикам (см. раздел «Настройка на состав технических средств»).
2. Выполнить предварительные действия перед установкой программы (см. раздел «Предварительные действия»).
3. Установить на серверы, в соответствующем порядке, следующее программное обеспечение:
  - Arenadata Cluster Manager (ADCM) - при условии использования CentOS 7.9;
  - Arenadata Streaming (ADS) - при условии использования CentOS 7.9;
  - коннектор Kafka-Postgres;
  - ProStore;
  - СМЭВ QL Сервер;
  - СМЭВ3-адаптер;
  - ПОДД-адаптера - Модуль исполнения запросов;
  - ПОДД-адаптер – Модуль MPPR;
  - ПОДД-адаптер – Модуль MPPW;
  - ПОДД-адаптер – Модуль импорта данных табличных параметров;
  - ПОДД-адаптер – Модуль группировки данных табличных параметров;
  - ПОДД-адаптер – ПОДД-адаптер – Wrapper;
  - Data-uploader – Модуль исполнения асинхронных заданий;
  - REST-uploader – Модуль асинхронной загрузки данных из сторонних источников;
  - ПОДД-адаптер – Модуль подписки;
  - BLOB-адаптер;
  - Сервис формирования документов;
  - ETL;
  - CSV-uploader;
  - REST-адаптер;
  - Counter-provider.
4. Проверить работу всех компонентов программы (см. раздел «Проверка программы»).

Версии компонентов Типового ПО «Витрина данных» конфигурации Стандарт представлены в разделе [1.3 Компоненты системы](#).

## 4.2 Установка ПО ProStore

Установка ProStore должна осуществляться после установки СУБД и брокера сообщений.

ПО ProStore поставляется в виде дистрибутива с компонентами в jar-файлах.

Процесс установки состоит из следующих действий:

- запуск службы Prostore-query-execution-core.

### 4.2.2 Запуск службы Prostore-query-execution-core

1. Создать на сервере директорию для загрузки дистрибутива.
2. Загрузить файлы дистрибутива в созданную директорию.
3. Настроить конфигурационный файл Prostore **application.yaml**.
4. Запустить jar-файлы со значением номера порта, указанным в конфигурации Prostore (по умолчанию — 8080):

```
# запуск файла dtm-query-execution-core-<version>.jar (например, dtm-query-execution-  
core-5.8.0.jar)  
cd ~/prostore/dtm-query-execution-core/target  
java -jar dtm-query-execution-core-<version>.jar
```

#### Примечание:

Чтобы запустить ProStore с другим номером порта, задайте нужное значение с помощью параметра `server:port` в конфигурации Prostore или с помощью переменной окружения `DTM_METRICS_PORT`. Подробнее о параметрах конфигурации и способах их переопределения см. в разделе [Конфигурация системы](#).

5. Проверить корректность функционирования ProStore.

Конфигурационный файл Prostore **application.yaml**

```
#  
# Copyright © 2021 ProStore  
#  
# Licensed under the Apache License, Version 2.0 (the "License");  
# you may not use this file except in compliance with the License.  
# You may obtain a copy of the License at  
#  
# http://www.apache.org/licenses/LICENSE-2.0  
#  
# Unless required by applicable law or agreed to in writing, software  
# distributed under the License is distributed on an "AS IS" BASIS,  
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
# See the License for the specific language governing permissions and  
# limitations under the License.  
#  
  
logging:  
level:  
    ru.datamart.prostore.query.execution: ${DTM_LOGGING_LEVEL:TRACE}
```

```

server:
port: ${DTM_METRICS_PORT:8080}

management:
endpoints:
  enabled-by-default: ${DTM_METRICS_ENABLED:true}
  web:
  exposure:
    include: ${DTM_METRICS_SCOPE:info, health}

core:
plugins:
  active: ${CORE_PLUGINS_ACTIVE:ADP}

http:
  port: ${DTM_CORE_HTTP_PORT:9090}
  tcpNoDelay: ${DTM_CORE_HTTP_TCP_NO_DELAY:true}
  tcpFastOpen: ${DTM_CORE_HTTP_TCP_FAST_OPEN:true}
  tcpQuickAck: ${DTM_CORE_HTTP_TCP_QUICK_ACK:true}

env:
  name: ${DTM_NAME:test}

restoration:
  autoRestoreState: ${AUTO_RESTORE_STATE:true}

matviewsync:
  periodMs: ${MATERIALIZED_VIEWS_SYNC_PERIOD_MS:5000}
  retryCount: ${MATERIALIZED_VIEWS_RETRY_COUNT:10}
  maxConcurrent: ${MATERIALIZED_VIEWS_CONCURRENT:2}

ddlqueue:
  enabled: ${CORE_DDL_QUEUE_ENABLED:true}

datasource:
  edml:
    defaultChunkSize: ${EDML_DEFAULT_CHUNK_SIZE:1000}
    pluginStatusCheckPeriodMs: ${EDML_STATUS_CHECK_PERIOD_MS:1000}
    firstOffsetTimeoutMs: ${EDML_FIRST_OFFSET_TIMEOUT_MS:15000}
    changeOffsetTimeoutMs: ${EDML_CHANGE_OFFSET_TIMEOUT_MS:10000}
  zookeeper:
    connection-string: ${ZOOKEEPER_DS_ADDRESS:localhost}
    connection-timeout-ms: ${ZOOKEEPER_DS_CONNECTION_TIMEOUT_MS:30000}
    session-timeout-ms: ${ZOOKEEPER_DS_SESSION_TIMEOUT_MS:86400000}
    chroot: ${ZOOKEEPER_DS_CHROOT:/adtm}

kafka:
  producer:
  property:
    key.serializer: org.apache.kafka.common.serialization.StringSerializer
    value.serializer: org.apache.kafka.common.serialization.StringSerializer
  cluster:
  zookeeper:
    connection-string: ${ZOOKEEPER_KAFKA_ADDRESS:localhost}
    connection-timeout-ms: ${ZOOKEEPER_KAFKA_CONNECTION_TIMEOUT_MS:30000}
    session-timeout-ms: ${ZOOKEEPER_KAFKA_SESSION_TIMEOUT_MS:86400000}
    chroot: ${ZOOKEEPER_KAFKA_CHROOT:}
  admin:
    inputStreamTimeoutMs: ${KAFKA_INPUT_STREAM_TIMEOUT_MS:2000}
  status.event.publish:
  enabled: ${KAFKA_STATUS_EVENT_ENABLED:false}

```

```

statusMonitor:
  statusUrl: ${STATUS_MONITOR_URL:http://localhost:9095/status}
  versionUrl: ${STATUS_MONITOR_VERSION_URL:http://localhost:9095/versions}

vertx:
  blocking-stacktrace-time: ${DTM_VERTX_BLOCKING_STACKTRACE_TIME:1}
  pool:
  worker-pool: ${DTM_CORE_WORKER_POOL_SIZE:20}
  event-loop-pool: ${DTM_CORE_EVENT_LOOP_POOL_SIZE:20}
  task-pool: ${DTM_CORE_TASK_POOL_SIZE:20}
  task-timeout: ${DTM_CORE_TASK_TIMEOUT:86400000}

cache:
  initialCapacity: ${CACHE_INITIAL_CAPACITY:100000}
  maximumSize: ${CACHE_MAXIMUM_SIZE:100000}
  expireAfterAccessMinutes: ${CACHE_EXPIRE_AFTER_ACCESS_MINUTES:99960}

delta:
  rollback-status-calls-ms: ${DELTA_ROLLBACK_STATUS_CALLS_MS:2000}

statistics:
  enabled: ${CORE_STATISTICS_ENABLED:true}
  threadsCount: ${CORE_STATISTICS_THREADS_COUNT:2}

adp:
datasource:
  user: ${ADP_USERNAME:dtm}
  password: ${ADP_PASS:dtm}
  host: ${ADP_HOST:localhost}
  port: ${ADP_PORT:5432}
  poolSize: ${ADP_MAX_POOL_SIZE:3}
  executorsCount: ${ADP_EXECUTORS_COUNT:3}
  fetchSize: ${ADP_FETCH_SIZE:1000}
  poolRequestTimeout: ${ADP_POOL_REQUEST_TIMEOUT:0}
  preparedStatementsCacheMaxSize: ${ADP_PREPARED_CACHE_MAX_SIZE:256}
  preparedStatementsCacheSqlLimit: ${ADP_PREPARED_CACHE_SQL_LIMIT:2048}
  preparedStatementsCache: ${ADP_PREPARED_CACHE:true}

mppw:
  restStartLoadUrl: ${ADP_REST_START_LOAD_URL:http://localhost:8096/newdata/start}
  restStopLoadUrl: ${ADP_REST_STOP_LOAD_URL:http://localhost:8096/newdata/stop}
  restVersionUrl: ${ADP_MPPW_CONNECTOR_VERSION_URL:http://localhost:8096/versions}
  kafkaConsumerGroup: ${ADP_KAFKA_CONSUMER_GROUP:adp-load}

mppr:
  restLoadUrl: ${ADP_MPPR_QUERY_URL:http://localhost:8094/query}
  restVersionUrl: ${ADP_MPPR_CONNECTOR_VERSION_URL:http://localhost:8094/versions}

```

### 4.3 Установка СМЭВ QL Сервера

Создать новое приложение СМЭВ QL Сервера командой:

```
java -jar smeovql-server-all.jar new <new-app-name>
```

Данная команда создаст структуру папок сервера внутри **<new-app-name>** и исполняемый файл **smeovql**.

### 4.4 Установка СМЭВ3-адаптера

Описание настроек модуля приведено в «Руководстве администратора».

Действия по установке выполняются через SSH консоль технологического пользователя.

Установка СМЭВ3-адаптера возможна, только если были добавлены ключи провайдера электронной подписи (например, [КриптоПро](#)).

Установка СМЭВ3-адаптер возможна, только если были добавлены ключи (контейнер закрытого ключа) для сервиса КриптоПро.

В случае использования для электронной подписи сервиса КриптоПро, необходимо предварительно скачать:

- КриптоПро JCP (<https://www.cryptopro.ru/download?pid=129>) для Java JDK на сервер, где будет установлен СМЭВ3-адаптер.
- Установить загруженное ПО, следуя инструкции и документации на официальном сайте.
- Получить сертификат для установки от уполномоченных лиц и установить его в КриптоПро.

Модуль СМЭВ3-адаптер поставляется в виде JAR-файла. В поставку также входят следующие файлы:

- файл настроек конфигурации модуля СМЭВ3-адаптер `application.yml`;
- файлы для подключения к ProStore: JDBC-драйвер (`dtm-jdbc-driver-*.*.jar`) и `commons-lang3-3.12.0.jar`;
- сконфигурированные rebble-шаблоны.

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить JAR-файл модуля.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`. Пример конфигурации файла `application.yml` и возможные настройки конфигурации модуля см. в разделе [Конфигурация СМЭВ3-адаптер \(application.yml\)](#) Руководства администратора.

Создайте на сервере папку, в которую будут загружены файлы модуля, например, `/opt/smev3-adapter`.

В случае, если ранее была установлена старая версия **СМЭВ3-адаптера**, сделайте его резервную копию.

Загрузите в созданную на предыдущем шаге папку:

- JAR-файл модуля;
- файл настроек конфигурации модуля СМЭВ3-адаптер (`application.yml`);
- файлы JDBC-драйвер (`dtm-jdbc-driver-*.*.jar`) и `commons-lang3-3.12.0.jar` для



подключения к ProStore;

- сконфигурированные rebble-шаблоны.

Кластеризация модуля достигается путем запуска копии экземпляра данного модуля. Оптимальным вариантом является использование оркестраторов, например:

- Kubernetes;
- Openshift;
- Docker-swarm;
- Nomad.

## 4.5 Установка ПОДД-адаптера - Модуль исполнения запросов

Описание настроек модуля приведено в «Руководстве администратора».

### Внимание:

Данное руководство описывает процесс установки *ПОДД-адаптера - Модуль исполнения запросов* в «Витрину данных НСУД», для «Витрины данных Лайт» модуль будет добавлен автоматически при первоначальной установке программы.

Общий процесс установки состоит из следующих действий:

1. Настройка конфигурации **ПОДД-адаптера - Модуль исполнения запросов** в файле `application.yml`.
2. Запуск модуля согласно инструкции см [Запуск ПОДД-адаптера - Модуль исполнения запросов](#).
3. Установка сервисов и необходимых сервисных баз данных.

### Внимание:

Установка данных сервисов выполняется после установки «Core Services».

Действия по установке **ПОДД-адаптера - Модуль исполнения запросов** выполняются через SSH-консоль технологического пользователя.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`. Пример конфигурации файла `application.yml` для **ПОДД-адаптера - Модуль исполнения запросов** см. в разделе [Конфигурация ПОДД-адаптера - Модуль исполнения запросов \(application.yml\)](#) Руководства администратора.

### Примечание:

Значения настроек `MYSQL_USER` для MariaDB и `SUBSCR_DB_USER` для *ПОДД-адаптера - Модуль исполнения запросов*, а также `MYSQL_PASSWORD` и `SUBSCR_DB_PASS`, должны совпадать.

Кластеризация модуля достигается путем запуска копии экземпляра данного модуля. Оптимальным вариантом является использование оркестраторов, например:

- Kubernetes;
- Openshift;
- Docker-swarm;
- Nomad.

## 4.6 Установка ПОДД-адаптер – Модуль MPPR

Описание настроек модуля приведено в «Руководстве администратора».

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`, пример которого и возможные настройки конфигурации модуля см. в разделе [Конфигурация ПОДД-адаптера - Модуль MPPR \(application.yml\)](#) Руководства администратора.

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/home/dir
```

где,

- `file.jar` - название jar-файла;
- `user_name` - имя пользователя, например, `sudo` или `root`;
- `IP` - адрес сервера;
- `/home/dir` - директория на сервере, в которую будет загружен файл.

Кластеризация модуля достигается путем запуска копии экземпляра данного модуля. Оптимальным вариантом является использование оркестраторов, например:

- Kubernetes;
- Openshift;
- Docker-swarm;
- Nomad.

## 4.7 Установка ПОДД-адаптер-Модуль MPPW

Описание настроек модуля приведено в «Руководстве администратора».

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`, пример которого и возможные настройки конфигурации модуля см. в разделе см. [Конфигурация модуля ПОДД-адаптер - Модуль MPPW \(application.yml\)](#) Руководства

администратора.

Создайте на сервере папку, в которую будут загружены файлы модуля, например, `/opt/podd-adapter-mppw`.

В случае, если ранее была установлена старая версия модуля **ПОДД-адаптер – Модуль MPPW**, сделайте его резервную копию.

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/opt/podd-adapter-mppw
```

где,

- `file.jar` - название jar-файла модуля;
- `user_name` - имя пользователя, например, `sudo` или `root`;
- `IP` - адрес сервера;
- `/opt/podd-adapter-mppw` - директория на сервере, в которую будет загружен файл.

Кластеризация модуля достигается путем запуска копии экземпляра данного модуля. Оптимальным вариантом является использование оркестраторов, например:

- Kubernetes;
- Openshift;
- Docker-swarm;
- Nomad.

## 4.8 Установка ПОДД-адаптер – Модуль импорта данных табличных параметров

Описание настроек модуля приведено в «Руководстве администратора».

Действия по установке выполняются через SSH консоль технологического пользователя.

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`. Пример конфигурации файла `application.yml` и возможные настройки конфигурации модуля см. в разделе [Конфигурация модуля ПОДД-адаптер - Модуль импорта данных табличных параметров \(application.yml\)](#) Руководства администратора.

Создайте на сервере папку, в которую будут загружены файлы модуля, например, `/opt/podd-adapter-import-tp`.

