

Министерство науки и высшего образования Российской Федерации  
Санкт-Петербургский политехнический университет Петра Великого  
Физико-механический институт

Работа допущена к защите  
Руководитель образовательной программы  
«Прикладная математика и информатика»  
\_\_\_\_\_ К.Н. Козлов  
« \_\_\_\_\_ » \_\_\_\_\_ 202\_ г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**  
**РАБОТА БАКАЛАВРА**  
**АВТОМАТИЧЕСКАЯ ГЕНЕРАЦИЯ КОНФИГУРАЦИЙ ЭЛЕМЕНТОВ**  
**ИНФРАСТРУКТУРЫ ПРОГРАММНЫХ СИСТЕМ ДЛЯ РАБОТЫ С**  
**БОЛЬШИМИ ДАННЫМИ**

по направлению подготовки 01.03.02 Прикладная математика и информатика  
Направленность (профиль) 01.03.02\_02 Системное программирование

Выполнил  
студент гр. 5030102/10201

И.И. Хамидуллин

Руководитель  
д.т.н.,  
профессор ВШПМиВФ

Ф.А. Новиков

Консультант ВКР

Д.Ю. Иванов

Консультант  
по нормоконтролю

Л.А. Ареньева

**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
ПЕТРА ВЕЛИКОГО  
Физико-механический институт**

УТВЕРЖДАЮ

Руководитель образовательной программы

\_\_\_\_\_ К.Н. Козлов

« \_\_\_\_\_ » \_\_\_\_\_ 202\_ г.

**ЗАДАНИЕ  
на выполнение выпускной квалификационной работы**

студенту Хамидуллину Ильясу Ильназовичу гр. 5030102/10201

1. Тема работы: Автоматическая генерация конфигураций элементов инфраструктуры программных систем для работы с большими данными.
2. Срок сдачи студентом законченной работы: июнь 2025 г.
3. Исходные данные по работе:
  - Декларативные конфигурационные файлы в формате YAML, задаваемые пользователем (инженером данных) для описания инфраструктуры обработки данных
  - Демонстрационный датасет, включающий тестовые данные, хранящиеся в различных источниках (PostgreSQL, S3) и обрабатываемые в системе
  - Автоматически сгенерированные конфигурационные файлы
4. Инструментальные средства:
  - Языки программирования: Python
  - Форматы конфигурационных файлов: YAML, JSON
  - Среда разработки: VS Code
  - Система управления версиями: Git
  - Средства контейнеризации и оркестрации: Docker, Docker Compose
  - Платформы потоковой обработки данных: Apache Kafka, Kafka Connect (Debezium)
  - Системы управления базами данных (СУБД): PostgreSQL, ClickHouse
  - BI-инструмент: Apache Superset

5. Содержание работы (перечень подлежащих разработке вопросов):

5.1. Введение.

5.2. Постановка задачи.

5.3. Обзор существующих решений.

5.4. Введение в предметную область.

5.5. Разработка инструмента

5.6. Проектирование и реализация инфраструктуры для работы с большими данными

5.7. Исследование разработанного продукта

5.8. Заключение

Ключевые источники литературы:

- Альфред Ахо, Рави Сети, Джефффри Ульман. Раскрутка // Компиляторы: принципы, технологии и инструменты = Compilers: Principles, Techniques, and Tools. — М.: Вильямс, 2003. — С. 681—684. — 768 с. — ISBN 5-8459-0189-8.
- Фаулер М. Непрерывная поставка: Надежная автоматизация сборки, тестирования и развертывания программного обеспечения = Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. — М.: Вильямс, 2011. — 432 с. — ISBN 978-5-8459-1739-3
- Таненбаум Э., ван Стин М. Распределенные системы: принципы и парадигмы = Distributed Systems: Principles and Paradigms. — 2-е изд. — М.: ДМК Пресс, 2021. — 584 с. — ISBN 978-5-97060-708-4

6. Дата выдачи задания: 03.02.2025.

Руководитель ВКР

Ф.А. Новиков

Консультант ВКР

Д.Ю. Иванов

Задание принял к исполнению

Студент

И.И. Хамидуллин

## РЕФЕРАТ

На 58 с., 17 рисунков, 0 таблиц, 3 приложения

**КЛЮЧЕВЫЕ СЛОВА:** АВТОМАТИЗАЦИЯ РАЗВЕРТЫВАНИЯ, БОЛЬШИЕ ДАННЫЕ, ГЕНЕРАЦИЯ КОНФИГУРАЦИЙ, ДЕКЛАРАТИВНОЕ ОПИСАНИЕ, YAML, DOCKER COMPOSE, ПОТОКОВАЯ ОБРАБОТКА В РЕАЛЬНОМ ВРЕМЕНИ, ИНФРАСТРУКТУРА КАК КОД.

Дипломная работа посвящена актуальной проблеме автоматизации развертывания инфраструктуры для работы с большими данными.

Целью работы является разработка программного инструмента Data Platform Deployer(далее dpd), способного на основе декларативного описания, предоставленного пользователем в формате YAML, генерировать полный набор конфигурационных файлов и скриптов для запуска комплексной платформы данных. Входное описание включает определение таких компонентов, как системы управления базами данных (PostgreSQL в качестве источника, ClickHouse в качестве аналитического хранилища), S3-совместимое объектное хранилище (Minio), брокер сообщений Apache Kafka с настроенными топиками и коннекторами Kafka Connect (включая Debezium для CDC и S3 Sink), а также инструмент бизнес-аналитики Apache Superset. Разработанный инструмент dpd автоматически формирует docker-compose.yml файлы для контейнеризации сервисов, скрипты их инициализации и обеспечивает согласованность настроек между всеми компонентами.

Ключевыми преимуществами предлагаемого решения являются воспроизводимость конфигураций, значительное сокращение трудозатрат по сравнению с ручной настройкой, модульность для поддержки новых компонентов и обеспечение корректности взаимосвязей в развертываемой системе.

## **ABSTRACT**

58 pages, 17 figures, 0 tables, 3 appendices

**KEYWORDS:** DEPLOYMENT AUTOMATION, BIG DATA, CONFIGURATION GENERATION, DECLARATIVE DESCRIPTION, YAML, DOCKER, DOCKER COMPOSE, REAL TIME STREAM PROCESSING, INFRASTRUCTURE AS CODE.

The subject of the graduate qualification work is «Automatic generation of configurations for infrastructure elements of software systems for big data processing».

