|  |  |  |
| --- | --- | --- |
| УТВЕРЖДАЮДолжность (полное наименование) АО «Цифровые платформы и решения Умного Города» |  |  |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ФИО/« » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г. |  |  |

Программа для ЭВМ

**«Инфраструктурная IoT платформа», версия 1.3**

**Описание ПО**

RU.1197746546282.00017-01 13 01

**Аннотация**

Настоящий документ является описанием программы для ЭВМ «Инфраструктурная IoT платформа», версия 1.3 (далее – IoT платформа).

Документ разработан в соответствии с требованиями ГОСТ Р 59795–2021 «Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Требования к содержанию документов».

**СОДЕРЖАНИЕ**

[1 ОБЩИЕ СВЕДЕНИЯ 4](#_Toc132909495)

[1.1 Обозначения и сокращения 4](#_Toc132909496)

[1.2 Технология и архитектура платформы 4](#_Toc132909497)

[1.3 Структурные уровни платформы 5](#_Toc132909498)

[2 Базовые компоненты Платформы 6](#_Toc132909499)

[2.1 Компонентный состав ПО 6](#_Toc132909500)

[2.2 Ядро IoT 6](#_Toc132909501)

[2.2.1 Средства моделирования 7](#_Toc132909502)

[2.2.2 Системные возможности 7](#_Toc132909503)

[2.2.3 Шлюз 12](#_Toc132909504)

[2.3 PostgreSQL 13](#_Toc132909505)

[2.3.1 Путь запроса 13](#_Toc132909506)

[2.3.2 Установка соединений 14](#_Toc132909507)

[2.3.3 Этап разбора 14](#_Toc132909508)

[2.4 Планировщик и оптимизатор 15](#_Toc132909509)

[2.5 pgAdmin 17](#_Toc132909510)

[2.6 Apache Pulsar 17](#_Toc132909511)

[2.7 BIRT 17](#_Toc132909512)

[2.7.1 Дизайн 17](#_Toc132909513)

[2.7.2 Анатомия отчета 18](#_Toc132909514)

[2.7.3 Компоненты BIRT 18](#_Toc132909515)

[3 Версии используемого программного обеспечения. 20](#_Toc132909516)

[4 Требования к аппаратному обеспечению 21](#_Toc132909517)

[5 Перечень необходимого ПО или общесистемных пакетов 21](#_Toc132909518)

[5.1 Состав дистрибутива 22](#_Toc132909519)

[6 Соответствие ПЭВМ требованиям реестра Российского программного обеспечения 22](#_Toc132909520)

# ОБЩИЕ СВЕДЕНИЯ

Программа для ЭВМ «Инфраструктурная IoT платформа», версия 1.3» (далее – IoT платформа) предназначена для подключения, управления, централизованного наблюдения за параметрами устройств, подключенных к ней по стандартным промышленным протоколам и протоколам интернета вещей, поддерживает как облачные, так и локальные развертывания.

## Обозначения и сокращения

|  |  |
| --- | --- |
| Термин(сокращение) | Определение |
| Интернет вещей, IoT | Концепция сети передачи данных между физическими объектами («вещами»), оснащёнными встроенными средствами и технологиями для взаимодействия друг с другом или с внешней средой |
| Инфраструктурная IoT платформа | Программно-аппаратный комплекс интернета вещей, предназначенный для работы в качестве решения в области автоматизированных систем управления технологическими процессами |
| Платформа, Система | ПЭВМ «Инфраструктурная IoT платформа» |
| ПО  | Программное обеспечение для ЭВМ «Инфраструктурная IoT платформа», версия 1.3 |
| API | Программный интерфейс приложения, интерфейс прикладного программирования |
| IoT  | Интернет вещей. Концепция сети передачи данных между физическими объектами («вещами»), оснащёнными встроенными средствами и технологиями для взаимодействия друг с другом или с внешней средой. |
| Open-source | Программное обеспечение с открытым исходным кодом. |
| SSO | Технология единого входа, метод аутентификации, который позволяет пользователям безопасно аутентифицироваться сразу в нескольких приложениях и сайтах, используя один набор учетных данных. |

## Технология и архитектура платформы

С технологической точки зрения Платформа является программно-аппаратным комплексом интернета вещей для сбора, обработки, визуализации данных и управления устройствами. Платформа обеспечивает подключение устройств по стандартным промышленным протоколам и протоколам интернета вещей, поддерживает как облачные, так и локальные развертывания.

Архитектура IoT платформы состоит условно из логических слоев:

* слой ядра. Основу ядра составляет программный комплекс, отвечающий за вызов отдельных модулей (подсистем), обмен данными между различными подсистемами, управление потоками данных, преобразование и представление (визуализация) данных;
* слой модулей, реализующих функционал IoT платформы. Основная масса расчётных и служебных функций, поддерживаемых IoT платформой, реализована в виде отдельных модулей, которые взаимодействуют с ядром и имеют единый пользовательский интерфейс.