В случае, если ранее была установлена старая версия модуля **ПОДД-адаптер - Модуль импорта данных табличных параметров**, сделайте его резервную копию.

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/podd-adapter-import-tp
```

где,

- **file.jar** - название jar-файла модуля;
- **user\_name** - имя пользователя, например, **sudo** или **root**;
- **IP** - адрес сервера;
- **/opt/podd-adapter-import-tp** - директория на сервере, в которую будет загружен файл.

Кластеризация модуля достигается путем запуска копии экземпляра данного модуля. Оптимальным вариантом является использование оркестраторов, например:

- Kubernetes;
- Openshift;
- Docker-swarm;
- Nomad.

## 4.9 Установка ПОДД-адаптер – Модуль группировки данных табличных параметров

Описание настроек модуля приведено в «Руководстве администратора».

Действия по установке выполняются через SSH консоль технологического пользователя.

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла **application.yml**. Пример конфигурации файла **application.yml** и возможные настройки конфигурации модуля см. в разделе [Конфигурация модуля ПОДД-адаптер - Модуль импорта данных табличных параметров \(application.yml\)](#) Руководства администратора.

Создайте на сервере папку, в которую будут загружены файлы модуля, например, **/opt/podd-adapter-import-tp**.

В случае, если ранее была установлена старая версия модуля **ПОДД-адаптер - Модуль импорта данных табличных параметров**, сделайте его резервную копию.

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/podd-adapter-import-tp
```

где,

- **file.jar** - название jar-файла модуля;

- `user_name` - имя пользователя, например, `sudo` или `root`;
- `IP` - адрес сервера;
- `/opt/podd-adapter-import-tp` - директория на сервере, в которую будет загружен файл.

Кластеризация модуля достигается путем запуска копии экземпляра данного модуля. Оптимальным вариантом является использование оркестраторов, например:

- Kubernetes;
- Openshift;
- Docker-swarm;
- Nomad.

## 4.10 Установка ПОДД-адаптер – Wrapper

Описание настроек модуля приведено в «Руководстве администратора».

Действия по установке выполняются через SSH консоль технологического пользователя.

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`. Пример конфигурации файла `application.yml` и возможные настройки конфигурации модуля см. в разделе [Конфигурация модуля ПОДД-адаптер - Wrapper \(application.yml\)](#) Руководства администратора.

Создайте на сервере папку, в которую будут загружены файлы модуля, например, `/opt/podd-avro-defragmentator`.

В случае, если ранее была установлена старая версия модуля **ПОДД-адаптер - Wrapper**, сделайте его резервную копию.

Загрузите в созданную на предыдущем шаге папку:

- JAR-файл модуля;
- файл настроек конфигурации модуля **ПОДД-адаптер - Wrapper** (`application.yml`);

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/opt/podd-avro-defragmentator
```

где,

- `file.jar` - название JAR-файла модуля;
- `user_name` - имя пользователя, например, `sudo` или `root`;
- `IP` - адрес сервера;

- `/opt/podd-avro-defragmentator` - директория на сервере, в которую будет загружен файл.

## 4.11 Установка модуля группировки чанков репликации

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`, пример которого и возможные настройки конфигурации модуля см. в разделе [Конфигурация Модуля группировки чанков репликации \(application.yml\)](#) Руководства администратора.

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/home/dir
```

где,

- `file.jar` - название jar-файла;
- `user_name` - имя пользователя, например, `sudo` или `root`;
- `IP` - адрес сервера;
- `/home/dir` - директория на сервере, в которую будет загружен файл.

## 4.12 Установка DATA-uploader – Модуль исполнения асинхронных заданий

Описание настроек модуля приведено в «Руководстве администратора».

Действия по установке выполняются через SSH консоль технологического пользователя.

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`. Пример конфигурации файла `application.yml` и возможные настройки конфигурации модуля см. в разделе [Конфигурация модуля DATA-Uploader \(application.yml\)](#) Руководства администратора.

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/home/dir
```

где,

- `file.jar` - название JAR-файла;
- `user_name` - имя пользователя, например, `sudo` или `root`;
- `IP` - адрес сервера;
- `/home/dir` - директория на сервере, в которую будет загружен файл.

#### 4.13 Установка REST-uploader – Модуль асинхронной загрузки данных из сторонних источников

Описание настроек модуля приведено в «Руководстве администратора».

Действия по установке выполняются через SSH консоль технологического пользователя.

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`. Пример конфигурации файла `application.yml` и возможные настройки конфигурации модуля см. в разделе [Конфигурация модуля REST-Uploader \(application.yml\)](#) Руководства администратора.

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/home/dir
```

где,

- `file.jar` - название JAR-файла;
- `user_name` - имя пользователя, например, `sudo` или `root`;
- `IP` - адрес сервера;
- `/home/dir` - директория на сервере, в которую будет загружен файл.

#### 4.14 Установка ПОДД-адаптер – Модуль подписки

Описание настроек модуля приведено в «Руководстве администратора».

Действия по установке выполняются через SSH консоль технологического пользователя.

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.

## 5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`. Пример конфигурации файла `application.yml` и возможные настройки конфигурации модуля см. в разделе [Конфигурация модуля ПОДД-адаптер - Модуль подписок \(application.yml\)](#) Руководства администратора.

Создайте на сервере папку, в которую будут загружены файлы модуля, например, `/opt/podd-adapter-replicator`.

В случае, если ранее была установлена старая версия **ПОДД-адаптера – Модуль подписки**, сделайте его резервную копию.

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/opt/podd-adapter-replicator
```

где,

- `file.jar` - название JAR-файла модуля;
- `user_name` - имя пользователя, например, `sudo` или `root`;
- `IP` - адрес сервера;
- `/opt/podd-adapter-replicator` - директория на сервере, в которую будет загружен файл.

Кластеризация модуля достигается путем запуска копии экземпляра данного модуля. Оптимальным вариантом является использование оркестраторов, например:

- Kubernetes;
- Openshift;
- Docker-swarm;
- Nomad.

## 4.15 Установка BLOB-адаптер

Описание настроек модуля приведено в «Руководстве администратора».

Модуль BLOB-адаптер поставляется в виде `jar`-файла. В поставку также входит файл настроек конфигурации модуля BLOB-адаптер (`application.yml`);

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить `jar`-файл модуля.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`.

Пример конфигурации файла `application.yml` и возможные настройки конфигурации модуля см. [Конфигурация BLOB-адаптера \(application.yml\)](#).



Создайте на сервере папку, в которую будут загружены файлы модуля, например, `/opt/blob-adapter`. В случае, если ранее была установлена старая версия модуля BLOB-адаптер, сделайте его резервную копию.

Загрузите в созданную на предыдущем шаге папку:

- jar-файл модуля;
- файл настроек конфигурации модуля BLOB-адаптер (`application.yml`).

## 4.16 Установка Сервиса формирования документов

Описание настроек модуля приведено в «Руководстве администратора».

Действия по установке выполняются через SSH консоль технологического пользователя.

Модуль Сервис формирования документов поставляется в виде JAR-файла. В поставку также входят следующие файлы:

- файл настроек конфигурации модуля *Сервис формирования документов* (`application.yml`);
- файлы для подключения к Prostore: JDBC-драйвер (`dtm-jdbc-driver-*.*.*.jar` и `commons-lang3-3.12.0.jar`).

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить JAR-файл модуля.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`. Пример конфигурации файла `application.yml` и возможные настройки конфигурации модуля см. в разделе [Конфигурация Сервиса Формирования документов \(application.yml\)](#) Руководства администратора.

Создайте на сервере папку, в которую будут загружены файлы модуля, например, `/opt/printable-form-service`. В случае, если ранее была установлена старая версия модуля **Сервис формирования документов**, сделайте его резервную копию.

Загрузите в созданную на предыдущем шаге папку:

- JAR-файл модуля;
- файл настроек конфигурации модуля Сервис формирования документов (`application.yml`);
- файлы JDBC-драйвер (`dtm-jdbc-driver-*.*.*.jar`) и `commons-lang3-3.12.0.jar` для подключения к Prostore.

Кластеризация модуля достигается путем запуска копии экземпляра данного модуля. Оптимальным вариантом является использование оркестраторов, например:

- Kubernetes;
- Openshift;
- Docker-swarm;
- Nomad.

## **4.17 Загрузка и удаление данных через ETL**

### **4.17.1 Общее описание**

#### **4.17.1.1 Общие положения**

Базовый сервис загрузки данных предоставляет возможность асинхронного приёма данных из сторонних источников с целью последующей загрузки их в Витрину данных. Загрузка/обновление данных осуществляется в соответствии с заранее подготовленными Авто-схемами.

Сервис загрузки данных реализуется компонентой ETL, предоставляющей REST API. Доступ к REST API должен осуществляться через Proxy API Datamart Studio. Перед загрузкой необходимо получить токен Proxy API, и в дальнейшем токен используется для авторизации при операциях, описанных в разделах ниже. При получении ошибки авторизации необходимо повторить авторизацию, получить новый токен и использовать его для дальнейшей загрузки.

Для взаимодействия с REST API (через Proxy API) в продуктивной среде на стороне источника данных требуется использовать сертифицированную версию ОС, а также механизмы, соответствующие требованиям безопасности, установленным для эксплуатации системы (например, через программный продукт Postman).

На стороне источника данных должны соблюдаться следующие требования к механизму взаимодействия:

1. Запрещается хранение логина и пароля в открытом виде на диске. Логин и пароль должны выгружаться из безопасного хранилища в память ВМ (или контейнера) при старте взаимодействия с Proxy API для дальнейшего использования, сама ВМ должна находиться в закрытом контуре ИС или ведомства.
2. Рекомендуются реализовать механизм стирания логина и пароля из памяти после успешной аутентификации через Proxy API.

В целях тестирования взаимодействия на тестовой среде может использоваться утилита curl.

### **4.17.2 Особенности реализации**

#### **4.17.2.1 Основные требования к исходным файлам**

Загрузка данных в систему производится в виде сообщений, каждое из которых имеет структуру, представленную на [Рисунок - 4.1](#)

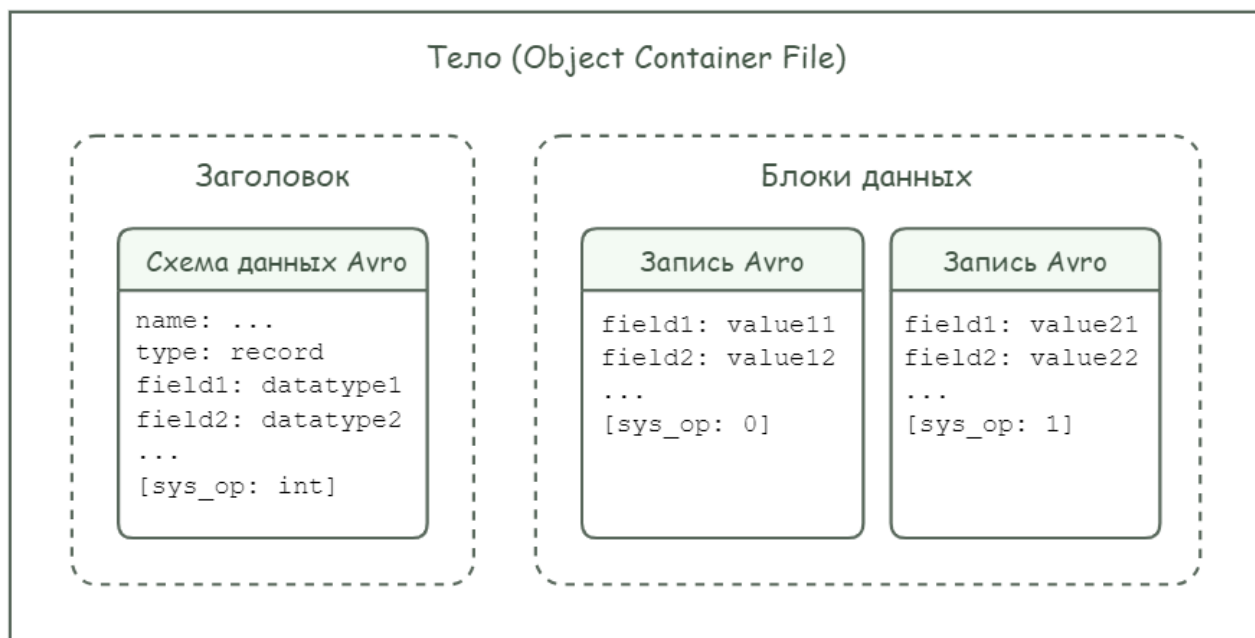


Рисунок - 4.1 Структура загружаемых сообщений

Для успешной загрузки данные должны соответствовать следующим условиям:

1. Тело сообщения представляет собой файл Avro (Object Container File), который состоит из заголовка и блоков данных.
2. Заголовок файла содержит схему данных Avro.
3. Схема данных содержит следующие элементы:
  - имя;
  - тип “record”;
  - перечень полей.
4. Последним полем схемы должно быть указано служебное поле sys\_op с типом данных avro.int.
5. Каждый блок данных содержит запись, представленную в бинарной кодировке.
6. Каждая запись содержит перечень полей и их значений.
7. Состав и порядок полей должны совпадать во всех следующих объектах:
  - в схеме данных заголовка файла Avro;
  - в наборе загружаемых записей;
  - во внешней таблице загрузки (поле sys\_op должно отсутствовать);
  - в таблице-приемнике данных (поле sys\_op должно отсутствовать).

Более подробно про формат данных Avro описано в источнике:  
<https://avro.apache.org/docs/1.10.2/spec.html#Object+Container+Files>

Примечание:

В загружаемой схеме данных Avro и записях Avro важны порядок и тип полей. Имена полей не

сравниваются с именами полей внешней таблицы и таблицы-приемника.

Пример ниже содержит схему данных Avro, используемую для загрузки данных о сотрудниках в таблицу `staff`. Для поля `date_of_birth` указан логический тип Avro, для поля `middle_name` — элемент union (поле является не обязательным для заполнения, поэтому маркер `null` выведен в отдельный параметр).

Пример схемы данных Avro:

```
{
  "name": "staff",
  "type": "record",
  "fields": [
    {
      "name": "id",
      "type": "long"
    },
    {
      "name": "firstname",
      "type": "string"
    },
    {
      "name": "lastname",
      "type": "string"
    },
    {
      "name": "middle_name",
      "type": [
        "null",
        "string"
      ]
    },
    {
      "name": "date_of_birth",
      "type": {
        "logicalType": "timestamp-micros",
        "type": "long"
      }
    },
    {
      "name": "employee_position",
      "type": "string"
    },
    {
      "name": "department_category",
      "type": "long"
    },
    {
      "name": "sys_op",
      "type": "int"
    }
  ]
}
```

Пример ниже содержит набор записей о сотрудниках, загружаемых в таблицу `staff`. Для наглядности примера бинарные данные представлены в JSON-формате.