This thesis addresses the relevant problem of automating the deployment of infrastructure for big data operations.

The aim of the work is to develop a software tool dpd (Data Platform Deployer) capable of generating a complete set of configuration files and scripts for launching a comprehensive data platform based on a declarative description provided by the user in YAML format. The input description includes the definition of components such as database management systems (PostgreSQL as a source, ClickHouse as an analytical data warehouse), S3-compatible object storage (Minio), Apache Kafka message broker with configured topics and Kafka Connect connectors (including Debezium for CDC and S3 Sink), and the Apache Superset business intelligence tool. The developed dpd tool automatically generates docker-compose.yml files for service containerization, their initialization scripts, and ensures the consistency of settings across all components.

Key advantages of the proposed solution include configuration reproducibility, significant reduction in labor costs compared to manual setup, modularity for supporting new components, and ensuring the correctness of interconnections within the deployed system.

## СОДЕРЖАНИЕ

Введение .....	7
Постановка задачи.....	8
Глава 1. Обзор существующих решений .....	11
Глава 2. Введение в предметную область .....	14
2.1. Программные системы для работы с большими данными .....	16
Глава 3. Разработка инструмента .....	20
3.1. План разработки инструмента.....	20
3.2. Язык декларативного описания (DSL) .....	22
3.3. Описание процесса автогенерации конфигураций программных систем платформы данных .....	24
Глава 4. Проектирование и реализация инфраструктуры программных си- стемы для работы с большими данными .....	27
4.1. Пример Northwind: Связь клиентов с заказами .....	28
4.2. Пример Chinook: Анализ музыкальных продаж .....	37
Глава 5. Исследование разработанного продукта .....	42
5.1. Подтверждение эффективности автоматической генерации конфигура- ций .....	43
5.2. Оценка снижения трудоемкости и времени развертывания .....	44
5.3. Обеспечение корректности, согласованности и воспроизводимости конфигураций .....	45
5.4. Результаты апробации в ПАО «Магнит» и экспертная оценка.....	46
Заключение .....	48
Словарь терминов.....	51
Список использованных источников.....	56
Приложение 1. Грамматика языка DPD .....	59
Приложение 2. SQL код для забора данных из Kafka в ClickHouse .....	63
Приложение 3. Апробация в компанию МАГНИТ .....	65

## ВВЕДЕНИЕ

Современный этап развития цифровых технологий характеризуется стремительным увеличением объемов данных, генерируемых в различных сферах человеческой деятельности. Этот феномен, известный как "информационный взрыв" требует принципиально новых подходов к обработке, хранению и анализу информации. В условиях, когда традиционные методы работы с данными становятся неэффективными, особую актуальность приобретают технологии больших данных (Big Data), предлагающие инновационные решения для извлечения ценных знаний из огромных массивов неструктурированной информации[1].

**Актуальность** темы данного исследования обусловлена несколькими ключевыми факторами. Во-первых, в эпоху цифровой экономики данные становятся стратегическим ресурсом, сравнимый по значимости с традиционными материальными активами[2]. Во-вторых, сложность современных информационных систем достигла такого уровня, когда ручная настройка их компонентов становится не только трудоемкой, но и потенциально подверженной человеческим ошибкам. В-третьих, переход к agile-методологиям и DevOps-практикам требует новых подходов к управлению инфраструктурой, обеспечивающих скорость, надежность и воспроизводимость развертывания сложных систем[3].

В контексте этих вызовов особое значение приобретает автоматизация процессов настройки и конфигурирования инфраструктуры для работы с большими данными. Концепция "Инфраструктура как код" (Infrastructure As Code)[4] предлагает подходы для управления и предоставления вычислительной инфраструктуры с помощью декларативных или скриптовых определений.

Традиционные подходы, основанные на ручном редактировании конфигурационных файлов и скриптов, не только требуют значительных временных затрат, но и создают риски возникновения "дрейфа конфигураций" (configuration drift)[5], когда фактическое состояние системы постепенно расходится с документально зафиксированным.

## ПОСТАНОВКА ЗАДАЧИ

В современных условиях стремительного роста объёмов данных и усложнения архитектуры информационных систем ручное конфигурирование инфраструктуры для работы с большими данными становится неэффективным и подверженным ошибкам. Существующие инструменты автоматизации, такие как Terraform и Ansible, предоставляют общие механизмы развёртывания, но не предлагают специализированных решений для технологий Big Data, требующих согласованной настройки множества взаимосвязанных компонентов: систем хранения, потоковой обработки, ETL-конвейеров и инструментов визуализации.

**Целью** данной работы является разработка программного инструмента Data Platform Deployer (далее dpd), автоматизирующего процесс создания конфигурационных файлов для развёртывания платформы обработки данных на основе декларативного описания её компонентов пользователем. Инструмент принимает на вход описание целевой инфраструктуры в формате YAML и генерирует готовые к использованию артефакты:

- конфигурационные файлы `docker-compose.yml` для быстрого развёртывания всех необходимых сервисов в контейнерной среде;
- скрипты инициализации и базовые конфигурационные файлы для СУБД (PostgreSQL[6], ClickHouse[7]), адаптированные под типовые задачи обработки данных;
- конфигурации для S3-совместимого хранилища[8];
- конфигурации для брокера сообщений Apache Kafka[9], включая создание топиков и настройки Kafka Connect с необходимыми коннекторами (Debezium для CDC, S3 Sink Connector);
- конфигурации для AKNQ — инструмента мониторинга Kafka и Kafka Connect[10];
- базовые настройки для подключения BI-систем (например, Apache Superset[11]) к развёрнутым источникам данных.

При разработке инструмента dpd должны соблюдаться следующие критерии:

- Воспроизводимость: идентичные конфигурации при одинаковом входном описании.
- Масштабируемость: поддержка добавления новых типов компонентов через модули.



- Согласованность: автоматическая проверка зависимостей между сервисами.