Уровни структурной организации IoT платформы:

* уровень сбора данных - реализуется посредством «Программного модуля «Промышленный IoT шлюз» (входит в состав IoT платформы в качестве сервера сбора данных) и ПЛК, работающих в зоне исполнительных устройств.
* уровень передачи данных – реализация потокового транспорта, позволяющего осуществлять передачу данных.
* уровень хранения данных – выполнение сжатия и обработки данных, хранение в СУБД.
* уровень обработки данных (SCADA) – уровень организации логики работы и отображения информации о работе инженерных систем.
* уровень управления интеграцией и безопасностью – механизмы встраивания в инфраструктуру заказчика и обеспечения безопасности.

## Структурные уровни платформы

Уровни структурной организации Платформы:

1. Уровень сбора данных - реализуется посредством Шлюза (сервер сбора данных), и ПЛК, работающих в зоне исполнительных устройств. Реализуется управления в режиме реального времени.
2. Уровень передачи данных – реализация потокового транспорта, позволяющего осуществлять передачу данных.
3. Уровень хранения данных – выполнение сжатия и обработки данных, хранение в СУБД.
4. Уровень обработки данных (SCADA) – уровень организации логики работы и отображения информации о работе инженерных систем.
5. Уровень управления интеграцией и безопасностью – механизмы встраивания в инфраструктуру заказчика и обеспечения безопасности.

# Базовые компоненты Платформы

## Компонентный состав ПО

Компонентный состав программного обеспечения Платформы приведен в таблице ниже (см. Таблица 1).

Таблица – Компонентный состав программного обеспечения Платформы

|  |  |  |
| --- | --- | --- |
| № п.п. | Компонент | Описание |
| 1 | Ядро IoT | Реализовано на основе продукта с открытым исходным кодом для сбора, обработки, визуализации данных и управления устройствами. Устройства подключаются по стандартным протоколам IoT (BACnet, Modbus, SNMP, KNX и другие) |
| 2 | БД Postgres SQL TimescaleDB | Платформа использует базу данных для хранения сущностей (устройств, активов, клиентов, информационных панелей и т. д.) и данных телеметрии (атрибутов, показаний датчиков временных рядов, статистики, событий).TimescaleDB — это расширение PostgreSQL для работы с временными рядами (time series) |
| 4 | Apache Pulsar | Служба (брокер) внутреннего обмена сообщениями (управление очередями) |
| 5 | BIRT | Инструмент для составления отчетов (по концепции WYSIWYG) |
| 6 | Шлюз | Шлюз позволяет подключать устройства, которые находятся в локальной сети и не имеют доступа к Интернету или используют определенные протоколы, отличные от IP. Основные задачи модуля - мониторинг и управление в режиме реального времени технологическим оборудованием, устройствами автоматики, контроллерами и т.д. |
| 7 | Redis | Система управления базами данных класса NoSQL с открытым исходным кодом, работающая со структурами данных типа «ключ — значение». Используется как для баз данных, так и для реализации кэшей, брокеров сообщений |
| 8 | Сommunication-service | Сервис отправки CМС сообщений собственной разработки |
| 9 | MinIO | Программное обеспечение с открытым исходным кодом для распределенного хранения объектов, совместимое с S3. Применяется для хранения файлов |
| 10 | Nginx | HTTP-сервер и обратный прокси-сервер с открытым исходным кодом, почтовый прокси-сервер, а также TCP/UDP прокси-сервер общего назначения |

## Ядро IoT

Ядро IoT - платформа Интернета вещей с открытым исходным кодом, которая обеспечивает быструю разработку, управление и масштабирование проектов Интернета вещей.

Характеристики Ядра:

* масштабируемость: горизонтально масштабируемая платформа, построенная с использованием ведущих технологий с открытым исходным кодом;
* отказоустойчивость: отсутствие единой точки отказа, все узлы в кластере идентичны;
* надежность и эффективность: один серверный узел может обрабатывать десятки или даже сотни тысяч устройств в зависимости от сценария использования. Кластер может обрабатывать миллионы устройств;
* долговечность: гарантируется сохранность данных. Поддерживаются различные реализации очередей, чтобы обеспечить чрезвычайно высокую надежность сообщений.
* настраиваемость: легко добавлять новые функции с помощью настраиваемых виджетов и узлов механизма правил.

### Средства моделирования

#### Физическая модель

Модель физического представления реальности включает:

* Территории, на которых находятся и перемещаются Сотрудники, Жители и Транспортные средства. Последние размещаются в Зданиях, состоящих из Помещений.
* Территория, сотрудник, житель, транспортное средство, здание, помещение - оснащаются датчиками-сенсорами, а также средствами телемеханического управления.

#### Логическая модель

Платформа добавляет логический уровень к физической модели. Платформа предоставляет средства управления несколькими типами сущностей и их отношениями.

Примеры сущностей: Арендаторы, Клиенты, Пользователи, Устройства, Активы и др.

#### Средства описания предметной области

Помимо физического и логического представлений, платформа позволяет описать предметную область:

* формировать и управлять инженерной топологией сетей передачи ресурсов и сервисов;
* добавляет географический слой для контроля территории инженерных сооружений;
* позволяет формировать единое представление для зданий и помещений.