```
[
  {
    "id": 1000111,
    "firstname": "Елена",
```

```

    "lastname": "Фролова",
    "middle_name": "Андреевна",
    "date_of_birth": 4641084000000000,
    "employee_position": "Менеджер по подбору персонала",
    "department_category": 1,
    "description": "Сотрудники отдела кадров",
    "sys_op": 0
  },
  {
    "id": 1000005,
    "firstname": "Пётр",
    "lastname": "Платонов",
    "middle_name": "",
    "date_of_birth": 5639904000000000,
    "employee_position": "Руководитель отдела кадров",
    "department_category": 1,
    "description": "Сотрудники отдела кадров",
    "sys_op": 0
  }
]

```

#### 4.17.2.2 Особенности реализации ETL

Функциональные особенности реализованного ETL включают в себя следующие пункты:

1. Генерация первичных ключей записей, передаваемых для загрузки, производится на стороне источника.
2. Каждая Avro-структура должна содержать данные только для одной таблицы Витрины.
3. В Avro-структурах данных источник заполняет тип операции **sys\_op**:
  - 0 – для добавления новой или обновления существующей записи;
  - 1 – для удаления существующей записи (см. пример записей Avro в [Раздел 4.17.2.1](#)).
4. ETL не выполняет преобразования данных, предназначенных для загрузки в Витрину данных.
5. При выполнении операций, требующих консистентности данных, в рамках одной дельты могут быть только операции одного типа: либо добавления/обновления, либо удаления. При этом в рамках одной дельты первичные ключи всех записей должны быть уникальны.
6. Не должно быть двух версий одной записи в рамках одной дельты.
7. Нельзя менять порядок атрибутов в авро-схеме, поскольку данные при загрузке в БД распределяются в соответствии с тем перечнем, который был указан в авро-схеме.

#### 4.17.2.3 Получение токена Proxy API

Этапы настройки Proxy API включают в себя следующие шаги:

1. Для начала необходимо пройти авторизацию в Datamart Studio. Сделать это можно, направив запрос ниже:

```
curl -X POST ¥  
'http://<ip-studio>:8088/api/v1/auth_system' ¥  
-d "username=<username>" ¥  
-d "password=<password>" ¥  
-d "organization_ogrn=<organization_ogrn>" ¥  
-d "datamart_mnemonic=<datamart_mnemonic>"
```

где:

- **ip-studio** — ip-адрес Datamart Studio;
- **username** — имя пользователя IAM;
- **password** — пароль пользователя IAM;
- **organization\_ogrn** — ОГРН организации, в рамках которой развёрнута Витрина данных;
- **datamart\_mnemonic** — мнемоника Витрины (пример: eduejd##, где ## – номер региона).

**Примечание:**

Безопасность передачи данных по протоколу HTTP обеспечивается защищенной сетью. Требуется обращать внимание на протокол запроса. Если он будет некорректным (например, HTTPS вместо HTTP), то ответ сервера будет содержать ошибку с кодом 500 - InternalServerError.

Пример успешного ответа Datamart Studio на запрос токена представлен ниже:

[illegible]

2. С полученным токеном послать запрос для подключения к API инсталляции приложения типового ПО Витрины данных для выбранной организации:

```
curl -X <method>
'http://<ip-
studio>:8088/api/v1/secure/<organization_ogrn>/<datamart_mnemonic>/<installation_name>/
<installation_id>/<request_path>' ¥
-H "Authorization: Bearer <access_token>" ¥
-H "<headers>" ¥
-d "<data>"
```

где:

- **ip-studio** — ip-адрес Datamart Studio;
- **organization\_ogrn** — ОГРН организации, в рамках которой развёрнута Витрина данных;
- **datamart\_mnemonic** — мнемоника Витрины (пример: eduejd##, где ## – номер региона);
- **installation\_name** — имя инсталляции в целевой Витрине;
- **installation\_id** — идентификатор инсталляции (присутствует в её названии);
- **request\_path** — URI оригинального API инсталляции;
- **access\_token** — токен Proxy API;
- **headers** — заголовки запроса;
- **data** — данные запроса.

### 4.17.3 Загрузка / удаление данных

#### 4.17.3.1 Загрузка/ удаление данных

Сначала источник данных формирует и передаёт файлы по REST API, которые накапливаются Компонентом ETL. Далее данные загружаются и вставляются в Витрину данных через внешние таблицы, или удаляются. Статусы обработки каждой операции необходимо отслеживать через Endpoint **/status**.

Информация о каждом шаге процесса содержится в подразделах ниже.

##### 4.17.3.1.1 Начало операции загрузки / удаления согласованных данных (Endpoint – newDelta)

Под Endpoint'ом **/newDelta** регистрируется новая порция данных. Для того чтобы начать работу с данными, источнику данных необходимо сгенерировать UUID (идентификатор для новой порции данных) и вставить его в запрос с Endpoint'ом **/newDelta**.

**Согласованные данные** — это данные для нескольких таблиц, которые должны попасть в Витрину данных за одну операцию вставки (то есть в одной дельте), а значит будут доступны для потребителя одномоментно (такой способ загрузки актуален, когда необходимо обновить данные).

Пример набора данных, который будет загружен или удален в рамках одной дельты представлен ниже:

```
{
  "requestId": "6B29FC40-CA47-1067-B31D-00DD010662DA",
  "dataSetName": ["product", "stock"]
}
```



где:

- **requestId** — идентификатор порции изменений (дельты);
- **dataSetName** — массив имен набора данных (product и stock - имена таблиц).

Запрос с Endpoint'ом **/newDelta** будет иметь вид:

```
curl -X POST "http://<ip-  
studio>:8088/api/v1/secure/<organization_ogrn>/<datamart_mnemonic>/<installation_name>/  
<installation_id>/newDelta" -H "Authorization: Bearer <access_token>" -H 'Content-Type:  
application/json' -d '{"requestId": "6B29FC40-CA47-1067-B31D-00DD010662DA",  
"dataSetName": ["product", "stock"]}'
```

где:

- **requestId** — идентификатор порции изменений (дельты), который был ранее сгенерирован;
- **dataSetName** — имя набора данных (имена таблиц).

#### Примечание:

Данные для обновления/вставки и для удаления должны быть отдельно зарегистрированы в newDelta с разными requestId. Если требуется обновить/вставить данные, то сначала нужно зарегистрировать порцию данных через newDelta с requestId, затем прислать данные с зарегистрированным requestId через endpoint /partOfDelta (описан в [Раздел 4.17.3.1.2](#)). Допускается передача как в одном файле одним запросом, так и в двух файлах двумя запросами - это не имеет значения. Главное условие - нужно отправлять только данные на обновление и вставку, и они должны быть уникальны по первичному ключу. Также в них не должно быть одинаковых записей с одним первичным ключом. Пример: отправлены два обновления одной записи (с одинаковым первичным ключом). В этом случае в хранилище Prostore попадет только одна запись, и неизвестно какая, поэтому дублей по первичному ключу отправленных с одним requestId быть не должно!

Чтобы проверить статус выполнения запроса, необходимо направить запрос с Endpoint'ом **/status** (пример запроса описан в [Раздел 4.17.4.1](#))

Если обработка запроса завершится успешно, то Витрина данных вернёт JSON-ответ, содержащий статус-сообщение об успешной операции:

```
{  
  "requestId": "6B29FC40-CA47-1067-B31D-00DD010662DA",  
  "dataSets": [  
    "product",  
    "stock"  
  ],  
  "inDeltaFlag": true,  
  "statusCode": "SUCCESS",  
  "statusMessage": "Загрузка порции данных успешно завершена"  
  "errors": []  
}
```

где:

- **requestId** — идентификатор порции изменений (дельты);
- **statusCode** — код возвращаемого статуса (SUCCESS – запрос выполнен успешно).

Пример JSON-ответа на проверку статуса, завершившегося ошибкой, приведен ниже:

```
{
  "requestId": "6B29FC40-CA47-1067-B31D-00DD010662DA",
  "statusCode": "REQUESTID_ALREADY_EXIST",
  "statusMessage": "Дельта с requestId = 6B29FC40-CA47-1067-B31D-00DD010662DA уже существует. Статус загрузки дельты: SUCCESS"
}
```

где:

- **requestId** — идентификатор порции изменений (дельты) который необходимо проверить;
- **statusCode** — код возвращаемого статуса (REQUESTID\_ALREADY\_EXIST – идентификатор уже существует в БД);
- **statusMessage** — сообщение с описанием кода ошибки.

#### Примечание:

Для удаления записей необходимо зарегистрировать новую дельту во Endpoint'e **/newDelta** с новым **requestId**, и по зарегистрированному **requestId** должны быть присланы только данные на удаление.

#### 4.17.3.1.2 Загрузка / удаление согласованных данных (Endpoint – **partOfDelta**)

На данном шаге выполняется накапливание порции данных, создаётся вставка в Витрину данных по Endpoint'у **isLastChunk**.

#### Примечание:

Если в процессе загрузки вызван метод **newDelta**, то текущая загрузка будет прервана и порция не попадет в Витрину данных.

Чтобы отправить последнюю порцию данных для таблицы **product**, необходимо направить запрос с Endpoint'ом **/partOfDelta** с указанием **dataSetName=product** и **isLastChunk=true** (что означает что данная порция данных - последняя). Обработка и загрузка данных не начнётся, пока не будет направлен такой же запрос, но уже по таблице **stock: dataSetName=stock, isLastChunk=true**.

Пример запроса на загрузку данных под ранее созданный набор данных:

```
curl -X POST "http://<ip-studio>:8088/api/v1/secure/<organization_ogrn>/<datamart_mnemonic>/<installation_name>/<installation_id>/partOfDelta" -H "Authorization: Bearer <access_token>" -F upload=@"/product.avro" -F dataSetName=product -F chunkNumber=0 -F isLastChunk=false -F requestId=a6212a7d-4526-4e2d-89a7-9828f380c91d
```

где:

- **upload** — загружаемый avro-файл (пример avro-файла с данными представлен в разделе 2);
- **dataSetName** — имя набора данных (имя таблицы);
- **chunkNumber** — номер порции dataSet в рамках дельты;
- **isLastChunk** — флаг последней порции dataSet;
- **requestId** — идентификатор порции изменений (дельты).

В результате успешной загрузки при проверке статуса **requestId** (пример запроса по

Endpoint'y `/status` представлен в [Раздел 4.17.4.1](#)) на запрос Витрина данных вернёт JSON-ответ, содержащий статус-сообщение об успешной операции.

Пример успешной загрузки:

```
{
  "requestId": "f3947645-88c8-4044-bd8b-de273f8a8461",
  "statusCode": "SUCCESS",
  "statusMessage": "Порция получена."
}
```

где:

- `requestId` — идентификатор порции изменений (дельты);
- `statusCode` — статус код результата запроса (SUCCESS - запрос выполнен успешно);
- `statusMessage` — описание статусного сообщения.

Если загрузка прервалась ошибкой, то при проверке статуса `requestId` (пример запроса по Endpoint'y `/status` представлен в [Раздел 4.17.4.1](#)) на запрос Витрина данных вернёт JSON-ответ с описанием ошибки.

Пример загрузки, прерванной ошибкой:

```
{
  "requestId": "aef2f195-b037-4aaa-b171-f2746511e7e2",
  "dataSets": [
    "stock"
  ],
  "inDeltaFlag": true,
  "statusCode": "ERROR",
  "statusMessage": "Произошла ошибка"
  "errors": [
    {
      "dataSet": "stock",
      "errorType": "INSERT",
      "message": "Ошибка вставки в таблицы: stock"
    }
  ]
}
```

где:

- `requestId` — идентификатор порции изменений (дельты);
- `dataSets` — массив имен набора данных (имен таблиц где была допущена ошибка);
- `inDeltaFlag` = true — загрузка согласованных данных производилась через endpoint `/partOfDelta`;
- `status` — статус код результата запроса (ERROR – внутренняя ошибка);
- `statusMessage` — описание статусного сообщения;
- `errors` — массив, ошибки загрузки или парсинга входящих данных;
- `dataSet` — название таблицы где допущена ошибка;
- `errorType` — тип ошибки;

- **message** — описание ошибки.

Примечание:

Возможна ситуация, когда после падения ETL приходит запрос с **requestId**, который был до падения, в данном случае Витрина данных возвращает ошибку со статусом NOT\_FOUND. Необходимо снова направить запрос по Endpoint'у **/newDelta** с новым **requestId** и начать процесс загрузки заново.

#### 4.17.3.1.3 Загрузка / удаление несогласованных данных (Endpoint – data)

Для загрузки несогласованных данных поддерживается возможность накопления данных, аналогично загрузке согласованных данных, описанной в [Раздел 4.17.3.1.2](#).

**Несогласованные данные** – могут быть вставлены в разных дельтах и будут доступны потребителю постепенно по мере загрузки. Этот способ подходит для первоначальной загрузки, когда еще нет потребителей.

Вставка в Витрину данных выполнится после накопления порции или по флагу **isLast**, который используется для последней порции данных. Флаг **isLast** подаёт сигнал для завершения формирования дельты, для того чтобы выполнить вставку накопленных данных и закрыть транзакцию.

Пример запроса:

```
curl -X POST "http://<ip-studio>:8088/api/v1/secure/<organization ogrn>/<datamart_mnemonic>/<installation_name>/<installation_id>/data" -H "Authorization: Bearer <access_token>" -F upload=@"./product.avro" -F dataSetName=product -F isLast=false -F requestId=a6212a7d-4526-4e2d-89a7-9828f380c91d
```

где:

- **upload** — загружаемый авро-файл (пример авро-файла с данными представлен в [Раздел 4.17.2.1](#));
- **dataSetName** — имя набора данных (имя таблицы);
- **isLast** — флаг последней порции данных (сигнал для завершения формирования дельты, для того чтобы выполнить вставку накопленных данных и закрыть транзакцию.);
- **requestId** — идентификатор порции изменений (дельты).

В результате успешной операции при проверке статуса запроса по Endpoint'у **/status** (пример запроса описан в [Раздел 4.17.4.1](#)) Витрина данных вернёт JSON-ответ, содержащий статус-сообщение:

```
{
  "requestId": "6B29FC40-CA47-1067-B31D-00DD010662DA",
  "statusCode": "SUCCESS",
  "statusMessage": "Порция получена."
}
```

где:

- **requestId** — идентификатор порции изменений (дельты);
- **statusCode** — статус код результата запроса (SUCCESS - запрос выполнен успешно);

- **statusMessage** — описание статусного сообщения.

Если загрузка прервалась ошибкой, то при проверке **requestId** (пример запроса по Endpoint'у **/status** представлен в [Раздел 4.17.4.1](#)) Витрина данных вернёт JSON-ответ с описанием ошибки.

Пример JSON-ответа:

```
{
  "requestId": "6B29FC40-CA47-1067-B31D-00DD010662DA",
  "statusCode": "NOT_FOUND",
  "statusMessage": "Не найдена дельта с requestId = 6B29FC40-CA47-1067-B31D-00DD010662DA"
}
```

где:

- **requestId** — идентификатор порции изменений (дельты);
- **statusCode** — статус код результата запроса (NOT\_FOUND - данные по requestId были утеряны в результате остановки сервиса, необходимо зарегистрировать новую дельту и снова загрузить данные);
- **statusMessage** — описание статусного сообщения.

#### 4.17.3.1.4 Описание возвращаемых кодов

Таблица 4.2 Описание возвращаемых кодов

Наименование	Код	Описание
EMPTY_ATTACHMENT	400	Нет файла вложения
ERROR	500	Внутренняя ошибка
NOT_FOUND	400	Данные не найдены, либо были утеряны в результате остановки сервиса
PROCESSING	400	Идет обработка данных
REQUESTID_ALREADY_EXIST	400	<b>requestId</b> уже зарегистрирован
SUCCESS	200	Успешное выполнение
UNREGISTERED_DATASETNAME	400	Незарегистрированный набор данных
WRONG_ENDPOINT	400	<b>requestId</b> зарегистрирован для другого Endpoint'a

### 4.17.4 Проверка статусной информации по загрузке

#### 4.17.4.1 Проверка статусной информации по загрузке / удалению данных (Endpoint – status)

В данном разделе производится проверка статусной информации из сервисных таблиц по **requestId**.

Пример запроса:

```
Curl -X GET "http://<ip-studio>:8088/api/v1/secure/<organization_ogrn>/<datamart_mnemonic>/<installation_name>/<installation_id>/status/<requestId>" -H "Authorization: Bearer <access_token>"
```

где:

- **requestId** — UUID идентификатор порции изменений (дельты).

Пример ответа на такой запрос представлен ниже.