Для достижения поставленной цели были определены следующие задачи:

1. Анализ предметной области и существующих подходов к развёртыванию платформ Big Data:
  - изучение типовых архитектурных паттернов платформ для обработки больших данных[12]
  - исследование возможностей и ограничений существующих инструментов управления конфигурациями и IaC в контексте Big Data;
  - определение ключевых компонентов и их типовых конфигураций для включения в состав dpd.
2. Проектирование метамодели декларативного описания инфраструктуры:
  - разработка структуры YAML-файла для описания компонентов платформы, их параметров и взаимосвязей;
  - проектирование системы валидации входных конфигураций на основе JSON Schema[13] для обеспечения корректности пользовательского ввода.
3. Разработка ядра генератора конфигураций (dpd):
  - реализация логики парсинга входного YAML-описания;
  - создание механизма шаблонизации для генерации конфигурационных файлов (`docker-compose.yml`, настройки сервисов и др.).
4. Реализация модулей генерации для ключевых компонентов платформы:
  - модуль для Apache Kafka и Kafka Connect (включая коннекторы Debezium PostgreSQL, S3 Sink);
  - модуль для СУБД PostgreSQL (источник данных);
  - модуль для аналитической СУБД ClickHouse (хранилище данных);
  - модуль для S3-совместимого хранилища Minio (архивное хранилище/Data Lake);
  - модули для вспомогательных инструментов: AKHQ (мониторинг Kafka), Apache Superset (BI).
5. Тестирование и валидация инструмента dpd:
  - развёртывание тестовых стендов с различной конфигурацией при помощи сгенерированных артефактов;

- функциональное тестирование развернутых платформ для проверки корректности работы и взаимодействия компонентов;
- сравнительный анализ времени и сложности развёртывания платформы с использованием dpd и традиционных ручных методов.

## ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Развертывание и управление инфраструктурой для систем обработки больших данных (Big Data) представляет собой сложную задачу, требующую координации множества разнородных компонентов, настройки их взаимодействия и обеспечения масштабируемости, надёжности и безопасности.

Современные подходы к управлению инфраструктурой стремятся автоматизировать эти процессы, однако специфика Big Data-систем накладывает дополнительные требования. В данном разделе рассматриваются существующие инструменты управления инфраструктурой и анализируются ключевые требования к интеграции компонентов, характерные для платформ обработки больших данных, с акцентом на решения, актуальные для российского рынка.

**А. Управляемые облачные сервисы (Managed Cloud Services)** на примере российских провайдеров Российские облачные провайдеры, такие как Yandex Cloud[14] и VK Cloud[15] (ранее Mail.ru Cloud Solutions), активно развивают свои портфели управляемых сервисов, предназначенных для работы с большими данными. Эти сервисы позволяют значительно упростить создание и обслуживание сложной инфраструктуры.

Предлагаемые сервисы:

- **Yandex Cloud:** Yandex Data Proc (управляемый сервис для Apache Spark™ и Apache Hadoop®), Yandex Managed Service for Apache Kafka®, Yandex Managed Service for ClickHouse®, Yandex Managed Service for Greenplum®, Yandex Managed Service for PostgreSQL, объектное хранилище Yandex Object Storage (совместимое с S3 API).
- **VK Cloud:** управляемые базы данных (PostgreSQL, ClickHouse, MySQL и др.), сервис «Большие данные» на базе Arenadata Hadoop (ADH) и Arenadata Kafka (ADK) для пакетной и потоковой обработки, объектное хранилище (совместимое с S3 API).

Механизм генерации конфигураций: при создании и настройке управляемых сервисов через веб-консоль, CLI или API провайдера пользователь указывает высокоуровневые параметры (тип и количество вычислительных узлов, версии ПО, базовые настройки сети и параметры безопасности). Облачная платформа автоматически генерирует и поддерживает

низкоуровневые конфигурации виртуальной инфраструктуры и сервисов (например, \*-site.xml для Hadoop, server.properties для Kafka). Возможности кастомизации расширены через дополнительные опции или параметры запуска.

Преимущества:

- значительное упрощение развёртывания и управления: время запуска инфраструктуры сокращается с недель или дней до часов или минут;
- снижение операционной нагрузки: провайдер обеспечивает обновление ПО, патчинг, мониторинг и доступность;
- встроенные механизмы масштабирования и отказоустойчивости;
- интеграция с сервисами экосистемы провайдера.

Недостатки:

- привязка к конкретному провайдеру (vendor lock-in);
- ограниченная гибкость в глубоких настройках;
- высокая стоимость при постоянной нагрузке;
- непрозрачность детальных конфигураций.

## В. Интегрированные платформы данных (Integrated Data Platforms)

на примере российских разработок На российском рынке представлены комплексные решения для жизненного цикла работы с данными, от сбора и хранения до обработки и анализа. Примеры: Arenadata[16] и CedrusData[17].

- **Arenadata:** продукты на базе открытого кода, включая Arenadata Hadoop (ADH), Arenadata DB (Greenplum), Arenadata Streaming (Kafka, NiFi), Arenadata QuickMarts (ClickHouse). Для управления развёртыванием используется Arenadata Platform Manager (ADPM). Механизм генерации (ADPM): пользователь задаёт через интерфейс или декларативный файл состав кластера и ключевые параметры; ADPM генерирует и применяет конфигурации для всех компонентов, обеспечивая их согласованность.
- **CedrusData:** платформа для высокопроизводительной аналитики и обработки данных, включающая распределённое хранилище, SQL-обработку и инструменты управления. Механизм генерации: инсталлятор или скрипты запрашивают у администратора

параметры системы, после чего генерируются и применяются конфигурации для всех компонентов платформы.

Преимущества:

- единая точка входа и управления;
- проверенные интеграции и оптимальная совместимость;
- упрощённое развёртывание и обновление;
- техническая поддержка от вендора.

Недостатки:

- высокая стоимость лицензий и поддержки;
- «чёрный ящик» внутренних настроек;
- ограниченный выбор компонентов и версий;
- зависимость от экосистемы вендора;
- сложность освоения комплексных платформ.

**С. Kubernetes Operators** Операторы в Kubernetes[18] позволяют управлять stateful-приложениями Big Data: Strimzi для Kafka, CrunchyData и Zalando для PostgreSQL, операторы для ClickHouse, Flink, Spark и др. Механизм генерации: пользователь задаёт CRD (Custom Resource Definition) с высокоуровневым описанием (например, `kind: Kafka, spec:replicas:3, storage:...`), оператор создаёт и управляет Kubernetes-объектами (Deployments, StatefulSets, ConfigMaps, Secrets) в соответствии с этим описанием.

Преимущества: декларативный подход, автоматизация жизненного цикла, нативная модель управления в Kubernetes. Недостатки: требует инфраструктуры и экспертизы в Kubernetes, каждый оператор соответствует одному сервису, интеграции между операторами частично ручные, не генерирует `docker-compose.yml` и не подходит для сред вне Kubernetes.