### Системные возможности

#### Обзор сущностей

Платформа предоставляет средства управления несколькими типами сущностей и их отношениями в приложениях. Поддерживаемые объекты:

* Арендаторы - можно рассматривать арендатора как отдельный бизнес-объект: это физическое лицо или организация, которые владеют или производят устройства и активы; У арендатора может быть несколько пользователей-администраторов арендатора и миллионы клиентов, устройств и активов;
* Клиенты - клиент также является отдельным бизнес-субъектом: физическое лицо или организация, которые покупают или используют арендаторские устройства и / или активы; у клиента может быть несколько пользователей и миллионы устройств и / или активов;
* Пользователи - пользователи могут просматривать информационные панели и управлять объектами;
* Устройства - основные объекты Интернета вещей, которые могут создавать данные телеметрии и обрабатывать команды RPC. Например, датчики, исполнительные механизмы, переключатели;
* Активы - абстрактные объекты Интернета вещей, которые могут быть связаны с другими устройствами и активами. Например, завод, поле, или автомобиль;
* Представления - полезны, если нужно поделиться с клиентами только частью данных об устройстве или активах;
* Тревоги - события, которые выявляют проблемы с активами, устройствами или другими объектами;
* Панели мониторинга - визуализация данных IoT и возможность управления конкретными устройствами через пользовательский интерфейс;
* Узел правил - блоки обработки входящих сообщений, событий жизненного цикла сущностей и т. д .;
* Цепочка правил - определяет поток обработки в механизме правил. Может содержать множество узлов правил и ссылок на другие цепочки правил.

Каждая сущность поддерживает:

* Атрибуты - статические и полустатические пары "ключ-значение", связанные с сущностями. Например, серийный номер, модель, версия прошивки;
* Данные временных рядов - точки данных временных рядов, доступные для хранения, запросов и визуализации. Например, температура, влажность, уровень заряда батареи;
* Отношения - направленные связи с другими сущностями. Например, содержит, управляет, владеет, производит.

Некоторые организации поддерживают профили:

* Профили арендаторов - содержат общие настройки для нескольких арендаторов: сущность, API, ограничения скорости и т. Д. Каждый арендатор имеет один-единственный профиль в определенный момент времени.
* Профили устройств - содержат общие настройки для нескольких устройств: конфигурация обработки и транспортировки и т. д. Каждое устройство имеет один-единственный профиль в определенный момент времени.

#### Связи

Отношение сущностей определяет связь между двумя сущностями платформы, принадлежащими одному и тому же арендатору. Отношение имеет произвольный тип: содержит, управляет, поддерживает и т. д. Отношение также является направленным.

#### Устройства

Устройства представляют собой базовые объекты Интернета вещей, которые могут создавать данные телеметрии и обрабатывать команды RPC. Например, датчики, исполнительные механизмы, переключатели.

Платформа поддерживает следующие функции управления устройствами:

* с помощью веб-интерфейса;
* удаленные вызовы(REST API).

Каждое устройство может одновременно входить в несколько групп.

На платформе поддерживаются следующие функции для работы с устройствами:

* добавление нового устройства;
* редактирование устройства;
* удаление устройства.

#### Атрибуты устройства

Атрибуты - это статические и полустатические пары "ключ-значение", связанные с устройствами. Например, серийный номер, модель, версия прошивки.

#### Телеметрия устройства

Точки данных временных рядов доступны для хранения, запросов и визуализации. Например, температура, влажность или уровень заряда батареи.

#### Сигналы устройства

Тревоги - это события, которые выявляют проблемы с устройствами.

#### События устройства

События помогают отслеживать сообщения, чтобы узнать, что произошло с активом.

#### Отношения устройств

Отношения представляют собой направленные связи с другими сущностями.

#### Журналы аудита устройств

Платформа предоставляет возможность отслеживать действия пользователя с записью в журнал аудита. Можно регистрировать действия пользователя, связанные с основными объектами: активами, устройствами, дашбордом, правилами и т. д.

#### Настройки профиля устройств

Администратор может настраивать общие параметры для нескольких устройств с помощью профилей устройств. Каждое устройство имеет один-единственный профиль в определенный момент времени.

#### Работа с атрибутами устройства

Платформа предоставляет возможность назначать пользовательские атрибуты объектам и управлять этими атрибутами. Эти атрибуты хранятся в базе данных и могут использоваться для визуализации данных и обработки данных.

Атрибуты рассматриваются как пары "ключ-значение". Гибкость и простота формата «ключ-значение» позволяют легко и без проблем интегрироваться практически с любым устройством IoT на рынке. Ключ всегда является строкой и в основном является именем атрибута, тогда как значение атрибута может быть строковым, логическим, двойным, целым числом или JSON.

#### Имена атрибутов

Пользователь платформы может определить любое имя атрибута. Рекомендуется использовать camelCase. Это упрощает написание пользовательских JS-функций для обработки и визуализации данных.

Примечание: сamelCase - стиль написания составных слов, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово внутри фразы пишется с прописной буквы.

#### Типы атрибутов

Есть три типа атрибутов, которые будут описаны ниже.

#### Атрибуты на стороне сервера

Этот тип атрибута поддерживается практически любым объектом платформы: устройством, активом, клиентом, арендатором, пользователем и т. д. Атрибуты на стороне сервера - это те, которые можно настроить через пользовательский интерфейс администрирования или REST API. Прошивка устройства не может получить доступ к атрибуту на стороне сервера.