```
{
  "requestId": "13f2475e-f3dc-4c9e-b2f6-3a98320261f1",
  "inDeltaFlag": false,
  "dataSets": [
    "stock"
  ],
  "status": "ERROR",
  "statusMessage": "Произошла ошибка",
  "errors": [
    {
      "dataSet": "stock",
      "errorType": "PARCING",
      "message": "Неверно указан тип поля count_pieces: LONG. Ожидается: INTEGER"
    },
    {
      "dataSet": "stock",
      "errorType": "PARCING",
      "message": "Неверно указан тип поля product_id: LONG. Ожидается: INTEGER"
    }
  ]
}
```

где:

- **requestId** — UUID идентификатор порции изменений (дельты);
- **inDeltaFlag = false** — загрузка несогласованных данных производилась через endpoint /data;
- **dataSets** — массив имен набора данных (имен таблиц где была допущена ошибка);
- **status** — статус код результата запроса (NOT\_FOUND, PROCESSING, ERROR, SUCCESS);
- **statusMessage** — описание статусного сообщения;
- **errors** — массив, ошибки загрузки или парсинга входящих данных;
- **dataSet** — название таблицы где допущена ошибка;
- **errorType** — тип ошибки;
- **message** — описание ошибки.

#### 4.17.5 Работа с вложениями через S3

##### 4.17.5.1 Работа с вложениями через S3

###### 4.17.5.1.1 Загрузка данных в хранилище (Endpoint – uploadAttachment)

Перед загрузкой источнику данных необходимо сгенерировать UUID (идентификатор для новой порции данных) и вставить его в запрос с Endpoint'ом **/uploadAttachment**. При совпадении имен вложений в хранилище, вложение перезаписывается. Пример запроса на загрузку вложения в хранилище представлен ниже:

```
curl -X POST "http://<ip-  
studio>:8088/api/v1/secure/<organization_ogrn>/<datamart_mnemonic>/<installation_name>/  
<installation_id>/uploadAttachment" -H "Authorization: Bearer <access_token>" -F  
upload=@"document.pdf" -F requestId=13f2475e-f3dc-4c9e-b2f6-3a98320261f1 -F name=Doc_1
```

где:

- **upload** — путь до загружаемого файла-вложения;
- **requestId** — UUID идентификатор запроса;
- **name** — уникальное имя вложения.

После успешной загрузки при проверке по Endpoint'у **/status** (пример запроса описан в [Раздел 4.17.4.1](#)) Витрина данных вернёт JSON-ответ, содержащий статус-сообщение:

```
{  
  "requestId": "13f2475e-f3dc-4c9e-b2f6-3a98320261f1",  
  "statusCode": "UPDATED",  
  "statusMessage": "Файл Doc_1 успешно обновлен."  
}
```

где:

- **requestId** — UUID идентификатор запроса;
- **statusCode** — статус код результата запроса (SUCCESS - запрос выполнен успешно; UPDATED – данные обновлены);
- **statusMessage** — описание статусного сообщения.

Пример неуспешной загрузки после проверки по endpoint'у **/status** (пример запроса описан в [Раздел 4.17.4.1](#)) представлен ниже:

```
{  
  "requestId": "13f2475e-f3dc-4c9e-b2f6-3a98320261f1",  
  "statusCode": "ERROR",  
  "statusMessage": "Произошла ошибка"  
}
```

где:

- **requestId** — UUID идентификатор запроса;
- **statusCode** — статус код результата запроса (ERROR – запрос завершился ошибкой);
- **statusMessage** — описание статусного сообщения.

#### 4.17.5.1.2 Удаление данных из хранилища (Endpoint – deleteAttachment)

Для того чтобы удалить вложения из хранилища S3 необходимо направить следующий запрос:

```
curl -X DELETE "http://<ip-  
studio>:8088/api/v1/secure/<organization_ogrn>/<datamart_mnemonic>/<installation_name>/  
<installation_id>/deleteAttachment/Doc_1/requestId/<requestId>" -H "Authorization:  
Bearer <access_token>"
```

где:

- **requestId** — UUID идентификатор запроса.

В результате успешного удаления Витрина данных вернёт JSON-ответ, содержащий статус-сообщение:

```
{
  "requestId": "13f2475e-f3dc-4c9e-b2f6-3a98320261f1",
  "statusCode": "SUCCESS",
  "statusMessage": "Файл Дос_1 успешно удален."
}
```

где:

- **requestId** — UUID идентификатор запроса;
- **statusCode** — статус код результата запроса (SUCCESS - запрос выполнен успешно);
- **statusMessage** — описание статусного сообщения.

Если удаление завершилось ошибкой, то Витрина данных вернёт JSON-ответ с кодом ошибки:

```
{
  "requestId": "13f2475e-f3dc-4c9e-b2f6-3a98320261f1",
  "statusCode": "NOT_FOUND",
  "statusMessage": "Файл Дос_1 не найден."
}
```

где:

- **requestId** — UUID идентификатор запроса;
- **statusCode** — статус код результата запроса (NOT\_FOUND);
- **statusMessage** — описание статусного сообщения.

#### 4.17.5.1.3 Описание возвращаемых кодов

Таблица 4.3 Описание возвращаемых кодов

Наименование	Код	Описание
EMPTY_ATTACHMENT	400	нет файла вложения
ERROR	500	Внутренняя ошибка
SUCCESS	200	Успешное выполнение
UPDATED	200	данные обновлены

### 4.17.6 Маппинг данных

#### 4.17.6.1 Маппинг данных (Endpoint – generateMapping)

В данном разделе описывается генерация файла маппинга. Endpoint предназначен для первичной настройки, а также перенастройки сервиса в случае изменения модели данных Витрины.

Файл маппинга генерируется источником данных в формате **Kotlin-script**. В файле описана модель данных Витрины данных в виде структуры объектов Kotlin (**table**, **column**). Объекты **table** описывают таблицы, каждый из них содержит имя таблицы и список колонок в том порядке, в котором они созданы в Витрине. Объекты **column** описывают колонки, каждый из них содержит имя колонки, тип данных, признак обязательности (nullable), признак



первичного ключа.

Файл используется сервисом для описания модели данных и валидации входящих данных. Выполняются следующие проверки:

- проверяется соответствие состава полей входящей avro-структуры составу полей, описанных в файле маппинга;
- проверяется соответствие порядка полей входящей avro-структуры порядку полей, описанных в файле маппинга;
- проверяется соответствие типов данных полей входящей avro-структуры типам полей, описанных в файле маппинга. Для полей с установленным признаком обязательности (`nullable = false`) выполняется проверка на `null`.

При вызове Endpoint'a `/generateMapping` сервис генерирует файл на основе информации о модели, полученной из развернутой Витрины данных. Файл складывается сервисом на диск, а также возвращается в ответе на вызов.

Пример запроса на генерацию маппинга представлен ниже:

```
curl -X GET "http://<ip-studio>:8088/api/v1/secure/<organization ogrn>/<datamart_mnemonic>/<installation_name>/<installation_id>/generateMapping/aef2f195-0001-4aaa-b171-f2746511e889" -H "Authorization: Bearer <access_token>"
```

Результат Витрина данных вернёт в формате Kotlin-script:

```
import ru.supercode.mapping.common.ColumnType.*
import ru.supercode.mapping.mapper.dsl.mappingAvro
mappingAvro {
    table("product") {
        column("id", INTEGER) { nullable = false; primary = true; }
        column("name", STRING) { nullable = false; }
    }
    table("stock") {
        column("product_id", INTEGER) { nullable = false; primary = true; }
        column("count_pieces", INTEGER) { nullable = false; }
    }
}
```

#### 4.17.7 Валидация данных

##### 4.17.7.1 Валидация данных

Валидация порции данных производится в момент обработки и вставки.

Примечание:

Помимо валидации данных осуществляется валидация параметров запроса. Во всех Endpoint'ax `requestId` должен быть в формате UUID.

В случае ошибок при валидации результат будет возвращен при вызове Endpoint'a `/status`.

**Ошибки, возникающие в процессе обработки Endpoint'a `/newDelta`:**

- отклоняются запросы, которые получены в момент обработки порции данных (`statusCode: PROCESSED`);
- если пустой параметр `dataSetName`;

- прислан запрос с уже зарегистрированным `requestId` и `statusCode` данного `requestId` не равен `NOT_FOUND` или `WAIT_DATA`.

#### Ошибки, возникающие в процессе обработки Endpoint'a `/partOfDelta`:

- прислан запрос с незарегистрированным `requestId`;
- прислан запрос с уже зарегистрированным `requestId` и `statusCode` данного `requestId` не равен `NOT_FOUND` или `WAIT_DATA`;
- прислан запрос с `requestId` зарегистрированным для Endpoint'a `/data`;
- прислан запрос с параметром `dataSetName`, который не был зарегистрирован в Endpoint'e `/newDelta`;
- нет файла вложения в параметре `upload`.

#### Ошибки, возникающие в процессе обработки Endpoint'a `/data`:

- отклоняются запросы, которые получены в момент обработки порции данных (`statusCode: PROCESSED`);
- прислан запрос с незарегистрированным `requestId`;
- прислан запрос с уже зарегистрированным `requestId` и `statusCode` данного `requestId` не равен `NOT_FOUND` или `WAIT_DATA`;
- прислан запрос с `requestId` зарегистрированным для Endpoint'a `/partOfDelta`;
- нет файла вложения в параметре `upload`.

#### Ошибки, возникающие в процессе обработки Endpoint'a `/uploadAttachment`:

- нет файла вложения в параметре `upload`.

#### Ошибки, возникающие в процессе обработки Endpoint'a `/generateMapping`:

- не созданы логические таблицы в схеме.

### 4.17.8 Установка Apache Airflow

1. Клонировать репозиторий Apache Airflow из официального репозитория:

```
git clone https://github.com/apache/airflow.git
```

2. Перейти в папку **airflow**
3. Создать в папке файл `Dockerfile.custom` со следующим содержимым:

```
FROM ${BASE_AIRFLOW_IMAGE}
SHELL ["/bin/bash", "-o", "pipefail", "-e", "-u", "-x", "-c"]
USER 0
# Install Java
RUN mkdir -pv /usr/share/man/man1 ¥¥
&& mkdir -pv /usr/share/man/man7 ¥¥
&& curl -fsSL
https://adoptopenjdk.jfrog.io/adoptopenjdk/api/gpg/key/public ¥|
apt-key add - ¥¥
```

```

&& echo 'deb https://adoptopenjdk.jfrog.io/adoptopenjdk/deb/ buster
main' > ¥¥
/etc/apt/sources.list.d/adoptopenjdk.list ¥¥
&& apt-get update ¥¥
&& apt-get install --no-install-recommends -y ¥¥
adoptopenjdk-8-hotspot-jre ¥¥
&& apt-get autoremove -yqq --purge ¥¥
&& apt-get clean ¥¥
&& rm -rf /var/lib/apt/lists/¥*
# Установка JDR 1.8
ENV JAVA_HOME=/usr/lib/jvm/adoptopenjdk-8-hotspot-jre-amd64
USER ${AIRFLOW_UID}
# Установка компонента apache.spark
RUN pip install --upgrade pip ¥¥
&& pip install --no-cache-dir --user 'apache-airflow[apache.spark]'
```

Установка Apache Airflow выполняется с помощью *Docker*. Установка *Docker* описана в разделе «Установка Arenadata Cluster Manager (ADCM)» [Установка Arenadata Cluster Manager \(ADCM\)](#).

1. Создайте новый образ в *Docker*, в той же директории, где расположен файл

`Dockerfile.custom`.

Пример:

```

docker build . ¥¥
--build-arg BASE_AIRFLOW_IMAGE="apache/airflow:2.0.1-python3.8" ¥¥
-f Dockerfile.java8 ¥¥
-t airflow:2.0.1-python3.8-custom
```

2. Скачать файл `docker-compose.yaml` с официального сайта Airflow  
<https://Airflow.apache.org/docs/apache-airflow/2.0.1/docker-compose.yaml>
3. Установить `docker-compose.yaml` через SSH-консоль технологического пользователя.

- Выдать права на исполнение файла:

```
sudo chmod +x /usr/local/bin/docker-compose
```

- Запустить sudo:

```
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

- Проверить корректность установки:

```
$ docker-compose --version
docker-compose version 1.28.6, build 1110ad01
```

4. Перейти в директорию с файлом `docker-compose.yaml`

Например:

```
cd ~/direct
```

5. Инициализировать компоненты:

```
docker-compose up airflow-init
```

6. Запустить сервисы с новым образом:

```
docker-compose up
AIRFLOW_IMAGE_NAME=apache/airflow:2.0.1-python3.8-custom
```

#### 4.17.9 Установка Apache Spark

Установка *Docker* описана в разделе «Установка Arenadata Cluster Manager (ADCM)».

1. Клонировать репозиторий Apache Airflow из официального репозитория

```
git clone https://github.com/big-data-europe/docker-spark.git
```

2. Перейти в директорию `docker-spark`

Пример:

```
cd ~/direct
```

3. Выполнить скрипт `./build.sh` или вручную последовательно запустить следующие команды:

- сборка базового образа

```
docker build -t bde2020/spark-base:3.1.1-hadoop3.2
```

- сборка образа мастер

```
docker build -t bde2020/spark-master:3.1.1-hadoop3.2
```

- сборка образа воркера

```
docker build -t bde2020/spark-worker:3.1.1-hadoop3.2
```

4. Проверить наличие собранных образов:

Выполнить команду `docker images`:

REPOSITORY	TAG	IMAGE	ID	CREATED	SIZE
bde2020/spark-worker	3.1.1-hadoop3.2	05c349b4646f	4 minutes ago	460MB	
bde2020/spark-master	3.1.1-hadoop3.2	7918c5357d6d	4 minutes ago	460MB	
bde2020/spark-base	3.1.1-hadoop3.2	5430434220d2	4 minutes ago	460MB	

1. Перейти в директорию *docker-spark*, где располагается файл `docker-compose.yml`

Пример:

```
cd ~/direct
```

2. Запустить Apache Spark командой:

```
docker-compose up
```

3. Проверить командой:

```
::
```

```
docker ps
```

В списке запущенных образов должны присутствовать **spark-worker** и **spark-master**.

#### 4.17.10 Установка Apache Hadoop

Установка *Docker* описана в разделе «Установка Arenadata Cluster Manager (ADCM)».

1. Клонировать репозиторий Apache Hadoop из официального репозитория:

```
git clone https://github.com/big-data-europe/docker-hadoop.git
```

Перейти в папку **docker-hadoop**

Например:

```
cd ~/direct
```

2. Выполнить скрипт **make build** или вручную последовательно запустить следующие команды:

- сборка базового образа:

```
docker build -t bde2020/hadoop-base:master ./base
```

- сборка остальных образов:

```
docker build -t bde2020/hadoop-namenode:master ./namenode
docker build -t bde2020/hadoop-datanode:master ./datanode
docker build -t bde2020/hadoop-resourcemanager:master
./resourcemanager
docker build -t bde2020/hadoop-nodemanager:master ./nodemanager
docker build -t bde2020/hadoop-historyserver:master
./historyserver
docker build -t bde2020/hadoop-submit:master ./submit
```

3. Проверить наличие собранных образов:

Выполнить команду **docker images**:

```
REPOSITORY TAG IMAGE ID CREATED SIZE
bde2020/hadoop-submit master 665a424edc23 9 minutes ago 1.37GB
bde2020/hadoop-historyserver master ff53bf6835e2 9 minutes ago 1.37GB
bde2020/hadoop-nodemanager master c48c47bc840f 9 minutes ago 1.37GB
bde2020/hadoop-resourcemanager master 74fc55d664d2 9 minutes ago 1.37GB
bde2020/hadoop-datanode master f69c6460a292 9 minutes ago 1.37GB
bde2020/hadoop-namenode master 7a8250da8510 9 minutes ago 1.37GB
bde2020/hadoop-base master aeb6500ab4b5 10 minutes ago 1.37GB
```

1. Перейти в директорию **docker-hadoop**, где располагается файл **docker-compose.yml**

Например:

```
cd ~/direct
```

2. Поднять Apache Hadoop командой:

```
docker-compose up
```

3. Проверить командой:

```
docker ps
```

В списке запущенных образов должны присутствовать `hadoop-resource-manager`, `hadoop-historyserver`, `hadoop-datanode`, `hadoop-nodemanager`, `hadoop-namenode`

#### 4.17.11 Установка Tarantool(Vinyl)

Установка docker описана в разделе 3.3.2. Установка Менеджера кластера ADCM.