**D. Шаблонизаторы и модули для инструментов управления конфигурациями** Ansible, Chef, Puppet, SaltStack с шаблонами (Jinja2, ERB) и готовыми ролями/рецептами для конкретных приложений (PostgreSQL, Kafka и т.д.). Механизм генерации: переменные (например, число брокеров, параметры памяти) задаются в YAML-файлах, шаблоны конфигураций используют эти переменные для генерации файлов, которые затем распространяются на узлы.

Преимущества: гибкость, переиспользование кода, интеграция с существующими CI/CD. Недостатки: требует знания инструментов, описание

системы разбросано между плейбуками, шаблонами и переменными, сложнее задать стек целиком в одном декларативном файле.

Существующие решения предлагают разные уровни абстракции и автоматизации для развёртывания Big Data-систем. Управляемые облачные сервисы и интегрированные платформы обеспечивают высокий уровень автоматизации ценой гибкости и привязки к поставщику. Kubernetes Operators дают декларативность, но требуют экосистемы Kubernetes и экспертизы. Шаблонизаторы в конфигурационных инструментах позволяют генерировать файлы, но требуют значительной ручной настройки.

Подход, разрабатываемый в рамках данной работы, предлагает специализированный инструмент для автоматической генерации конфигураций распространённых Big Data-технологий из единого высокоуровневого YAML-файла, что снижает порог входа и ускоряет итерации при построении и тестировании платформ.

## **ГЛАВА 2. ВВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ**

В начале XXI века человечество вступило в эпоху информации, характеризующуюся беспрецедентным ростом объемов генерируемых и накапливаемых данных. Этот феномен, получивший название "Большие данные" (Big Data), стал одной из определяющих тенденций технологического и социально-экономического развития. Большие данные описываются не просто их колоссальным объемом (Volume), измеряемым в петабайтах и эксабайтах, но и другими ключевыми характеристиками. К ним относятся высокая скорость (Velocity) поступления и необходимость быстрой обработки, часто в режиме реального времени, а также чрезвычайное разнообразие (Variety) форматов – от структурированных данных в реляционных таблицах до неструктурированных текстов, изображений, видео, аудиопотоков, данных с сенсоров и логов веб-серверов. Нередко выделяют и дополнительные характеристики, такие как достоверность (Veracity), указывающая на неопределенность и возможное наличие шумов, неточностей или пропусков в данных, и ценность (Value), подчеркивающая потенциальную пользу, которую можно извлечь из анализа этих данных[19].

Движущими силами этого информационного взрыва стали повсеместная цифровизация бизнес-процессов, распространение интернета и мобильных

устройств, рост популярности социальных сетей, развитие технологий Интернета вещей (IoT), когда миллиарды устройств непрерывно генерируют данные об окружающей среде и своем состоянии. Традиционные подходы к хранению и обработке данных, основанные на реляционных базах данных и централизованных вычислениях, оказались неспособны эффективно справляться с такими масштабами, скоростью и разнообразием информации[20].

Понимание того, что эти огромные массивы данных содержат скрытые закономерности, тенденции и знания, привело к формированию новой парадигмы. Вместо того чтобы рассматривать данные лишь как побочный продукт деятельности, организации начали видеть в них стратегический актив. Способность собирать, обрабатывать, анализировать большие данные и извлекать из них ценную информацию (insights) превратилась в критически важное конкурентное преимущество. Применение анализа больших данных охватывает практически все сферы деятельности:

- Бизнес и ритейл: Персонализация маркетинговых предложений, оптимизация цепочек поставок, прогнозирование спроса, анализ потребительского поведения, управление рисками, обнаружение мошенничества[21].
- Наука: Ускорение исследований в геномике, астрономии, физике частиц[22], климатологии, материаловедении.
- Здравоохранение: Персонализированная медицина, анализ медицинских изображений, прогнозирование эпидемий, оптимизация работы клиник, разработка новых лекарств.
- Производство: Предиктивное обслуживание оборудования, контроль качества продукции, оптимизация производственных процессов.
- Государственное управление: Улучшение городских служб ("умный город"), анализ транспортных потоков, прогнозирование и предотвращение чрезвычайных ситуаций, повышение эффективности государственных услуг.
- Финансы: Алгоритмический трейдинг, оценка кредитоспособности, выявление финансовых махинаций.

Таким образом, большие данные – это не просто технологический вызов, связанный с хранением и обработкой информации. Это фундаментальный сдвиг, открывающий новые возможности для инноваций, повышения эффективности и создания ценности во всех аспектах человеческой деятельности. Умение работать с большими данными становится необходимым навыком для специалистов



различных профилей, а построение надежной и эффективной инфраструктуры для их обработки – ключевой задачей для организаций, стремящихся сохранить и укрепить свои позиции в современном мире [23]. Неспособность адаптироваться к этой новой реальности и использовать потенциал больших данных может привести к потере конкурентоспособности и отставанию в развитии.

## **2.1. Программные системы для работы с большими данными**

Рассмотрим ключевые компоненты нашей платформы данных: от источников до BI-уровня, указав, для чего каждый инструмент служит, какие задачи решает и какую роль играет в общей системе. Инструменты выбраны исходя из доклада Smart Data 2024 - State of Data RU Edition[24].

### **1. Транзакционная база данных**

PostgreSQL выступает в платформе в роли традиционного реляционного источника данных[6]. Он хранит структурированную информацию и обеспечивает гарантии ACID[25], сложные SQL-запросы и транзакционные операции. В рамках нашей архитектуры PostgreSQL отвечает за сохранность «исторических» и «оперативных» данных, а также выступает отправной точкой для потоковой репликации Change Data Capture[26] при помощи Debezium[27]. Основная цель PostgreSQL – быть надежным первоисточником, от которого запускаются процессы инкрементального экспорта изменений и пакетной выгрузки данных.

### **2. Объектное хранилище**

Minio в нашей платформе реализует объектное хранилище, совместимое с API Amazon S3[8]. Оно удобно для размещения файловых загрузок, дампов баз данных, бэкапов, а также — в качестве «озера данных»[28] — для хранения сырых или уже подготовленных дата-сетов в формате Parquet, CSV, JSON и т. д. Minio обеспечивает горизонтальное масштабирование и возможность распределенного хранения больших объемов неструктурированных данных. Его задача – служить долговременным и недорогим репозиторием для «сырых» данных и результатов пакетных обработок.