#### Общие атрибуты

Этот тип атрибутов доступен только для устройств. Он похож на серверные атрибуты, но имеет одно важное отличие. Микропрограммное обеспечение / приложение устройства может запрашивать значение совместно используемого атрибута (ов) или подписываться на обновления атрибута (ов). Устройства, которые обмениваются данными через MQTT или другие протоколы двунаправленной связи, могут подписаться на обновления атрибутов и получать уведомления в режиме реального времени. Устройства, которые обмениваются данными через HTTP или другие протоколы связи типа «запрос-ответ», могут периодически запрашивать значение общего атрибута.

#### Интерфейсы ядра IoT

На рисунке ниже показана схема ядра IoT и предоставляемые им интерфейсы.



Рисунок 2. Схема интерфейсов ядра IoT

#### Механизм правил

Механизм правил платформы является сердцем системы и отвечает за обработку входящих сообщений.

Механизм правил подписывается на поток входящих данных из очередей и подтверждает сообщение только после его обработки. Доступно несколько стратегий, которые управляют порядком обработки сообщений и критериями подтверждения сообщений.

Механизм правил платформы может работать в двух режимах: общий и изолированный. В общем режиме механизм правил обрабатывает сообщения, принадлежащие нескольким клиентам. В изолированном режиме механизм правил может быть настроен для обработки сообщений только для определенного клиента.

#### Веб-интерфейс

Платформа предоставляет компонент, написанный с использованием инфраструктуры Express.js, для размещения статического содержимого веб-интерфейса. Статический веб-интерфейс содержит пакет приложений. После загрузки приложение начинает использовать REST API и WebSockets API, предоставляемые платформой.

#### Очереди сообщений

Платформа включает службу обмена сообщениями Apache Pulsar 2.8.0. Использование устойчивых и масштабируемых очередей позволяет платформе противостоять пиковым нагрузкам и выполнять балансировку нагрузки.

#### Работа с данными телеметрии

Платформа предоставляет богатый набор функций, связанных с данными временных рядов:

* Сбор данных с устройств с использованием различных протоколов и интеграций;
* Хранение данных временных рядов в базах данных SQL;
* Запрос последних значений данных временных рядов или всех данных в указанном временном диапазоне с гибкой агрегацией;
* Подписка на обновления данных с помощью WebSockets для визуализации или аналитики в реальном времени;
* Визуализация данных временных рядов с помощью настраиваемых виджетов и панелей мониторинга;
* Фильтрацию и анализ данных с помощью гибкой системы правил;
* Создание сигналов тревоги на основе собранных данных;
* Перенаправление данных во внешние системы с помощью внешних узлов правил.

#### Точки данных

Платформа внутренне обрабатывает данные временных рядов как пары ключ-значение с отметками времени. Мы называем единственную пару ключ-значение с меткого времени точкой данных. Гибкость и простота формата «ключ-значение» позволяют легко и без проблем интегрироваться практически с любым устройством IoT на рынке. Ключ всегда является строкой и в основном представляет собой имя ключа точки данных, а значение может быть строковым, логическим, двойным, целым числом или JSON.

#### Встроенные транспортные протоколы

Реализации встроенного транспортного протокола применимы для устройств, которые обмениваются данными по этим протоколам и могут напрямую подключаться к платформе.

Например:

* MQTT API;
* CoAP API;
* HTTP API.

Большинство вышеперечисленных протоколов поддерживают JSON, Protobuf или собственный формат данных. Это лучший вариант для новых устройств, когда есть контроль над прошивкой.

#### Работа с панелями мониторинга Интернета вещей

Платформа предоставляет возможность создавать информационные панели и управлять ими.

Каждая панель управления может содержать множество виджетов. Панели мониторинга отображают данные от многих объектов: устройств, активов и т. д. Панели мониторинга могут быть назначены клиентам.

### Шлюз

Шлюз позволяет подключать устройства, которые находятся в локальной сети и не имеют доступа к Интернету или используют определенные протоколы, отличные от IP.

Шлюз поддерживает MQTT, OPC-UA, Modbus (TCP/RTU), BLE, HTTP, CAN, BACnet/IP, ODBC, SNMP, KNX-IP, PROFINET и другие протоколы. Шлюз преобразует данные с устройств во внутренний формат платформы и загружает их через MQTT на платформу.

## PostgreSQL

СУБД PostgreSQL — это объектно-реляционная система управления базами данных (ОРСУБД, ORDBMS), поддерживающая большую часть стандарта SQL и предлагающая множество современных функций:

* сложные запросы;
* внешние ключи;
* триггеры;
* изменяемые представления;
* транзакционная целостность;
* многоверсионность.

Кроме того, пользователи могут расширять возможности PostgreSQL путем создания:

* типов данных;
* функций;
* операторов;
* агрегатных функций;
* методов индексирования;
* процедурных языков.

PostgreSQL работает в качестве сервера в системе клиент-сервер. Клиент отправляет серверу запрос и получает отклик. Запросы PostgreSQL производятся с помощью языка SQL, откликами обычно являются таблицы данных из базы данных.

PostgreSQL реализован в архитектуре клиент-сервер. Рабочий сеанс PostgreSQL включает следующие взаимодействующие процессы (программы):

* главный серверный процесс, управляющий файлами баз данных, принимающий подключения клиентских приложений и выполняющий различные запросы клиентов к базам данных. Эта программа сервера БД называется postgres;
* клиентское приложение пользователя для осуществления операций в базе данных.