1. Клонировать репозиторий Tarantool из официального репозитория:

```
git clone https://github.com/tarantool/docker.git
```

2. Выполнить сборку согласно инструкции: <https://github.com/tarantool/docker#how-to-push-an-image-for-maintainers>
3. Перейти в клонированную директорию `docker`

Например:

```
cd ~/direct
```

4. Выполнить сборку образа `docker` для Tarantool

```
export TAG=2
export OS=alpine DIST=3.9 VER=2.8.0
PORT=5200 make -f .gitlab.mk build
```

5. Проверить наличие собранных образов командой `docker images`

```
REPOSITORY TAG IMAGE ID CREATED SIZE
tarantool/tarantool 2 27f80564ecce 2 minutes ago 298MB
```

6. Создать в клонированной директории `docker` файл `docker-compose.yml` со следующим содержимым:

```
version: "3"
services:
  tarantool1:
    image: tarantool/tarantool:2
    container_name: tarantool1
    restart: always
    networks:
      - tarantool_network
    ports:
      - "3301:3301"
    volumes:
      - tarantool1_volume:/opt/tarantool
    environment:
      # https://github.com/tarantool/docker#environment-variables
      TARANTOOL_REPLICATION: "tarantool1,tarantool2"
```

```

tarantool2:
image: tarantool/tarantool:2
container_name: tarantool2
restart: always
networks:
- tarantool_network
ports:
- "3302:3301"
volumes:
- tarantool2_volume:/opt/tarantool
environment:
TARANTOOL_REPLICATION: "tarantool1,tarantool2"
volumes:
tarantool1_volume:
tarantool2_volume:
networks:
tarantool_network:
driver: bridge

```

1. Перейти в директорию *docker*, где располагается файл `docker-compose.yml`

Например:

```
cd ~/direct
```

2. Пример запуска мастер-мастер репликации:

<https://github.com/tarantool/docker#start-a-master-master-replica-set>

3. Поднять Tarantool командой:

```
docker-compose up
```

4. Проверить командой:

```
docker ps
```

В списке запущенных образов должны присутствовать `tarantool1` и `tarantool2`

## 4.18 Установка CSV-Uploader

Описание настроек модуля приведено в «Руководстве администратора».

Действия по установке выполняются через SSH консоль технологического пользователя.

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`. Пример конфигурации файла `application.yml` и возможные настройки конфигурации модуля см. в разделе [Конфигурация CSV-uploader \(application.yml\)](#) Руководства администратора.

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/home/dir
```

где,

- **file.jar** - название JAR-файла;
- **user\_name** - имя пользователя, например, **sudo** или **root**;
- **IP** - адрес сервера;
- **/home/dir** - директория на сервере, в которую будет загружен файл.

## 4.19 Установка REST-адаптера

Установка сервисов и необходимых сервисных баз данных.

Внимание:

Установка данных сервисов выполняется после установки «Core Services».

Действия по установке выполняются на сервере ADCM через SSH-консоль технологического пользователя.

### 4.19.1 Установка docker-образов

Установка *Docker* описана в разделе «Установка Arenadata Cluster Manager (ADCM)» [Установка Arenadata Cluster Manager \(ADCM\)](#). Установить образ REST-адаптера, при этом файл образа **.tar** должен быть загружен через *Docker* из дистрибутива программы.

Пример команды для установки:

```
docker image load -i <image_file>
```

### 4.19.2 Подготовка конфигурации

Для подготовки скриптов необходимо выполнить следующие действия:

1. Отредактировать переменные в файле **application.yml** для рабочей среды. Файл необходимо поместить в директорию **config**:

- Указать корректный путь до целевой БД. Переменная **jdbc\_url**.

Например:

```
jdbc:{drv_name}://{IP_Host}:{Port}/{db_name}
```

, где:

- **drv\_name** - имя драйвера , берется из описания драйвера базы
- **IP\_Host** - ip или имя хоста с базой;
- **Port** - порт, на котором будет слушать сервис . По умолчанию 8080;
- **db\_name** - имя базы данных.
- Указать **driver-class-name** - имя класса , берется из описания драйвера базы
- Указать переменную **file\_path** - наименование файла с описанием запросов, по умолчанию **sample.yaml**.



- Указать переменные `execquery` - сопоставление `operationId` из файла с описанием запросов с файлом обработчиком, по умолчанию `sample.peb`.

### 4.19.3 Процесс установки

Для установки необходимо выполнить следующие действия:

1. Перейти в директорию с файлом `docker-compose.yml`

Например:

```
cd ~/direct
```

2. Поднять сервис конечных точек командой:

```
docker-compose up -d
```

3. Проверить установку следующей командой:

```
docker ps
```

В списке запущенных образов должен присутствовать `rest-adapter`

## 4.20 Установка Counter-provider

Описание настроек модуля приведено в «Руководстве администратора».

Действия по установке выполняются через SSH консоль технологического пользователя.

Общий процесс установки состоит из следующих действий:

1. Настроить конфигурацию модуля.
2. Создать на сервере директорию для загрузки файлов модуля.
3. Загрузить файлы модуля в созданную директорию.
4. Запустить модуль.
5. Проверить установку модуля.

Настройка конфигурации выполняется путем редактирования параметров файла `application.yml`. Пример конфигурации файла `application.yml` и возможные настройки конфигурации модуля см. в разделе [Конфигурация модуля Counter-Provider \(application.yml\)](#) Руководства администратора.

Для загрузки файла на сервер выполните команду:

```
scp file.jar user_name@IP:/home/dir
```

где,

- `file.jar` - название JAR-файла;
- `user_name` - имя пользователя, например, `sudo` или `root`;
- `IP` - адрес сервера;
- `/home/dir` - директория на сервере, в которую будет загружен файл.

## 4.21 Установка коннектора Kafka-Postgres

1. Скопировать файлы:

- kafka-postgres-writer-0.3.0.jar,
- kafka-postgres-reader-0.3.0.jar,
- kafka-postgres-avro-0.3.0.jar

из дистрибутива и загрузить в папку *kafka-postgres-connector*.

2. Скопировать конфигурационные файлы KAFKA-POSTGRES-WRITER

application.yaml и KAFKA-POSTGRES-READER application.yaml в папку *kafka-postgres-connector/config*

Конфигурационный файл KAFKA-POSTGRES-WRITER application.yaml:

```
logging:
level:
  ru.datamart.kafka: ${LOG_LEVEL:DEBUG}
  org.apache.kafka: ${KAFKA_LOG_LEVEL:INFO}

http:
port: ${SERVER_PORT:8096}

vertx:
pools:
  eventLoopPoolSize: ${VERTX_EVENT_LOOP_SIZE:12}
  workersPoolSize: ${VERTX_WORKERS_POOL_SIZE:32}
verticle:
  query:
    instances: ${QUERY_VERTICLE_INSTANCES:12}
    insert:
      poolSize: ${INSERT_WORKER_POOL_SIZE:32}
      insertPeriodMs: ${INSERT_PERIOD_MS:1000}
      batchSize: ${INSERT_BATCH_SIZE:500}
    consumer:
      poolSize: ${KAFKA_CONSUMER_WORKER_POOL_SIZE:32}
      maxFetchSize: ${KAFKA_CONSUMER_MAX_FETCH_SIZE:10000}
    commit:
      poolSize: ${KAFKA_COMMIT_WORKER_POOL_SIZE:1}
      commitPeriodMs: ${KAFKA_COMMIT_WORKER_COMMIT_PERIOD_MS:1000}

client:
kafka:
  consumer:
    checkingTimeoutMs: ${KAFKA_CHECKING_TIMEOUT_MS:10000}
    responseTimeoutMs: ${KAFKA_RESPONSE_TIMEOUT_MS:10000}
    consumerSize: ${KAFKA_CONSUMER_SIZE:10}
    closeConsumersTimeout: ${KAFKA_CLOSE_CONSUMER_TIMEOUT:15000}
  property:
    bootstrap.servers: ${KAFKA_BOOTSTRAP_SERVERS:kafka.host:9092}
    group.id: ${KAFKA_CONSUMER_GROUP_ID:postgres-query-execution}
    auto.offset.reset: ${KAFKA_AUTO_OFFSET_RESET:earliest}
    enable.auto.commit: ${KAFKA_AUTO_COMMIT:false}
    auto.commit.interval.ms: ${KAFKA_AUTO_INTERVAL_MS:1000}

datasource:
```

```

postgres:
  database: ${POSTGRES_DB_NAME:test}
  user: ${POSTGRES_USERNAME:dtm}
  password: ${POSTGRES_PASS:dtm}
  hosts: ${POSTGRES_HOSTS:localhost:5432}
  poolSize: ${POSTGRES_POOLSIZE:10}
  preparedStatementsCacheMaxSize: ${POSTGRES_CACHE_MAX_SIZE:256}
  preparedStatementsCacheSqlLimit: ${POSTGRES_CACHE_SQL_LIMIT:2048}
  preparedStatementsCache: ${POSTGRES_CACHE:true}

```

Конфигурационный файл KAFKA-POSTGRES-READER application.yaml:

```

logging:
  level:
    ru.datamart.kafka: ${LOG_LEVEL:DEBUG}
    org.apache.kafka: ${KAFKA_LOG_LEVEL:INFO}

http:
  port: ${SERVER_PORT:8094}

vertx:
  pools:
    eventLoopPoolSize: ${VERTX_EVENT_LOOP_SIZE:12}
    workersPoolSize: ${VERTX_WORKERS_POOL_SIZE:32}
  verticle:
    query:
      instances: ${QUERY_VERTICLE_INSTANCES:12}

datasource:
  postgres:
    database: ${POSTGRES_DB_NAME:test}
    user: ${POSTGRES_USERNAME:dtm}
    password: ${POSTGRES_PASS:dtm}
    hosts: ${POSTGRES_HOSTS:localhost:5432}
    poolSize: ${POSTGRES_POOLSIZE:10}
    preparedStatementsCacheMaxSize: ${POSTGRES_CACHE_MAX_SIZE:256}
    preparedStatementsCacheSqlLimit: ${POSTGRES_CACHE_SQL_LIMIT:2048}
    preparedStatementsCache: ${POSTGRES_CACHE:true}
    fetchSize: ${POSTGRES_FETCH_SIZE:1000}

kafka:
  client:
    property:
      key.serializer: org.apache.kafka.common.serialization.ByteArraySerializer
      value.serializer: org.apache.kafka.common.serialization.ByteArraySerializer

```

## 4.22 Установка Arenadata Cluster Manager (ADCM)

Внимание:

При условии установки CentOS 7.9

*Arenadata Cluster Manager (ADCM)* – сервер, с которого будет централизованно производиться установка почти всех компонентов программы. Данный компонент обязателен для установки.

*Arenadata Cluster Manager (ADCM)* поставляется в docker-контейнерах, поэтому чтобы установить ADCM на сервере должен быть установлен *Docker* или совместимое программное обеспечение для работы с контейнерами. Текущая версия программного обеспечения несовместима с SELinux.

Подробная инструкция по установке Arenadata Cluster Manager (ADCM) приведена в официальной документации разработчика (<https://docs.arenadata.io/adcm/user/install.html>).

Установка Arenadata Cluster Manager (ADCM) осуществляется пользователем согласно следующим этапам:

1. Выключить **selinux**. Для этого в файле `/etc/selinux/config` установите значение **SELINUX=disabled** и перезапустить сервер командой **reboot now**.
2. Установите *Docker* для этого используйте следующие команды:

```
yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
yum install -y containerd.io
yum install -y docker-ce-3:18.09.1-3.el7
systemctl enable docker
systemctl start docker
```

3. Установить ADCM в *Docker*. Для этого используйте следующие команды:

```
docker pull arenadata/adcm:latest
docker create --name adcm -p 8000:8000 -v /opt/adcm:/adcm/data
arenadata/adcm:latest
```

4. Запустите ADCM командой:

```
docker start adcm
```

5. Убедитесь, что ADCM установлен корректно, для этого перейдите в браузере по адресу `http: // <ip_of_your_server>`.

В случае корректной установки откроется окно «Авторизация» (см. [Рисунок - 4.2](#) ).

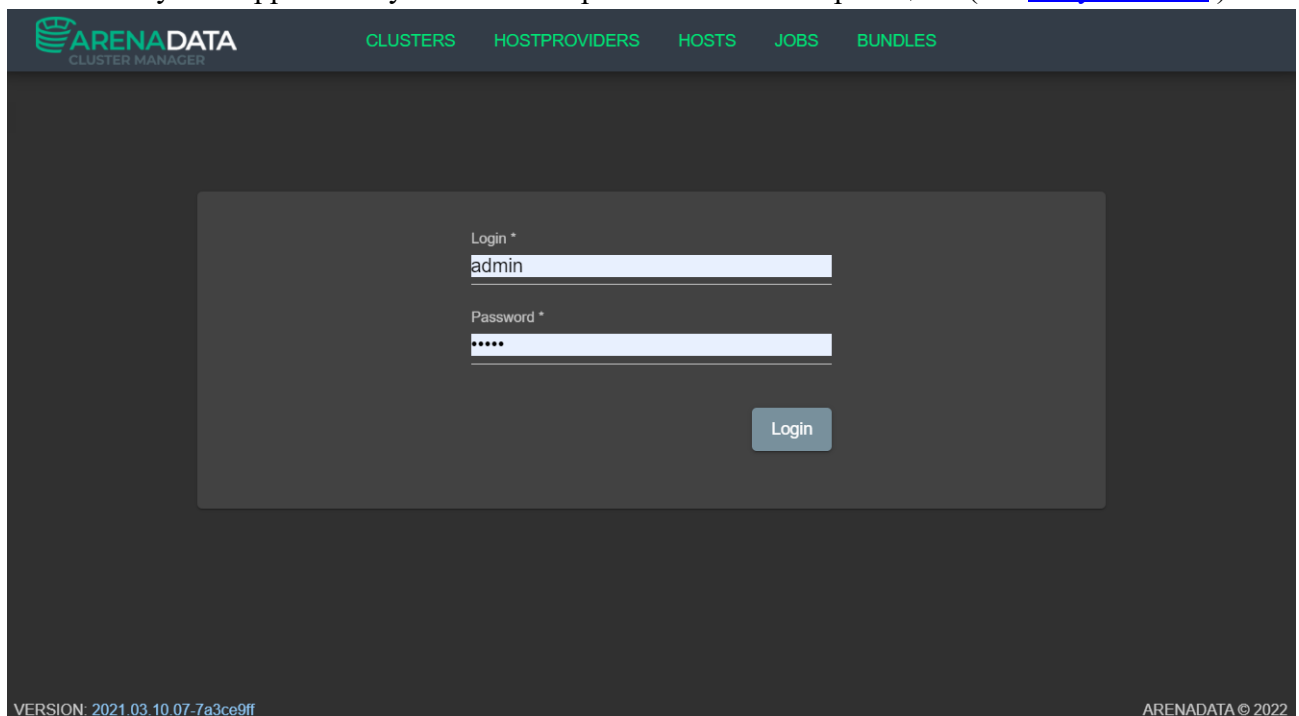


Рисунок - 4.2 Окно «Авторизация»

Для авторизации используйте следующие данные:

- пользователь - **admin**;
- пароль - **admin**.

После завершения процедуры авторизации откроется главное окно программы Arenadata Cluster Manager (см. [Рисунок - 4.3](#)).

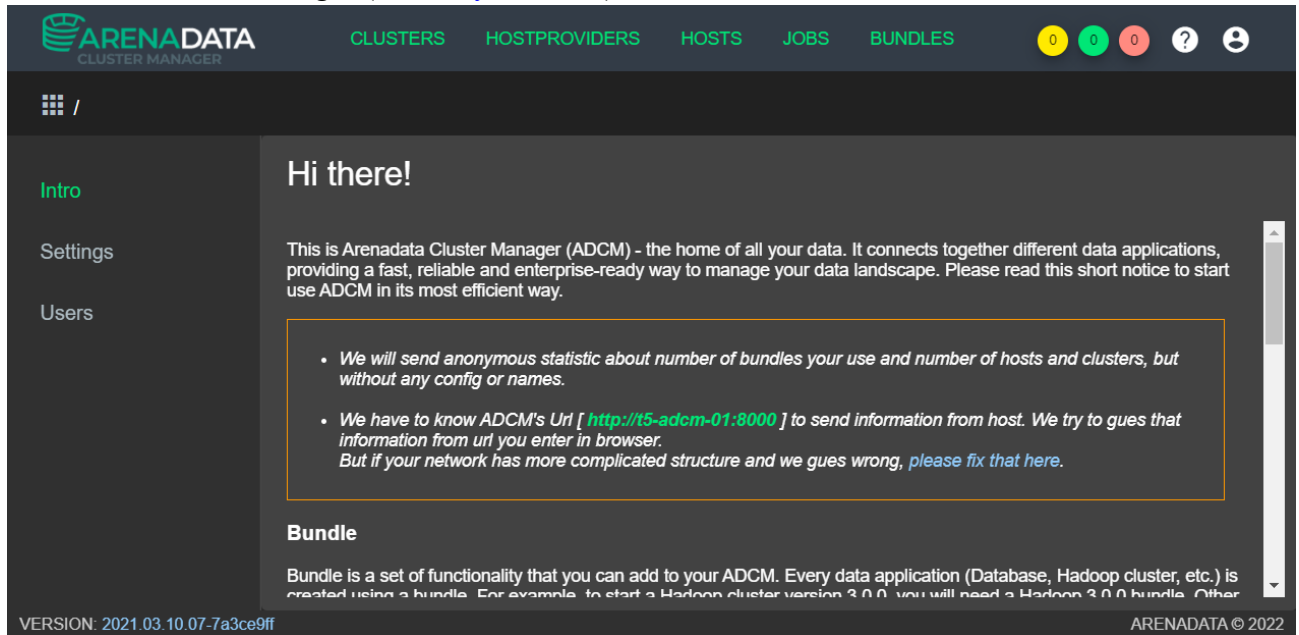


Рисунок - 4.3 Главное окно программы «Arenadata Cluster Manager»

## 4.23 Установка Arenadata Streaming (ADS)

Вниманис:

При условии установки CentOS 7.9

Arenadata Streaming (ADS) устанавливается на кластере с помощью ADCM из загруженного установочного пакета (бандла).

Скачать установочный пакет (версия: **ads\_v1.6.0.0-1**) можно с сайта разработчика: [https://store.arenadata.io/#products/arenadata\\_streaming](https://store.arenadata.io/#products/arenadata_streaming).

Подробная инструкция по установке Arenadata Streaming (ADS) приведена в официальной документации разработчика (<https://docs.arenadata.io/DTM/Install/ADS.html> , <https://docs.arenadata.io/ads/Install/index.html> ). В данной инструкции описывается установка кластера Kafka и импорт уже установленного кластера *Zookeeper*. Установка *Zookeeper* на кластер приведена в инструкции по установке ADQM.

1. Для создания кластера необходимо иметь предварительно загруженный бандл (версия: **ads\_v1.6.0.0-1**). В графическом интерфейсе ADCM перейти в раздел *CLUSTERS* -> *Create cluster*. Создать кластер (см. [Рисунок - 4.4](#)).

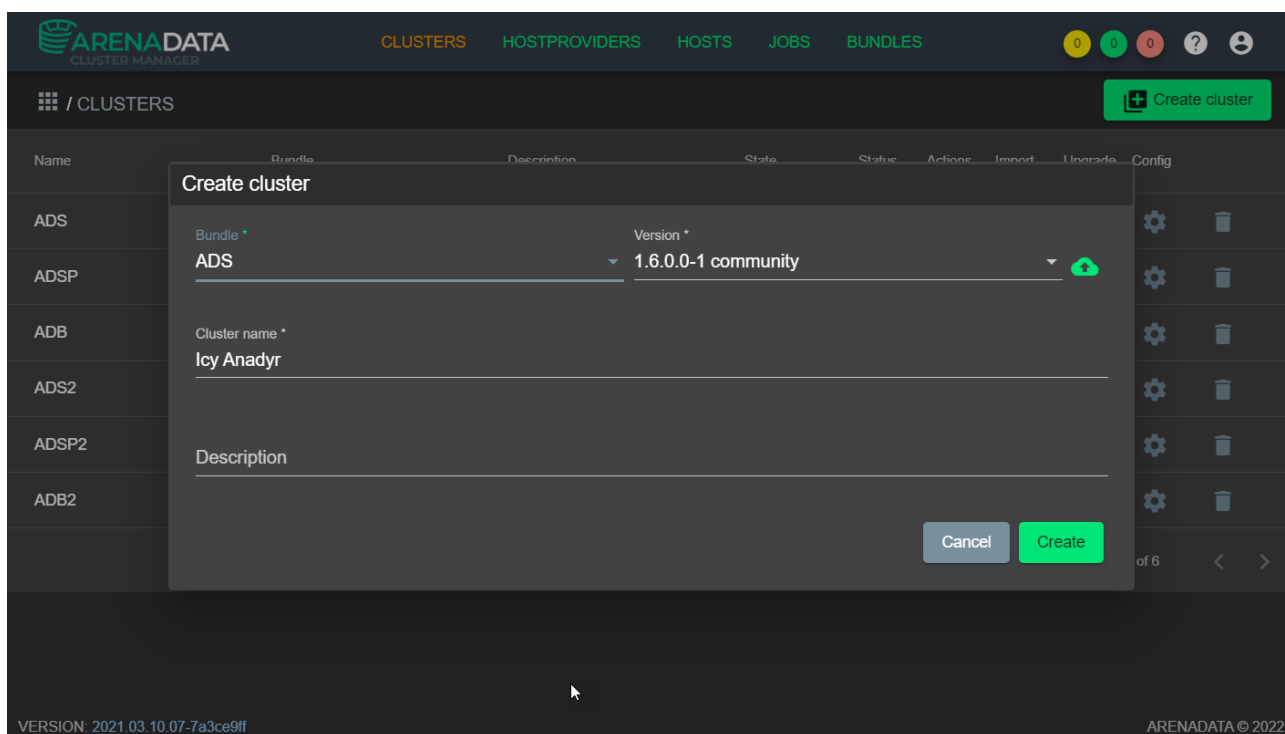


Рисунок - 4.4 Создание кластера ADS через графический интерфейс ADCM

1. В разделе **Services** графического интерфейса ADCM необходимо добавить сервисы в созданный кластер (см. [Рисунок - 4.5](#)). Версии сервисов могут отличаться от указанных на рисунке.

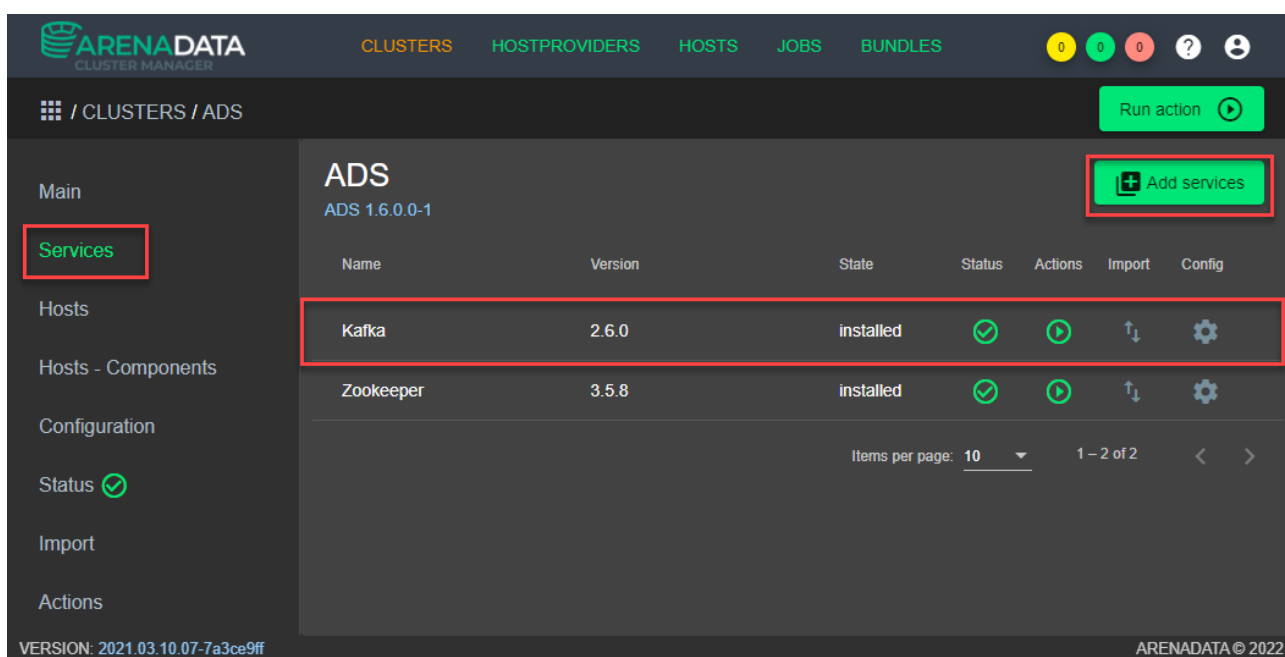


Рисунок - 4.5 Добавление через графический интерфейс ADCM сервисов в созданный кластер

3. В случае, если *Kafka* и *Zookeeper* разворачивается в одном кластере, необходимо добавить сервис *Zookeeper*.
4. В разделе *Hosts* графического интерфейса ADCM указать серверы созданного кластера,

на которых будет развёрнуто ПО Arenadata Streaming (ADS) (см. [Рисунок - 4.6](#)).

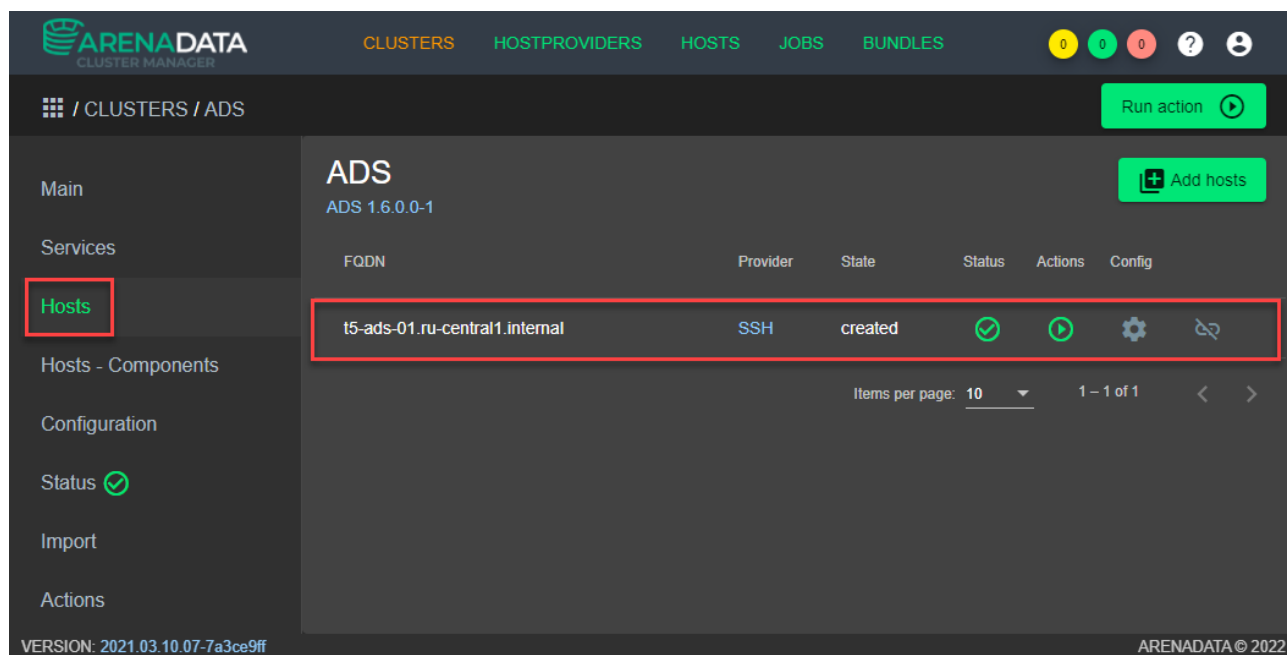


Рисунок - 4.6 Выбор через графический интерфейс ADCM серверов для развёртывания кластера ADS

5. В разделе *Hosts - Components* указать три узла (ноды) *Kafka* (см [Рисунок - 4.7](#)).

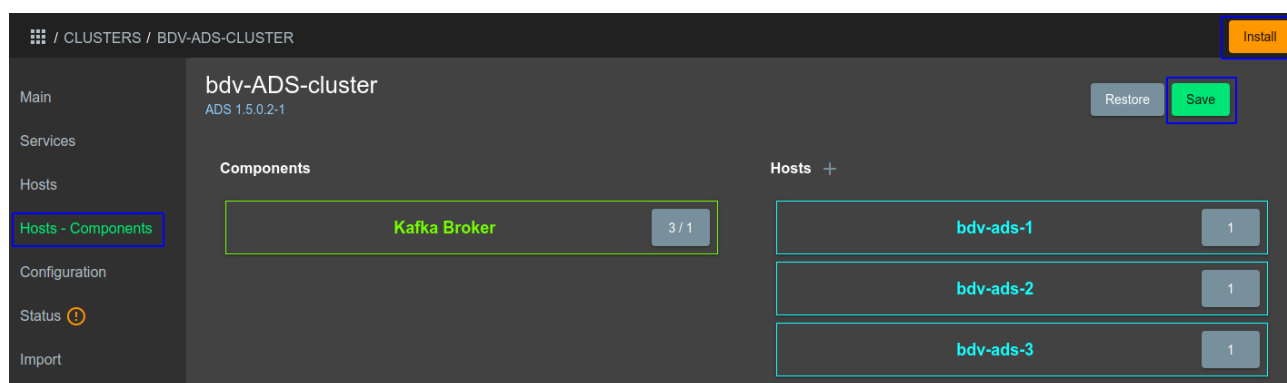


Рисунок - 4.7 Выбор через графический интерфейс ADCM узлов Kafka

6. Указать кластер *Zookeeper* в разделе **Import**, в случае если *Zookeeper* находится на другом кластере (см [Рисунок - 4.8](#)).

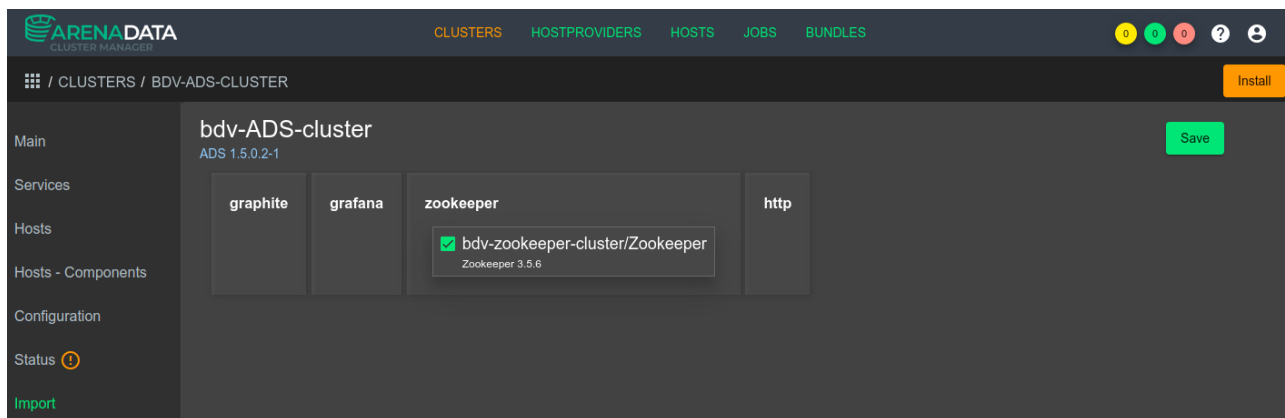


Рисунок - 4.8 Выбор через графический интерфейс ADCM кластера *Zookeeper* для импорта

7. Нажмите кнопку **Install**, чтобы запустить процесс установки. В окне «Дополнительные параметры» нажмите кнопку **Run**, чтобы выполнить установку (см [Рисунок - 4.9](#)).



Рисунок - 4.9 Запуск через графический интерфейс ADCM процесса установки

8. Проверьте, что установка завершена, для этого:

- Проверьте лог-файлы *Zookeeper* по относительному пути