### **3. Брокер сообщений**

Kafka выступает в роли распределенного брокера сообщений и обеспечивает надежную передачу сообщений между компонентами системы[9].



Она поддерживает высокую пропускную способность, устойчивость к сбоям и возможность горизонтального масштабирования. В платформе Kafka используется для организации событийного потока изменений из PostgreSQL (через Debezium) и передачи данных в потребителей – как для потоковой обработки, так и для загрузки в аналитическое хранилище. Kafka гарантирует упорядоченную доставку, сохранение истории сообщений (ретеншн) и управление потребительскими группами.

#### 4. Инструмент для мониторинга

AKHQ (ранее «Kafka HQ») [9] – это веб-интерфейс для администрирования и мониторинга кластера Kafka. Он предоставляет удобный UI для просмотра топиков, чтения и отправки сообщений, управления партициями и оффсетами, мониторинга состояния брокеров и групп потребителей. В платформе AKHQ решает задачу оперативного контроля за событиями в шине: инженер может быстро проверить, какие данные передаются, обнаружить «залипания» потребителей или переполненные топики, а также проводить ручную отладку потоков без необходимости работы с CLI или написания собственных скриптов.

#### 5. Инструмент для потоковой интеграции данных

Debezium[27] – это платформа Change Data Capture (CDC)[26], реализованная в виде набора коннекторов для Kafka Connect[10]. Kafka Connect, в свою очередь, представляет собой фреймворк для потоковой интеграции: он запускает коннекторы-источники (Source Connectors), которые читают данные из внешних систем (PostgreSQL, MongoDB и пр.), преобразуют их в события и записывают в топики Kafka, а также коннекторы-синкеры (Sink Connectors) для доставки данных из Kafka в хранилища (ClickHouse, S3 и т. д.). В нашей архитектуре Debezium Source Connector подключается к WAL PostgreSQL, транслирует изменения в Kafka, а далее отдельный Sink Connector записывает события в ClickHouse или выполняет другие действия. Цель этого двойного инструмента – обеспечить автоматическую и непрерывную синхронизацию данных между компонентами без написания собственного кода обработки.

#### 6. Аналитическое хранилище данных

ClickHouse[7] – это колоночная аналитическая СУБД с высокой скоростью выполнения сложных OLAP-запросов[29] на больших объемах данных. Она оптимизирована для чтения, эффективно сжимает колоночные дан-

ные и поддерживает параллельную обработку. В платформе ClickHouse служит основным аналитическим хранилищем, в которое из Kafka через Kafka Connect поступают агрегированные или сырые события. Благодаря своей архитектуре ClickHouse позволяет быстро строить отчеты, дашборды и выполнять ad-hoc анализ, обрабатывая десятки и сотни миллионов строк за доли секунды.

## 7. Инструмент для визуализации данных

Superset[11] – это BI-платформа с веб-интерфейсом, позволяющая создавать интерактивные дашборды, визуализации и аналитические отчеты поверх различных источников данных (SQL-базы, колоночные хранилища, дата-видео и т. д.). В нашей системе Superset подключается к ClickHouse и предоставляет бизнес-пользователям и аналитикам средства для построения графиков, таблиц, гео-карт и скриптов на SQL. Его цель – закрыть уровень представления данных: от сложных SQL-запросов непосредственного взаимодействия до простого drag-and-drop интерфейса для быстрого получения инсайтов.

В совокупности эти инструменты образуют сквозной конвейер данных: от первичного сбора и хранения, через трансформации и передачу событий, до хранения в аналитической СУБД и визуализации для конечных пользователей. Такое разделение ответственности позволяет использовать лучшие свойства каждого компонента и легко масштабировать или заменять отдельные части платформы по мере роста требований и объемов данных. Именно поэтому в рамках настоящей работы рассматривается вопрос создания инструмента, призванного облегчить рутинную работу инженера данных по настройке многочисленных компонентов, а также предоставить ему удобное средство для декларативного описания и развертывания конфигураций платформ данных. Исходя из всего вышеперечисленного, диаграмма компонентов(рис.2.1) и Use-Case диаграмма(рис.2.2) разработанного инструмента dpd выглядят следующим образом