Как и в других типичных клиент-серверных приложениях, клиент и сервер могут располагаться на разных компьютерах. В этом случае они взаимодействуют по сети TCP/IP. Важно понимать, что файлы, доступные на клиентском компьютере, могут быть недоступны (или доступны только под другим именем) на компьютере-сервере.

Сервер PostgreSQL может обслуживать одновременно несколько подключений клиентов. Для этого он запускает («порождает») отдельный процесс для каждого подключения. Другими словами, клиент и серверный процесс взаимодействуют, не затрагивая главный процесс postgres. Таким образом, главный серверный процесс всегда работает и ожидает подключения клиентов, принимая которые, он организует взаимодействие клиента и отдельного серверного процесса.

### Путь запроса

Приложение устанавливает подключение к PostgreSQL. Эта программа передает запрос на сервер и ждет от него результатов.

На этапе разбора запроса сервер выполняет синтаксическую проверку запроса, переданного прикладной программой, и создает дерево запроса.

Система правил принимает дерево запроса, созданное на стадии разбора, и ищет в системных каталогах правила для применения к этому дереву. Обнаружив подходящие правила, она выполняет преобразования, заданные в теле правил.

Одно из применений системы правил заключается в реализации представлений. Когда выполняется запрос к представлению (т. е. виртуальной таблице), система правил преобразует запрос пользователя в запрос, обращающийся не к представлению, а к базовым таблицам из определения представления.

Планировщик/оптимизатор принимает дерево запроса (возможно, переписанное) и создает план запроса, который будет передан исполнителю.

Он выбирает план, сначала рассматривая все возможные варианты получения одного и того же результата. Например, если для обрабатываемого отношения создан индекс, прочитать отношение можно двумя способами. Во-первых, можно выполнить простое последовательное сканирование, а во-вторых, можно использовать индекс. Затем оценивается стоимость каждого варианта и выбирается самый дешевый. Затем выбранный вариант разворачивается в полноценный план, который сможет использовать исполнитель.

Исполнитель рекурсивно проходит по дереву плана и получает строки тем способом, который указан в плане. Он сканирует отношения, обращаясь к системе хранения, выполняет сортировку и соединения, вычисляет условия фильтра и, наконец, возвращает полученные строки.

### Установка соединений

PostgreSQL реализует простую клиент-серверную модель по схеме «процесс для пользователя». В такой схеме один клиентский процесс подключается к одному отдельному серверному процессу. Так как заранее неизвестно, сколько подключений будет, нужен главный процесс, который будет запускать новый процесс при каждом запросе подключения. Главный процесс называется postgres и принимает входящие подключения в заданном порту TCP/IP. Получив запрос на подключение, процесс postgres порождает новый серверный процесс. Серверные задачи взаимодействуют между собой через семафоры и разделяемую память, чтобы обеспечить целостность данных при одновременном обращении к ним.

Многие клиенты базируются на библиотеке libpq для языка C.

Установив подключение, клиентский процесс передает запрос серверу. Запрос передается в обычном текстовом виде, клиент не занимается его анализом. Сервер разбирает запрос, строит план выполнения, выполняет его и возвращает полученные строки клиенту, передавая их через установленное подключение.

### Этап разбора

Этап разбора разделяется на две части:

* разбор, алгоритм которого описан в gram.y и scan.l, а программный код генерируется инструментами Unix bison и flex;
* преобразование, в процессе которого модифицируются и дополняются структуры данных, полученные после разбора запроса.

#### Разбор

При разборе проверяется сначала синтаксис строки запроса (поступающей в виде неструктурированного текста). Если он правильный, строится дерево запроса и передается дальше, в противном случае возвращается ошибка. Лексический и синтаксический анализ реализован с применением хорошо известных средств Unix bison и flex.

Лексическая структура определяется в файле scan.l и описывает идентификаторы, ключевые слова SQL и т.д. Для каждого найденного ключевого слова или идентификатора генерируется символ языка, который затем передается синтаксическому анализатору.

Синтаксис языка определен в файле gram.y в виде набора грамматических правил и действий, которые должны выполняться при срабатывании правил. Для построения дерева разбора используется код действий (являющийся кодом на C).

Файл scan.l преобразуется в программу на C scan.c с помощью flex, а gram.y — в gram.c с помощью bison. После этих преобразований исполняемый код анализатора создается обычным компилятором C. Не допускается вносить коррективы в сгенерированные файлы C, так как они будут перезаписаны при следующем вызове flex или bison.

ПРИМЕЧАНИЕ. Упомянутые преобразования и компиляция обычно производятся автоматически сборочными файлами Makefile, поставляемыми в составе дистрибутива.

Подробное описание bison и грамматических правил в gram.y выходит за рамки данной главы. Узнать больше о flex и bison можно из книг и документации. Изучение грамматики, описанной в gram.y, следует начать со знакомства с bison.

#### Преобразование

На этой стадии дерево разбора создается только с фиксированными знаниями о синтаксической структуре SQL. При его создании не просматриваются системные каталоги, что не дает возможность понять конкретную семантику запрошенной операции. После этого выполняется процедура преобразования, которая принимает дерево разбора от анализатора и выполняет семантический анализ, необходимый для понимания, к каким именно таблицам, функциям и операторам обращается запрос. Структура данных, которая создается для представления этой информации, называется деревом запроса.