```
/var/log/zookeeper/.
```

- Проверьте лог-файл *Kafka* по относительному пути

```
/var/log/kafka/.
```

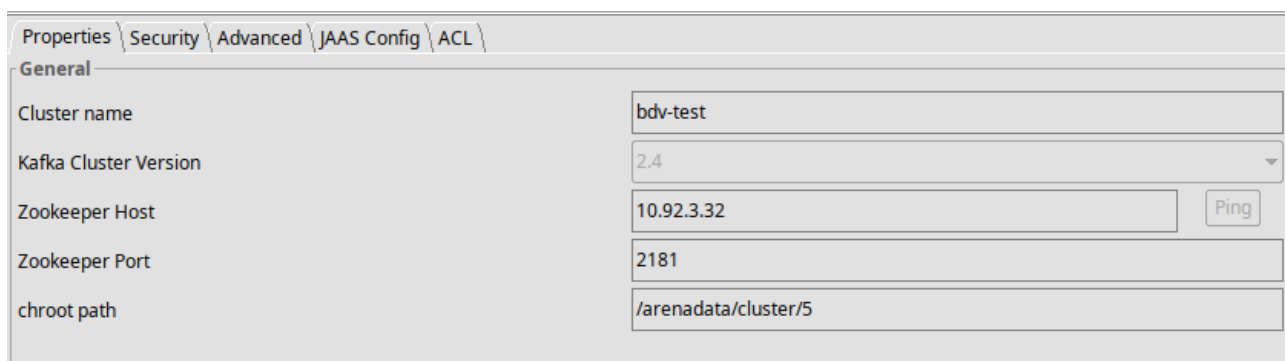
- Проверьте настройки подключения к кластеру *Zookeeper* с помощью сторонней утилиты *KafkaTool* (см [Рисунок - 4.10](#)).
- Проверьте корректность разрешения имен серверов для автоматического определения bootstrap-серверов *Kafka*. При отсутствии DNS-сервера добавьте в локальный **hosts** адреса *Kafka*.
- Определите путь **chroot path**, для этого выполните на любом сервере *Zookeeper* команду:

```
/usr/lib/zookeeper/bin/zkCli.sh
```



Далее, выполните команду:

```
ls /arenadata/cluster.
```



Properties \ Security \ Advanced \ JAAS Config \ ACL \	
General	
Cluster name	bdv-test
Kafka Cluster Version	2.4
Zookeeper Host	10.92.3.32 <input type="button" value="Ping"/>
Zookeeper Port	2181
chroot path	/arenadata/cluster/5

Рисунок - 4.10 Проверка через графический интерфейс ADCM настроек подключения к кластеру Zookeeper

При проверке, в данном руководстве, использовался графический интерфейс приложения Offset Explorer 2.1 (<https://www.kafkatool.com/download.html>).

## 5 ПРОВЕРКА ПРОГРАММЫ

### 5.1 Проверка Arenadata Cluster Manager (ADCM)

Внимание:

При условии установки CentOS 7.9

Arenadata Cluster Manager (ADCM), при успешной установке, должен быть доступен по адресу [http://<ip\\_address\\_of\\_server>:8000](http://<ip_address_of_server>:8000).

Для авторизации используйте следующие данные:

- логин: **admin**;
- пароль: **admin**.

### 5.2 Проверка Arenadata Streaming (ADS)

Внимание:

При условии установки CentOS 7.9

#### 5.2.1 Проверка сервиса Zookeeper

Проверка сервиса *Zookeeper* осуществляется через графический пользовательский интерфейс Arenadata Cluster Manager (ADCM). Чтобы выполнить проверку, выполните следующие действия:

1. Выберите кластер ADS, для этого откройте вкладку *Cluster-ADB*.
2. На вкладке **Services** для сервиса *Zookeeper*, в поле **Actions** нажмите кнопку **Run action** и выберите **Check**.

#### 5.2.2 Проверка сервиса Apache Kafka

Проверка сервиса *Apache Kafka* осуществляется через графический пользовательский интерфейс Arenadata Cluster Manager (ADCM). Чтобы выполнить проверку, выполните следующие действия:

1. Выберите кластер ADS, для этого откройте вкладку *Cluster-ADB*.
2. На вкладке **Services** для сервиса Apache Kafka, в поле **Actions** нажмите кнопку **Run action** и выберите **Check**.

### 5.3 Проверка ProStore

Проверка ПО [ProStore](#) осуществляется путём отправки SQL-запросов к [ProStore](#) через клиентское [JDBC](#)-подключение и сопоставления ожидаемого эталонного и полученного результатов.

Проверка осуществляется согласно следующим этапам:

1. Создать Витрину в [ProStore](#) с помощью SQL-запроса:

```
CREATE DATABASE <имя несуществующей логической базы>, например,  
CREATE DATABASE testdb;
```

2. Создать таблицу в [ProStore](#) со всеми типами колонок с помощью SQL-запроса:

```
CREATE TABLE <имя логической базы из п.1>.all_types (  
- id int not null,  
- double_col double,  
- float_col float,  
- char_col varchar(36),  
- boolean_col boolean,  
- int_col int not null,  
- bigint_col bigint,  
- date_col date,  
- timestamp_col timestamp,  
- primary key (id)  
- )  
- distributed by (id)
```

3. Проверить существование и структуру созданной таблицы в [ProStore](#) с помощью SQL-запросов:

```
select ¥* from <имя логической базы из п.1>.all_types  
DATASOURCE_TYPE='ADG'  
  
select ¥* from <имя логической базы из п.1>.all_types  
DATASOURCE_TYPE='ADQM'  
  
select ¥* from <имя логической базы из п.1>.all_types  
DATASOURCE_TYPE='ADB'
```

4. Удалить таблицу со всеми типами колонок из [ProStore](#) с помощью SQL-запроса:

```
DROP TABLE <имя логической базы из п.1>.all_types
```

5. Удалить Витрину с помощью SQL-запроса:

```
DROP DATABASE <имя логической базы из п.1>.
```

#### Внимание:

Наличие сообщений об ошибках, а также отличие получаемых состояний ProStore на различных этапах проверки от ожидаемых состояний является индикатором неуспешного прохождения проверки.

## 5.4 Проверка СМЭВ QL Сервера

### 5.4.1 Проверки и валидации

Валидации запускаются либо на все объекты данного типа (указать **all**), либо только на указанные, в том числе через запятую.

Доступность источников проверяется командой:

```
./smevql test source <all | source-name>
```

Валидность моделей проверяется командой:

```
./smevql test model <all | model-name>
```

## 5.5 Проверка СМЭВ3-адаптера

Для проверки модуля **СМЭВ3-адаптер** необходимо выполнить запрос к сервису.

Пример запроса

```
curl -s IP:Port/metrics | grep '^liveness '
```

где,

- **IP** - адрес сервера.
- **Port** - адрес сервера.
- **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

Пример успешного ответа

```
liveness 1.0
```

Ответ ``1`` означает, что модуль работает.

## 5.6 Проверка ПОДД-адаптера - Модуль исполнения запросов

Для проверки модуля **ПОДД-адаптер - Модуль исполнения запросов** необходимо выполнить запрос к сервису.

Пример запроса

```
curl -s IP:Port/metrics | grep '^liveness '
```

где, - **IP** - адрес сервера. - **Port** - адрес сервера. - **liveness** - параметр проверки работоспособности модуля.

Например:

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

Пример успешного ответа

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

## 5.7 Проверка ПОДД-адаптер – Модуль MPPR

Для проверки **ПОДД-адаптер - Модуль MPPR** необходимо выполнить запрос к сервису.

Пример запроса

```
curl -s IP:Port/metrics | grep '^liveness '
```

где, - **IP** - адрес сервера. - **Port** - адрес сервера. - **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

#### Пример успешного ответа

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

### 5.8 Проверка ПОДД-адаптер-Модуль MPPW

Для проверки модуля **ПОДД-адаптер - Модуль MPPW** необходимо выполнить запрос к сервису.

#### Пример запроса

```
curl -s IP:Port/metrics | grep '^liveness '
```

где, - **IP** - адрес сервера. - **Port** - адрес сервера. - **liveness** - параметр проверки работоспособности модуля.

Например:

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

#### Пример успешного ответа

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

### 5.9 Проверка ПОДД-адаптер – Модуль импорта данных табличных параметров

Для проверки модуля **ПОДД-адаптер – Модуль импорта данных табличных параметров** необходимо выполнить запрос к сервису.

#### Пример запроса

```
curl -s IP:Port/metrics | grep '^liveness '
```

где, - **IP** - адрес сервера. - **Port** - адрес сервера. - **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

#### Пример успешного ответа

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

### 5.10 Проверка ПОДД-адаптер – Модуль группировки данных табличных параметров

Для проверки модуля **ПОДД-адаптер – Модуль импорта данных табличных параметров** необходимо выполнить запрос к сервису.

#### Пример запроса

```
curl -s IP:Port/metrics | grep '^liveness '
```

где, - **IP** - адрес сервера. - **Port** - адрес сервера. - **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

**Пример успешного ответа**

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

## 5.11 Проверка ПОДД-адаптер – ПОДД-адаптер – Wrapper

Для проверки модуля **ПОДД-адаптер - Wrapper** необходимо выполнить запрос к сервису.

**Пример запроса**

```
curl -s IP:Port/metrics | grep '^liveness '
```

где,

- **IP** - адрес сервера.
- **Port** - адрес сервера.
- **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

**Пример успешного ответа**

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

## 5.12 Проверка модуля группировки чанков репликации

Для проверки **Модуля группировки чанков репликации** необходимо выполнить запрос к сервису.

**Пример запроса**

```
curl -s IP:Port/metrics | grep '^liveness '
```

где, - **IP** - адрес сервера. - **Port** - адрес сервера. - **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

**Пример успешного ответа**

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

## 5.13 Проверка DATA-uploader – Модуль исполнения асинхронных заданий

Для проверки модуля **DATA-Uploader** необходимо выполнить запрос к сервису.

### Пример запроса

```
curl -s IP:Port/metrics | grep '^liveness '
```

где, - **IP** - адрес сервера. - **Port** - адрес сервера. - **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

### Пример успешного ответа

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

## 5.14 Проверка REST-uploader – Модуль асинхронной загрузки данных из сторонних источников

Для проверки модуля **REST-uploader** необходимо выполнить запрос к сервису.

### Пример запроса

```
curl -s IP:Port/metrics | grep '^liveness '
```

где, - **IP** - адрес сервера. - **Port** - адрес сервера. - **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

### Пример успешного ответа

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

## 5.15 Проверка ПОДД-адаптер – Модуль подписки

Для проверки модуля **ПОДД-адаптер - Модуль подписок** необходимо выполнить запрос к сервису.

### Пример запроса

```
curl -s IP:Port/metrics | grep '^liveness '
```

где, - **IP** - адрес сервера. - **Port** - адрес сервера. - **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

### Пример успешного ответа

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

## 5.16 Проверка BLOB-адаптер

Для проверки модуля **Blob-адаптер** необходимо выполнить запрос к сервису.

Пример запроса:

```
curl -s IP:Port/metrics | grep '^liveness '
```

где,

- **IP** - адрес сервера.
- **Port** - адрес сервера.
- **liveness** - параметр проверки работоспособности модуля.

Например:

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

Пример успешного ответа:

```
liveness 1.0
```

Ответ ``1`` означает, что модуль работает.

## 5.17 Проверка Сервиса формирования документов

Для проверки модуля **Сервис Формирования документов** необходимо выполнить запрос к сервису.

Пример запроса

```
curl -s IP:Port/metrics | grep '^liveness '
```

где,

- **IP** - адрес сервера.
- **Port** - адрес сервера.
- **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

Пример успешного ответа

```
liveness 1.0
```

Ответ ``1`` означает, что модуль работает.

## 5.18 Проверка ETL

### 5.18.1 Проверка Apache Airflow

Проверка сервиса **Apache Airflow** осуществляется через графический пользовательский интерфейс, в случае успешной установки, он должен быть доступен по адресу <http://localhost:8080>.

Для авторизации используйте следующие данные:

- логин: **airflow**;
- пароль: **airflow**.

Также, можно проверить удалённое подключение с помощью http-запроса, для этого



выполните следующую команду:

```
ENDPOINT_URL="http://localhost:8080/"
curl -X GET ¥¥
--user "airflow:airflow" ¥¥
"${ENDPOINT_URL}/api/v1/pools"
```

### 5.18.2 Проверка Apache Spark

Проверка сервиса *Apache Spark* осуществляется через графический пользовательский интерфейс, в случае успешной установки, он должен быть доступен по адресу:

Веб-сервер *Spark Master* доступен по адресу:

```
http://<ваш ip-адрес>:8080 например http://localhost:8080/
```

Веб-сервер *Spark Worker 1* доступен по адресу:

```
http://<ваш ip-адрес>:8081 например http://localhost:8081/
```

### 5.18.3 Проверка Apache Hadoop

Проверить корректность работы [Apache Hadoop](#) можно командой:

```
make wordcount
```

или последовательно выполнить следующие команды:

```
docker build -t hadoop-wordcount ./submit

docker run --network docker-hadoop_default --env-file hadoop.env
bde2020/hadoop-base:master hdfs dfs -mkdir -p /input/

docker run --network docker-hadoop_default --env-file hadoop.env
bde2020/hadoop-base:master hdfs dfs -copyFromLocal -f
/opt/hadoop-3.2.1/README.txt /input/

docker run --network docker-hadoop_default --env-file hadoop.env
hadoop-wordcount

docker run --network docker-hadoop_default --env-file hadoop.env
bde2020/hadoop-base:master hdfs dfs -cat /output/¥*

docker run --network docker-hadoop_default --env-file hadoop.env
bde2020/hadoop-base:master hdfs dfs -rm -r /output

docker run --network docker-hadoop_default --env-file hadoop.env
bde2020/hadoop-base:master hdfs dfs -rm -r /input
```

После запуска [Apache Hadoop](#) можно зайти в следующие WEB-интерфейсы:

```
Namenode:
http://<dockerhadoop_IP_address>:9870/dfshealth.html#tab-overview

History server: http://<dockerhadoop_IP_address>:8188/applicationhistory

Datanode: http://<dockerhadoop_IP_address>:9864/

Nodemanager: http://<dockerhadoop_IP_address>:8042/node

Resource manager: http://<dockerhadoop_IP_address>:8088/
```

### 5.18.4 Проверка Tarantool(Vinyl)

Проверка СУБД [Tarantool](https://www.tarantool.io/en/doc/latest/book/monitoring/) осуществляется согласно документации на СУБД Tarantool: <https://www.tarantool.io/en/doc/latest/book/monitoring/> .

### 5.19 Проверка REST-адаптер

Проверить удалённое подключение с помощью http-запроса:

```
curl localhost:8080
```

В случае успешной установки ответ будет следующим:

```
{"timestamp":"2021-03-15T10:22:57.325+0000","status":404,"error":"Not Found","message":"","path":"/"}%
```

### 5.20 Проверка Counter-provider - Сервиса генерации уникального номера

Для проверки Сервиса генерации уникального номера необходимо выполнить запрос к сервису.

**Пример запроса**

```
curl -s IP:Port/metrics | grep '^liveness '
```

где, - **IP** - адрес сервера. - **Port** - адрес сервера. - **liveness** - параметр проверки работоспособности модуля.

Например

```
curl -s http://172.16.10.67:9837/metrics | grep '^liveness '
```

**Пример успешного ответа**

```
liveness 1.0
```

Ответ **1** означает, что модуль работает.

## 6 ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

### **ADCM**

Arenadata Cluster Manager (ADCM) — Универсальный оркестратор гибридного ландшафта. Он позволяет быстро устанавливать, настраивать все data-сервисы компании и управлять ими. Наиболее ярко преимущества ADCM раскрываются при работе с гетерогенной инфраструктурой, при которой появляется возможность размещать data-сервисы на различных типах инфраструктур: в облаке, on-premise или в качестве PaaS-сервисов.

### **ADS**

Arenadata Streaming (ADS) - Масштабируемая отказоустойчивая система для потоковой обработки данных в режиме реального времени на базе Apache Kafka и Apache Nifi.

### **Airflow**

открытое программное обеспечение для создания, выполнения, мониторинга и оркестровки потоков операций по обработке данных.

### **Apache**

Организация-фонд, способствующая развитию проектов программного обеспечения Apache.

### **Apache Airflow**

Платформа для программного создания, планирования и мониторинга рабочих процессов.

### **Apache Hadoop**

Свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределённых программ, работающих на кластерах из сотен и тысяч узлов.

### **Apache Spark**

Фреймворк с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных.

### **Apache Kafka**

Распределённый программный брокер сообщений, проект с открытым исходным кодом, разрабатываемый в рамках фонда Apache.

### **Apache Avro**

Система сериализации данных, разработанная в рамках проекта Hadoop.

### **API**

Application programming interface (англ.) – описание сервисов взаимодействия компьютерной программы с другими программами.

### **Avro**

(Object Container File) Линейно-ориентированный формат хранения файлов Big Data

### **BLOB-адаптер**

Информационно-технологический компонент Витрины, обеспечивающий чтение бинарных файлов из Хранилище BLOB-объектов ведомства.

### **ClickHouse**

Колоночная аналитическая СУБД с открытым кодом, позволяющая выполнять аналитические запросы в режиме реального времени на структурированных больших данных, разрабатываемая компанией Яндекс.

### **Docker**

Программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений.

### **Docker Compose**

Платформа контейнеризации, предназначена для конфигурирования многоконтейнерных приложений. В Docker Compose можно управлять несколькими контейнерами [Docker](https://docs.docker.com/compose/).

### **DATA-uploader**

Модуль исполнения асинхронных заданий.

## **Counter-Provider**

Сервис генерации уникального номера.

## **CSV**

Comma-Separated Values (англ.) – значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных.

## **CSV-extractor**

Специализированное программное обеспечение, которое извлекает данные из csv-файлов в собственную БД-хранилища сервиса ([Tarantool](#))

## **CSV-Uploader**

Программный модуль Витрины данных, который предназначен для загрузки csv-файлов в Витрину данных.

## **Grafana**

Веб-приложение для аналитики и интерактивной визуализации показателей мониторинга с открытым исходным кодом.

## **Greenplum**

Массово-параллельная СУБД для хранилищ данных на основе PostgreSQL.

## **DAG**

Файл, содержащий блок данных.

## **DBeaver**

Клиентское приложение для управления базами данных (БД), которое использует программный интерфейс [JDBC](#) для взаимодействия с реляционными БД через драйвер [JDBC-драйвер](#).

## **DDL**

Data definition language (англ.) – семейство компьютерных языков, используемых в компьютерных программах для описания структуры баз данных.

## **DNS**

Domain Name System «система доменных имён» — компьютерная распределённая система для получения информации о доменах. Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства), получения информации о маршрутизации почты и/или обслуживающих узлах для протоколов в домене.

## **Endpoint**

Шлюз (в переводе с англ. — конечная точка), который соединяет серверные процессы приложения с внешним интерфейсом. Простыми словами, это адрес, на который отправляются сообщения (работает с API)

## **ETL**

Extract, transform, load (англ.) – решение, используемое при выгрузке данных из различных источников ведомств и дальнейшего хранения их в Витрине [ProStore](#) для чтения, использования и взаимодействия с другими ведомствами.

## **FileZilla**

FTP-клиент

## **Greenplum**

Система управления данными из мира big data.

## **HikariCP**

Hikari Connection Pool.

## **HTTP**

HyperText Transfer Protocol (англ.) – протокол прикладного уровня передачи данных, в настоящий момент используется для передачи произвольных данных.

## **IAM**

Сервисы управления идентификацией и контролем доступа (Identity&AccessManagement)

## **JDBC**

Java DataBase connectivity (англ.) – платформенно-независимый промышленный стандарт взаимодействия Java-приложений с различными [СУБД](#).

## **JDBC-драйвер**

Специализированное программное обеспечение, которое размещается на стороне системы, использующей ADTM (клиента ADTM). Драйвер предоставляет JDBC-интерфейс подключения из этой системы к ADTM и взаимодействует с сервисом исполнения запросов по REST API, предоставляемым сервисом исполнения запросов.

## **JDBC-extractor**

Специализированное программное обеспечение, которое извлекает данные из jdbc-источника (ведомства) в собственную БД-хранилища сервиса ([Tarantool](#)).

## **JDBC-CSV-transformer**

Специализированное программное обеспечение, которое предназначено для подключения к БД по JDBC, с последующим сохранением в csv-файлы.

## **Kafka**

Распределённый программный брокер сообщений, проект с открытым исходным кодом, разрабатываемый в рамках фонда Apache.

## **Kafka-loader**

Специализированное программное обеспечение, которое загружает данные, извлеченные и приведенные в соответствие логической структуре данных Витрины, собственно в Витрину.

## **Loki**

Приложение для агрегирования log-файлов, используется совместно с [Prometheus](#).

## **MD5**

128-битный алгоритм хеширования. Предназначен для создания «отпечатков» или дайджестов сообщения произвольной длины и последующей проверки их подлинности.

## **MPP**

Массово-параллельная архитектура (англ. massive parallel processing, MPP, также «массивно-параллельная архитектура»).

## **NTP**

Network Time Protocol — сетевой протокол для синхронизации внутренних часов компьютера с использованием сетей с переменной латентностью.

## **OpenAPI**

The OpenAPI Specification (англ.) – формализованная спецификация и экосистема множества инструментов, предоставляющая интерфейс между front-end системами, кодом библиотек низкого уровня и коммерческими решениями в виде API.

## **ProStore**

Интеграционная система, обеспечивающая единый интерфейс к хранилищу разнородных данных. Определяет структуры данных, запись и чтение данных Витрины. Позволяет работать со входящими в состав хранилища СУБД одинаковым образом, используя единый синтаксис запросов SQL и единую логическую схему данных.

## **Prostore**

Ядро интеграционной системы ProStore, состоящее из сервиса исполнения запросов и сервиса мониторинга.

## **Prometheus**

Программное приложение, используемое для мониторинга событий и оповещения, которое записывает метрики в реальном времени в базу данных временных рядов, построенную с использованием модели HTTP-запроса, с гибкими запросами и оповещениями в режиме реального времени.

## **Proxy API**

Проксирование запросов через Datamart Studio к инсталляциям приложений Витрин данных

**PSQL**

Терминальный клиент для работы с PostgreSQL

**PuTTY**

Свободно распространяемый клиент для различных протоколов удалённого доступа, включая SSH, Telnet, rlogin.

**PXF**

Фреймворк, позволяющий ADB (Greenplum) параллельно обмениваться данными со сторонними системами.

**REST**

Representational state transfer (англ.) – архитектурный стиль взаимодействия компонентов распределенного приложения в сети.

**REST-адаптер**

Сервис, реализующий публикацию конечных точек API для обработки запросов с использованием спецификации OpenAPI версии 3. Используется для сохранения обратной совместимости получения данных из ведомства по REST.

**REST API**

Набор правил, по которым различные программы могут взаимодействовать между собой и обмениваться данными с помощью протокола HTTP

**REST-Uploader**

Модуль асинхронной загрузки данных из сторонних источников.

**SOAP**

(от англ. Simple Object Access Protocol — простой протокол доступа к объектам) — протокол обмена структурированными сообщениями в распределённой вычислительной среде.

**SQL**

Structured query language (англ.) – язык структурированных запросов. Декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

**SQL-запрос**

Запрос к Базе данных.

**SSH**

Secure Shell (англ.) – «безопасная оболочка». Сетевой протокол прикладного уровня, позволяющий производить удалённое управление операционной системой и туннелирование TCP-соединений.

**Tarantool**

Платформа in-memory вычислений с гибкой схемой данных для создания высоконагруженных приложений. Включает в себя базу данных и сервер приложений на Lua.

**UDP**

Протокол передачи данных. С UDP компьютерные приложения могут посылать сообщения другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных.

**URI**

Унифицированный идентификатор ресурса. URI — последовательность символов, идентифицирующая абстрактный или физический ресурс.

**UUID**

Стандарт идентификации, используемый в создании программного обеспечения, стандартизированный Open Software Foundation как часть DCE — среды распределённых вычислений. Основное назначение UUID — это позволить распределённым системам уникально идентифицировать информацию без центра координации.

**Vert.x**

Библиотека для разработки асинхронных приложений, основанная на событиях.

**VipNet**

программное обеспечение (далее - ПО) для защиты сетевого трафика на рабочих местах пользователей.

**XML**

eXtensible Markup Language (англ.) – универсальный текстовый формат для хранения и передачи структурированных данных.

**XML-extractor**

Специализированное программное обеспечение, для копирования данных из xml-файлов в собственную БД-хранилища сервиса ([Tarantool](#)).

**ZooKeeper**

Сервер с открытым исходным кодом для высоконадежной распределенной координации облачных приложений.

**Агент ПОДД**

Типовое программное обеспечение, устанавливаемое на стороне УВ и обеспечивающее сопряжение Витрин, хранилищ реплик, ИС Участника взаимодействия с ПОДД. В частности, чтение данных из Витрины, запись данных в реплику, обработка промежуточных/временных массивов данных, порождаемых в процессе выполнения распределённых запросов.

**База данных**

Совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

**Брокер сообщений**

Архитектурный паттерн в распределённых системах; приложение, которое преобразует сообщение по одному протоколу от приложения-источника в сообщение протокола приложения-приёмника, тем самым выступая между ними посредником.

**Витрина данных**

Комплекс программных и технических средств в составе информационно-телекоммуникационной инфраструктуры участника НСУД, предназначенный для формирования и (или) получения данных с использованием среды взаимодействия НСУД.

**ВС**

Вид сведений

**ГОСТ**

Нормативно-правовой документ, в соответствии требованиями которого производится стандартизация производственных процессов

**Дельта**

Логически целостная совокупность изменений информации об объектах. Каждой дельте поставлено в соответствие целое число из монотонно возрастающей последовательности целых чисел начиная с 0, отражающее ее место в общей последовательности дельт и дата-время ее исполнения.

**ЕИП**

Единая информационная платформа

**ИС**

Информационная система.

**КриптоПро**

Разработанная одноименной компанией линейка криптографических утилит (вспомогательных программ) — так называемых криптопровайдеров. Они используются в других программах для генерации электронной подписи (ЭП), работы с сертификатами, организации структуры РКИ и т.д.

**Логическая модель данных**

Схема базы данных, выраженная в понятиях бизнес-требований.

**набор данных**

Совокупность данных (датасетов), систематизированных в определённом формате, представляющих собой базовый элемент для работы с данными во многих отраслях

## **НСУД**

Национальная система управления данными.

## **ОГРН**

Основной государственный регистрационный номер, который налоговая служба присваивает юридическим лицам сразу же после регистрации

## **ПО**

Программное обеспечение.

## **ПОДД**

Подсистемы обеспечения доступа к данным

## **ПОДД-адаптер**

Программно-технический продукт, обеспечивающий взаимодействие витрины и ПОДД СМЭВ.

## **ПОДД-адаптер - Модуль исполнения запросов**

Логический модуль ПОДД-адаптера, предназначен для исполнения запросов ПОДД СМЭВ (через протокол коммуникации Агент ПОДД).

## **ПОДД-адаптер - Модуль MPPR**

Логический модуль ПОДД-адаптера, предназначен для чтения данных в многопоточном режиме (massively parallel processing, [MPP](#)).

## **ПОДД-адаптер - Модуль MPPW**

Логический модуль ПОДД-адаптера выполняет загрузку данных в многопоточном режиме.

## **Поставщик данных**

Организация, осуществляющая передачу государственных данных в НСУД.

## **Потребитель данных**

Организация, осуществляющая использование государственных данных, содержащихся в НСУД.

## **Реплика**

СУБД, хранящая реплицируемые наборы данных, полученные от Поставщика данных.

## **Сервис Формирования документов**

Модуль витрины, предназначенный для работы с формируемыми документами.

## **СМЭВ3**

Система межведомственного электронного взаимодействия.

## **СМЭВ3-адаптер**

Информационно-технологический компонент СМЭВ, устанавливаемый на стороне Участника взаимодействия. СМЭВ3-адаптер обеспечивает информационное взаимодействие через единый электронный сервис единой системы межведомственного электронного взаимодействия (СМЭВ).

## **Сообщение**

Сведения в виде законченного блока данных, передаваемые при функционировании информационной системы.

## **СУБД**

Система управления базами данных

## **Токен**

Ключ безопасности (Цифровой сертификат)

## **Участник НСУД**

Федеральный орган исполнительной власти, иной орган государственной власти, государственный внебюджетный фонд, орган местного самоуправления, иное юридическое лицо, являющиеся стороной действующего соглашения о присоединении к Национальной системе управления данными.

## **ФЛК**

Форматно-логический контроль загружаемых в Витрину данных.



### **Хранилище BLOB-объектов**

Место для хранения BLOB-объектов (бинарных данных). Располагается на стороне ведомства и не является частью Витрины данных. Взаимодействие с Хранилищем BLOB-объектов осуществляется через [BLOB-адаптер](#).

### **Хранилище S3 (объектное хранилище S3)**

Хранилище бинарных объектов, позволяющее хранить файлы любого типа и объема. Доступ к хранилищу предоставляется через API.