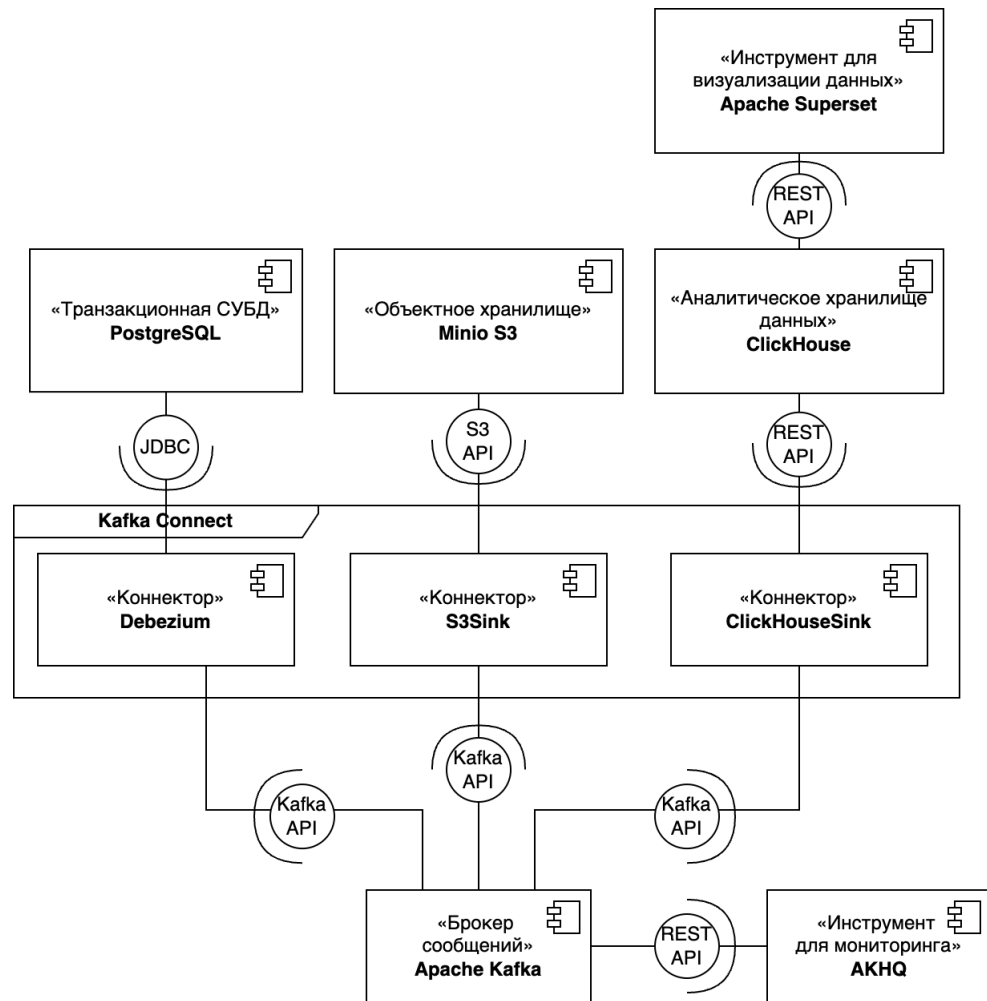


Рис.2.1. Диаграмма компонентов

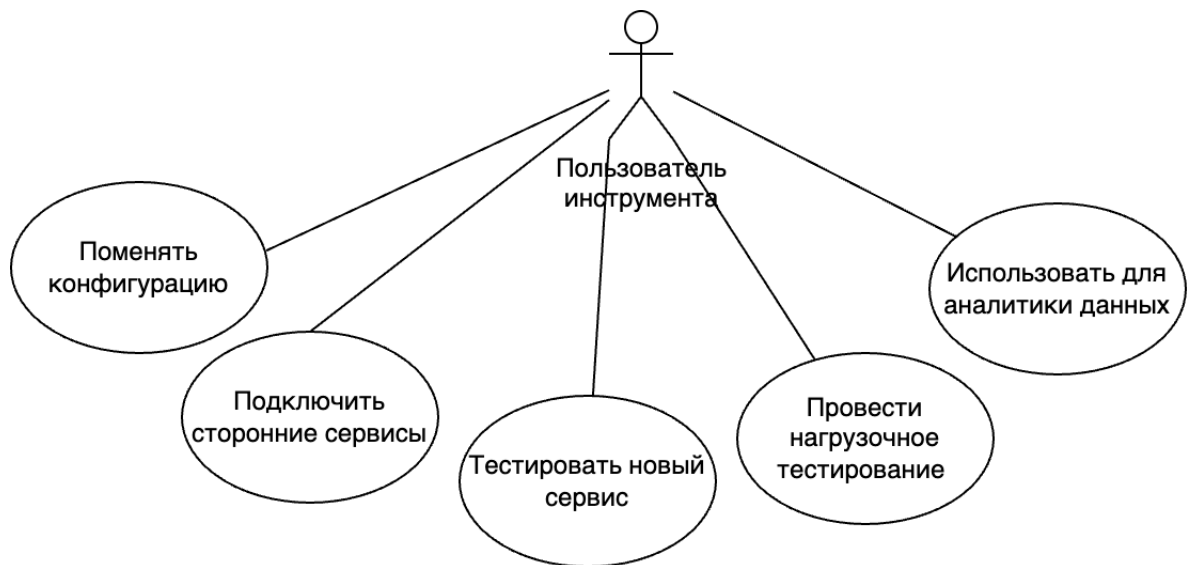


Рис.2.2. Use-Case диаграмма

## ГЛАВА 3. РАЗРАБОТКА ИНСТРУМЕНТА

Предварительно важно определить чёткую последовательность этапов, которая позволит организовать работу над инструментом системно и управляемо. Это поможет сократить риски неопределённости, оптимально распределить ресурсы и обеспечить своевременный контроль качества на каждом шаге.

Хорошим стилем является наличие введения к главе. Во введении может быть описана цель написания главы, а также приведена краткая структура главы.

### 3.1. План разработки инструмента

- Этап 1.** Сбор и анализ требований На этом этапе формализуются функциональные и нефункциональные требования: определяется, какие компоненты Big Data стека поддерживаются, в каком формате задается исходный YAML, какие конфигурационные артефакты должны генерироваться.
- Этап 2.** Проектирование архитектуры Разрабатывается модульная архитектура инструмента, включающая парсер входного описания, генератор промежуточного представления (AST), набор шаблонов для конфигураций инструментов и механизм их объединения в итоговые файлы. Определяются границы ответственности каждого модуля, протокол взаимодействия между ними и формат плагинов для расширения функциональности.
- Этап 3.** Определение языка декларативного описания Уточняются синтаксис и семантика входного YAML: структура разделов, типы параметров, возможные зависимости и проверки корректности. Разрабатывается схема jsonschema для валидации пользовательских описаний на раннем этапе.
- Этап 4.** Реализация ядра: парсер и промежуточное представление Пишется компонент, который читает декларативный файл, проводит его валидацию по схеме (jsonschema), конструирует внутреннее дерево объектов (AST) с отображением всех сущностей и их связей. Этот модуль обеспечивает основу для дальнейших операций по генерации конфигураций.

**Этап 5.** Разработка генераторов конфигурационных файлов На основе AST реализуются плагины-генераторы для каждого типа артефакта:

- docker-compose.yaml с сервисами и сетями;
- Конфигурации PostgreSQL (postgresql.conf, init.sql);
- JSON-файлы коннекторов Debezium и S3Sink;
- Файлы настроек для ClickHouse;
- Конфигурационные файлы для AKHQ и Superset.

Каждый генератор использует шаблонизатор и преобразует параметры из AST в конкретные строки и блоки файлов.

**Этап 6.** Создание CLI-интерфейса Реализуется утилита командной строки, позволяющая пользователю запускать генерацию: передавать путь к входному файлу, указывать директорию вывода, включать опции валидации и отладки. CLI обеспечивает удобство использования инструмента в скриптах и CI/CD-пайплайнах[25].

**Этап 7.** Модуль тестирования и валидации Пишутся автоматические тесты: модульные тесты для парсера и генераторов, интеграционные — для проверки корректного результата генерации по ряду типовых YAML-конфигураций. Добавляются проверки на соответствие с эталонными файлами и на корректность в Docker-среде (например, пробный запуск docker-compose up).