Синтаксический разбор отделен от семантического анализа, потому что обращаться к системным каталогам можно только внутри транзакции, а начинать транзакцию сразу после получения строки с запросом нежелательно. Синтаксического разбора достаточно, чтобы распознать команды управления транзакциями (BEGIN, ROLLBACK и т.д.), поэтому их можно выполнить без дальнейшего анализа. Убедившись, что мы имеем дело с собственно запросом (например, SELECT или UPDATE), можно начинать транзакцию, если она еще не начата. Только после этого можно переходить к процедуре преобразования.

Дерево запроса, создаваемое процедурой преобразования, по структуре во многом похоже на дерево разбора, но отличается во многих деталях. Например, узел FuncCall в дереве разбора представляет то, что по синтаксису похоже на вызов функции. Этот узел может быть преобразован в узел FuncExpr или Aggref в зависимости от того, какой (обычной или агрегатной) окажется функция с заданным именем. Кроме того, в дерево запроса добавляется информация о фактических типах данных столбцов и результатов выражений.

## Планировщик и оптимизатор

Задача планировщика/оптимизатора — построить наилучший план выполнения. Определенный SQL-запрос (а значит, и дерево запроса) можно выполнить самыми разными способами, при этом получая одни и те же результаты. Если это не требует больших вычислений, оптимизатор запросов будет перебирать все возможные варианты планов, чтобы в итоге выбрать тот, который должен выполниться быстрее остальных.

ПРИМЕЧАНИЕ. В некоторых ситуациях рассмотрение всех возможных вариантов выполнения запросов занимает слишком много времени и памяти. В частности, это имеет место при выполнении запросов с большим количеством операций соединения. Поэтому, чтобы выбрать разумный (но не обязательно наилучший) план запроса за приемлемое время, PostgreSQL использует генетический оптимизатор запросов, когда количество соединений превышает предел.

Процедура поиска лучшего плана работает со структурами данных, называемыми путями, которые представляют собой упрощенные схемы планов, содержащие минимум информации, необходимой планировщику для принятия решений. Когда наиболее выгодный план выбран, строится полноценное дерево плана, которое и передается исполнителю. Оно описывает желаемый план выполнения достаточно подробно, чтобы исполнитель мог обработать его.

Сначала планировщик/оптимизатор вырабатывает планы для сканирования каждого отдельного отношения (таблицы), используемого в запросе. Множество возможных планов определяется в зависимости от наличия индексов в каждом отношении. Произвести последовательное сканирование отношения можно в любом случае, так что план последовательного сканирования создается всегда. Планы сканирования индекса также создаются для индексов, если их порядок сортировки соответствует предложению ORDER BY (если оно есть), или этот порядок может быть полезен для соединения слиянием (см. ниже).

Если в запросе требуется соединить два или несколько отношений, после того как будут определены все подходящие планы сканирования отдельных отношений, рассматриваются планы соединения. При этом возможны три стратегии соединения:

* соединение с вложенным циклом: Правое отношение сканируется один раз для каждой строки, найденной в левом отношении;
* соединение слиянием: каждое отношение сортируется по атрибутам соединения до начала соединения. Затем два отношения сканируются параллельно, и соответствующие строки, объединяясь, формируют строки соединения;
* соединение по хешу: сначала сканируется правое отношение и формируется хеш-таблица, ключ в которой вычисляется по атрибутам соединения. Затем сканируется левое отношение и по тем же атрибутам в каждой строке вычисляется ключ для поиска в этой хеш-таблице соответствующих строк справа.

Когда в запросе задействованы более двух отношений, окончательный результат должен быть получен из дерева с узлами соединения, имеющими по два входа. Планировщик рассматривает все возможные последовательности соединения и выбирает самую выгодную.

Если число задействованных в запросе отношений меньше geqo\_threshold, для поиска оптимальной последовательности соединений производится практически полный перебор. Планировщик отдает предпочтение соединениям между двумя отношениями, для которых есть соответствующее предложение соединения в условии WHERE (то есть, для которых находится ограничение вида where табл1.атр1=табл2.атр2). Пары соединения без подобного предложения рассматриваются в том случае, если нет другого выбора, то есть когда для определенного отношения не находятся предложения соединения с каким-либо другим отношением. Планировщик рассматривает все возможные планы для каждой пары соединения и выбирает самый выгодный из них (по его оценке).

Если geqo\_threshold превышается, последовательность соединений выбирается эвристическим путем.

Законченное дерево плана содержит узлы сканирования по индексу или последовательного сканирования базовых отношений, плюс узлы соединения с вложенным циклом, соединения слиянием или соединения по хешу (если требуется), плюс узлы дополнительных действий, например, сортировки или вычисления агрегатных функций. Большинство из этих узлов могут дополнительно производить отбор (отбрасывать строки, не удовлетворяющие заданному логическому условию) и расчеты (вычислять производный набор столбцов по значениям заданных столбцов, то есть вычислять скалярные выражения). Одна из задач планировщика — добавить условия отбора из предложения WHERE и вычисления требуемых выходных выражений к наиболее подходящим узлам дерева плана.

## pgAdmin

pgAdmin — это open source – средство для администрирования (и разработки БД) для СУБД PostgreSQL.

ПРИМЕЧАНИЕ: допускается использование pgAdmin не старше 4 версии.

## Apache Pulsar

Apache Pulsar — это open source – средство для распределённого обмена сообщениями на основе публикации / подписки.

## BIRT

BIRT является open-source инструментом для составления отчетов(по концепции WYSIWYG). BIRT может извлекать и комбинировать данные из множества различных источников данных (базы данных, файлы, Java, Javascript, веб-службы и т. д.) и использовать эти данные для создания отчетов (текстовых) и диаграмм (визуальных).

BIRT — это программный проект с открытым исходным кодом, который предоставляет технологическую платформу для визуализации данных. Пользователи могут встраивать отчеты в настольные и веб-приложения, особенно основанные на Java и Jakarta EE. BIRT — это проект программного обеспечения высшего уровня в рамках Eclipse Foundation.

### Дизайн

С помощью BIRT можно добавлять в свое приложение разнообразные отчеты.

Списки. Самые простые отчеты представляют собой списки данных. По мере того, как списки становятся длиннее, можно добавить группировку, чтобы упорядочить связанные данные вместе (заказы сгруппированы по клиентам, продукты сгруппированы по поставщикам). Если данные числовые, можно легко добавлять итоги, средние значения и другие сводки.

Диаграммы. BIRT предоставляет круговые диаграммы, линейные гистограммы и др. Диаграммы BIRT можно отображать в формате SVG и поддерживать события, обеспечивающие взаимодействие с пользователем.

Кросс-таблицы. Кросс-таблицы показывают данные в двух измерениях: продажи за квартал или количество посещений на веб-страницу.

Письма и документы. Уведомления, формальные письма и другие текстовые документы легко создавать с помощью BIRT. Документы могут включать текст, форматирование, списки, диаграммы и др.

Составные отчеты. Отчеты, объединяющие вышеперечисленное в один документ.

### Анатомия отчета

Отчеты BIRT состоят из четырех основных частей: данные, преобразование данных, бизнес-логика и представление.

Данные. Базы данных, веб-службы, объекты Java могут предоставлять данные для отчета BIRT. BIRT обеспечивает поддержку JDBC, XML, веб-служб и неструктурированных файлов, а также поддержку использования кода для доступа к другим источникам данных. Использование BIRT инфраструктуры открытого доступа к данным (ODA) позволяет любому создавать новый пользовательский интерфейс и поддержку среды выполнения для любого вида табличных данных. Кроме того, один отчет может включать данные из любого количества источников данных. BIRT также предоставляет функцию, позволяющую объединять разрозненные источники данных с помощью внутренних и внешних соединений.

Преобразование данных. Отчеты представляют данные, отсортированные, обобщенные, отфильтрованные и сгруппированные в соответствии с потребностями пользователя. В то время как базы данных могут выполнять часть этой работы, BIRT должен выполнять ее для «простых» источников данных, таких как плоские файлы или объекты Java. BIRT позволяет выполнять сложные операции, такие как группировка по суммам, процентам от общей суммы и т. д.

Бизнес-логика. Реальные данные редко структурированы точно так, как хотелось бы для отчета. Для многих отчетов требуется бизнес-логика для преобразования необработанных данных в информацию, полезную для пользователя. Если логика предназначена только для отчета, вы можете написать ее с помощью поддержки JavaScript в BIRT. Если ваше приложение уже содержит логику, вы можете вызвать существующий код Java.

Презентация. После того, как данные готовы, есть широкий выбор вариантов их представления пользователю. Таблицы, диаграммы, текст и многое другое. Один набор данных может отображаться несколькими способами, а в одном отчете могут быть представлены данные из нескольких наборов данных.

### Компоненты BIRT

Проект BIRT включает множество компонентов. Часть перечислена ниже с кратким описанием:

* Дизайнер отчетов BIRT. Дизайнер отчетов BIRT представляет собой перспективу Eclipse, которая используется для создания макетов отчетов BIRT. Эти проекты хранятся в открытом формате XML. Конструктор можно загрузить как приложение Rich Client Platform (RCP), набор подключаемых модулей, позволяющих использовать проекцию Designer в существующей сборке Eclipse, или как комплексную загрузку, включающую Eclipse.
* Механизм проектирования. Этот механизм отвечает за создание и изменение макетов отчетов. API механизма проектирования (DE API) объединяет функциональные возможности механизма проектирования и доступен для использования в любом проекте Java/Java EE. Дизайнер отчетов BIRT использует этот API внутри для создания макетов XML.
* Механизм отчетов. Механизм отчетов использует файлы макета отчета для создания и отображения отчетов. С помощью API механизма отчетов (RE API) механизм может быть встроен в любое приложение Java/Java EE. Веб-программа просмотра BIRT использует этот API для выполнения и отображения отчетов.
* Charting Engine - Charting Engine используется для проектирования и создания диаграмм как в автономном режиме, так и в виде встроенных в отчеты BIRT. API Charting Engine (CE API) позволяет разработчикам Java/Java EE добавлять возможности построения диаграмм в свои приложения. Механизмы проектирования и отчетов используют этот API для доставки диаграмм.
* BIRT Viewer - проект BIRT предоставляет образец «средства просмотра», который используется для предварительного просмотра отчетов в Eclipse. BIRT включает сервер Apache Tomcat, который вызывается каждый раз при предварительном просмотре отчета. Средство просмотра не только упаковано в виде подключаемого модуля Eclipse, но также доступно в виде отдельного приложения Java EE, которое можно использовать на любом сервере Java EE, совместимом с JSP. Подключаемый модуль Viewer также может быть встроен в приложение Rich Client Platform (RCP). BIRT обеспечивает веб-вывод в виде единого HTML-документа, разбитого на страницы HTML, PDF, XLS, DOC, PPT и Postscript. Кроме того, средство просмотра позволяет экспортировать данные в CSV, печатать и использовать функции оглавления.