**Этап 8.** Документирование и примеры Готовится подробная документация: описание формата входного файла, руководство пользователя CLI, схемы и примеры конфигураций «из коробки» для типовых сценариев (EDW на PostgreSQL→Kafka→ClickHouse→Superset). Включаются рекомендации по расширению и отладке.

**Этап 9.** Пилотное развертывание и сбор обратной связи Инструмент развивается в тестовой среде или локально на реальных примерах, собираются отзывы от пользователей — инженеров и аналитиков. На основе полученных замечаний корректируются шаблоны, схемы и UX CLI.

**Этап 10.** Релиз и сопровождение Формируется релизная сборка, обеспечивается публикация в открытый репозиторий GitHub, настраивается процесс выпуска обновлений и приёма issue. Определяется модель поддержки: дорожная карта, приоритеты новых возможностей и исправлений.

### 3.2. Язык декларативного описания (DSL)

Практический опыт показывает, что повышение уровня абстракции и учёт специфики предметной области наиболее эффективно достигаются через разработку собственного языка предметной области - Domain Specific Language, DSL[30][31]. Такой язык представляет собой формальный аппарат, работающий непосредственно с понятиями и структурами предметной области, позволяя лаконично формулировать и решать большинство типовых задач.

В нашем случае DSL строится на основе YAML[32] – удобного человеко-ориентированного формата сериализации, концептуально схожего с языками разметки, но оптимизированного для записи и чтения распространённых структур данных.

Ключевые особенности синтаксиса YAML:

#### 1. Отступы и вложенность

Используются только пробелы (обычно 2 или 4) для обозначения уровней вложенности. Символ табуляции запрещён.

#### 2. Пары «ключ–значение»

Каждая запись имеет вид ключ: значение, где после двоеточия обязательно идёт пробел.

#### 3. Списки

Элементы маркируются дефисом и пробелом (- элемент).

#### 4. Многострочные литералы

Символ | сохраняет все разрывы строк. Символ > объединяет строки, заменяя отступы и разрывы единичными пробелами.

#### 5. Якоря и ссылки

Якорь (&имя) позволяет дать имя блоку значений. Ссылка (\*имя) повторно вставляет ранее объявленный блок.

Пример:

```
default: &base
имя: Oleg
возраст: 27

user_2:
<<: *base
```

Чтобы формализовать синтаксис DSL и задать конечное описание потенциально бесконечного множества допустимых конфигураций, мы опираемся на контекстно-свободную грамматику  $G = \langle N, T, R, S \rangle$ , где:

$N$  – множество нетерминальных символов

$T$  – терминальные (т. е. реальные лексемы)

$R$  – правило вида  $A \rightarrow \alpha$  (замена нетерминала  $A$  на строку символов  $\alpha$ )

$S$  – стартовый нетерминал.

По классификации Хомского такая грамматика относится ко второму типу (КСГ): в каждом правиле слева стоит ровно один нетерминал, который может быть заменён на любую допустимую цепочку из  $A \cup B$ .

Реализация парсера и генератора AST (абстрактного синтаксического дерева) опирается на ANTLR (ANother Tool for Language Recognition)[33]. Лексические правила (начинаются с большой буквы) описывают, как разбить входной текст на токены. Пример:

```
// Лексическое правило для целых чисел
INT : [0-9]+ ;
```

Синтаксические правила (начинаются с маленькой буквы) задают структуры из токенов. Пример:

```
// Синтаксическое правило для списка аргументов
args : expr (',' expr)* ;
```

Для группировки, повторений и альтернатив в ANTLR применяются:

«()» – группировка

«\*» – 0 или более повторений

«+» – 1 или более

«?» – 0 или 1 раз

«|» – выбор одной из альтернатив

«:» и «;» – разделители начала и конца правил.

### Описание языка DPD (Data Platform Deployer)

Язык DPD разработан для декларативного описания архитектуры платформы данных единым удобным форматом и автоматической генерации всех необходимых

инструментов для быстрого развертывания и тестирования готового стенда. В общих чертах имеет следующую структуру:

```
project:
  name: data-platform-14
  version: 1.2.0
  description: This is a project for testing data platform
sources:
  - type: postgres
    name: postgres_1
  - type: postgres
    name: postgres_2
  - type: s3
    name: s3_1
streaming:
  kafka:
    num_brokers: 3
  connect:
    name: connect-1
storage:
  clickhouse:
    name: clickhouse-1
bi:
  superset:
    name: superset-1
```

В приложении 1 приведена часть полной грамматики, описывающая правила в форме ANTLR

### **3.3. Описание процесса автогенерации конфигураций программных систем платформы данных**

Архитектура инструмента в значительной степени повторяет логическое разбиение на блоки, которое представлено в начале этой главы.