# Версии используемого программного обеспечения.

Версии используемого программного обеспечения приводятся в таблице ниже (см.
Таблица 2).

Таблица 2 – Версии используемого программного обеспечения

| № п.п. | Класс ПО | Наименование ПО и версия | Правообладатель | Лицензия | Кол-во |
| --- | --- | --- | --- | --- | --- |
| 1 | Операционная система. | Ubuntu версии 20.XX и выше или Linux Mint версии 20.XX и выше (см. Примечание 1) | Canonical Ltd. | GNU GPL | 1 |
| 2 | Система управления базами данных | PostgreSQL, версия 13.2 и выше | The PostgreSQL Global Development Group. | PostgreSQL license | 1 |
| 3 | Система обмена сообщениями. | Apache PulsarВерсия 2.8.2 и выше | Apache Software Foundation. | Apache License | 1 |
| 4 | BIRT | 4.9.0 | Eclipse BIRT | Eclipse Public License 2.0 | 1 |
| 5 | MinIO | 1.0 | MinIO | GNU Affero General Public License | 1 |
| 6 | nginx | nginx 1.23.2 | nginx | BSD license | 1 |
| 7 | Redis | Redis 7.0 | Redis | BSD license | 1 |

**Примечания:**

1.Возможно использование других UNIX-подобных операционных систем с поддержкой работы системы контейнеризации Docker.

2. Версия ПО может измениться при выходе новой версии или при изменении требований безопасности.

# Требования к аппаратному обеспечению

Минимальные и рекомендуемые требования к аппаратному обеспечению для установки IoT платформы:

Минимальные требования:

* 4 core x86 (intel Sandy Bridge) от 2.5 Мгц;
* 16 Gb RAM;
* 600 Gb HDD/SSD

Рекомендуемые требования (при обработке 10000 ключей телеметрии в секунду):

* 16 core x86 (intel Sandy Bridge) от 2.5 Мгц;
* 32 Gb RAM;
* 15 Tb SSD

Минимальные требования к аппаратному обеспечению для установки Шлюза:

* 2 core x86 (intel Sandy Bridge) от 1.8 Мгц;
* 4 Gb RAM;
* 128 Mb HDD/SSD

# Перечень необходимого ПО или общесистемных пакетов

IoT платформа поддерживает работу под управлением операционных систем семейства Linux с поддержкой работы системы контейнеризации Docker версии не ниже 1.18. Рекомендуемые операционные системы: Ubuntu версии 20.XX и выше или Linux Mint версии 20.XX и выше. Процесс установки операционной системы выходит за рамки руководства и должен выполняться в соответствии с инструкциями и рекомендациями к устанавливаемой ОС.

Для обеспечения функционирования IoT платформы требуется дополнительно установить следующее базовое программное обеспечение (бесплатно распространяемое программное обеспечение с открытым исходным кодом):

* Docker версии 1.18 или выше;
* Docker-compose версии 1.24 или выше;
* PostgreSQL, версия не ниже 13.2;
* Git версии 1.8.3 или выше.

Работа с IoT платформой возможна с использованием следующих браузеров:

* Яндекс.Браузер 20 версии и более новая версия;
* Google Chrome 72 версии и более новая версия;
* Firefox Browser 85.0 версии и более новая версия.

Браузеры не требуют установки дополнительных плагинов.

## Состав дистрибутива

Дистрибутив представляет собой Docker контейнер.

Процесс развертывания Платформы описан в документе «Руководство по развертыванию», RU.1197746546282.00017-01 32 01

# Соответствие ПЭВМ требованиям реестра Российского программного обеспечения

Программа для ЭВМ «Инфраструктурная IoT платформа», версия 1.3 правомерно введена в гражданский оборот на территории Российской Федерации, экземпляры программного обеспечения либо права использования программного обеспечения, услуги по предоставлению доступа к программному обеспечению свободно реализуются на всей территории Российской Федерации, отсутствуют ограничения, установленные в том числе иностранными государствами и препятствующие распространению или иному использованию программы для электронных вычислительных машин и базы данных на территории Российской Федерации или территориях отдельных субъектов Российской Федерации.

Сведения о программе для ЭВМ «Инфраструктурная IoT платформа», версия 1.3 не составляют государственную тайну, программное обеспечение не содержит сведений, составляющих государственную тайну.

Программа для ЭВМ «Инфраструктурная IoT платформа», версия 1.3 не имеет принудительного обновления и управления из-за рубежа.

Гарантийное обслуживание, техническая поддержка и модернизация Программа для ЭВМ «Инфраструктурная IoT платформа», версия 1.3 осуществляются российской коммерческой организацией без преобладающего иностранного участия.