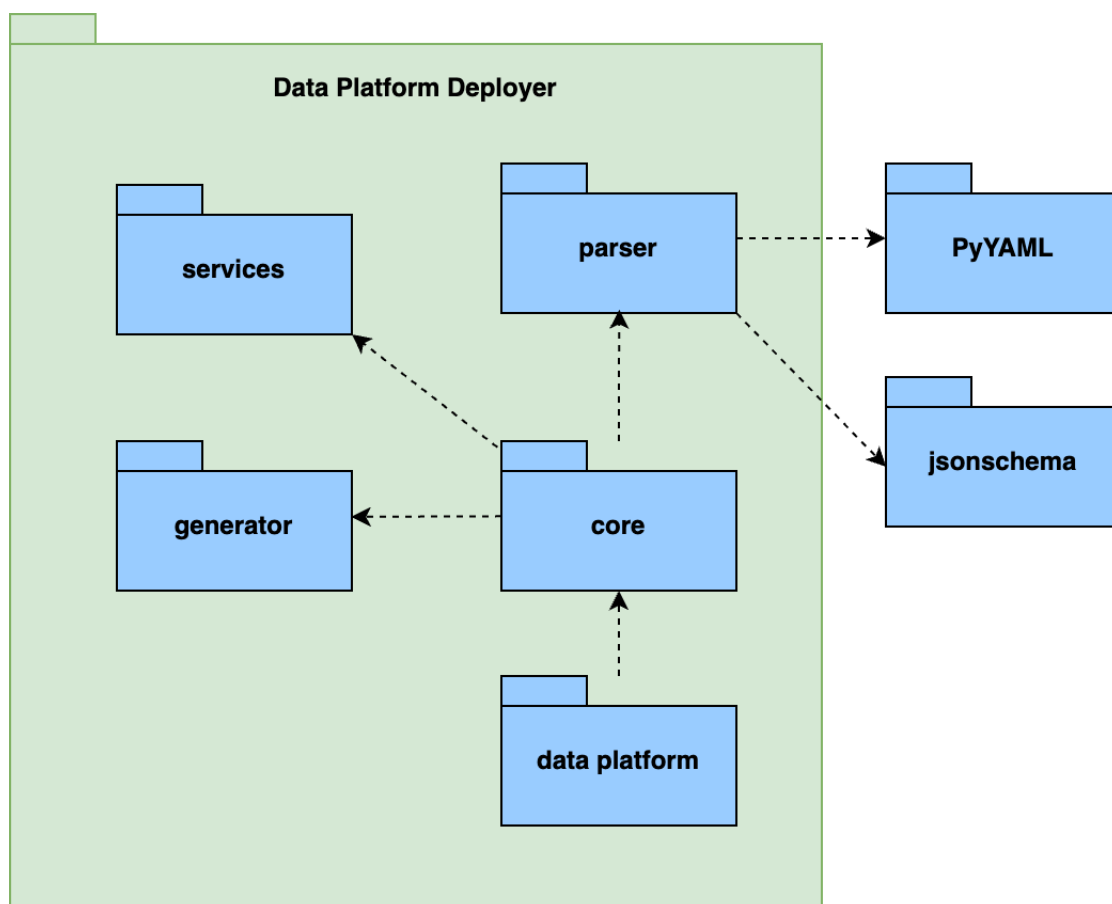


Рис.3.1. Диаграмма пакетов

Пакет `data platform` взаимодействует с пакетом `core`, внутри которого как раз таки находятся парсер, генератор, сервисы, а также другие дополнительные элементы. Диаграмма представлена на рисунке 3.1

Стоит детальнее рассмотреть содержимое пакета `core`, ведь именно там выполняется генерация конфигураций программных систем платформы данных. Процесс автоматической генерации конфигураций для платформы данных с использованием инструмента `dpr` можно разделить на следующие ключевые этапы:

1. Инициация через командную строку (`main.py`)

- **Входная точка.** Пользователь взаимодействует с инструментом через CLI, вызывая основную команду `dpr`.
- **Команда `generate`.** Логика запускается командой `dpr generate` с обязательным параметром `-config <путь_к_YAML>`.
- **Оркестрация.** Файл `main.py` парсит аргументы командной строки, вызывает функции валидации и загрузки конфигурации, инициализирует генератор платформы и запускает процесс, выводя информативные сообщения.

2. Валидация и загрузка конфигурации (`main.py` → `dpr.models`):

- **Проверка схемы.** Перед генерацией выполняется валидация YAML-конфига по JSON-схеме (`src/dpd/schema.json`) с помощью функции `validate` из `dpd.models` и библиотеки `jjsonschema`.
  - **Загрузка и моделирование.** При успешной валидации содержимое конфигурации загружается и преобразуется в Python-модели (`Config`, `Postgres`, `S3` и т. п.) через функцию `load_config_from_file`.
3. Инициализация генератора платформы (`main.py` → `data_platform.py`)
- **Создание DPGenerator.** В конструктор передаётся смоделированная конфигурация (`conf`).
  - **Начальное состояние.** Генератор инициализирует пустые словари для сервисов и настроек, создаёт описание сетей Docker на основе имени проекта, подготавливает `PortManager` и `EnvManager`.
4. Обработка сервисов и делегирование (`DPGenerator.process_services`)
- **Итерация.** Метод перебирает секции конфигурации (`sources`, `streaming`, `storage`, `bi`).
  - **Делегирование.** В зависимости от компонента (`postgres`, `s3`, `kafka`, `clickhouse`, `superset`) вызываются соответствующие статические методы `generate()` из модулей `dpd.services`.
  - **Сборка.** Каждый сервис генерирует свой блок для `docker-compose.yml` и вспомогательные файлы, результат добавляется в словарь `self.services`.
  - **Зависимости.** Для некоторых генераций (например, `KafkaConnectService`) учитываются заранее созданные компоненты (`Postgres`-источники и т. п.).
5. Генерация вспомогательных файлов
- **README.md.** `ReadmeService.generate_file()` создаёт описание платформы и инструкции.
  - **.env.** `EnvManager.generate_env_file()` помещает все секреты (пароли, ключи) в файл `.env`.
  - **init.sql.** `PostgresqlService.generate_init_sql_script()` формирует SQL-скрипт для инициализации (создание публикации, репликационные слоты).

- **postgresql.conf.** `PostgresqlService.generate_conf_file()` генерирует конфигурацию WAL (напр., `wal_level`, `max_wal_senders`, `max_replication_slots`).
- **Конфигурации Debezium и S3SinkConnector.** Функции `generate_debezium_configs()` и `generate_s3sink_configs()` создают JSON-файлы для репликации Postgres→Kafka и Kafka→S3, которые затем загружаются в Kafka Connect через REST API.
- **Плагины для Kafka Connect.** Автоматически скачиваются JAR-файлы `S3SinkConnector` и `ClickHouseConnector`.
- **АКНҚ.** `KafkaUIService.generate_conf_file()` связывает веб-интерфейс АКНҚ с кластерами Kafka и Kafka Connect.

#### 6. Сборка и запись итогового файла

- **Формирование структуры.** Метод `generate()` собирает все настройки, описания сервисов, тома (метод `_generate_volumes()`) и сети в единый Python-словарь, соответствующий формату `docker-compose.yml`.
- **Сериализация.** Структура преобразуется в YAML при помощи `PyYAML`.
- **Запись.** Итоговый YAML записывается в файл `docker-compose.yml` в директории с именем проекта. Директория создаётся автоматически при необходимости.

Таким образом, инструмент `drp` берёт на вход декларативное описание платформы данных, проверяет его, последовательно генерирует конфигурации для каждого компонента и собирает их в готовый к запуску стек под управлением Docker Compose, а также создаёт сопутствующие файлы (`README.md`, `.env`, `postgresql.conf`, `dbz_conf.json`, `s3_sink.json`, `akhq_conf.yml` и др.).

## ГЛАВА 4. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ИНФРАСТРУКТУРЫ ПРОГРАММНЫХ СИСТЕМЫ ДЛЯ РАБОТЫ С БОЛЬШИМИ ДАННЫМИ

Весь процесс проектирования можно иллюстрировать следующей диаграммой активности (рис.4.1)

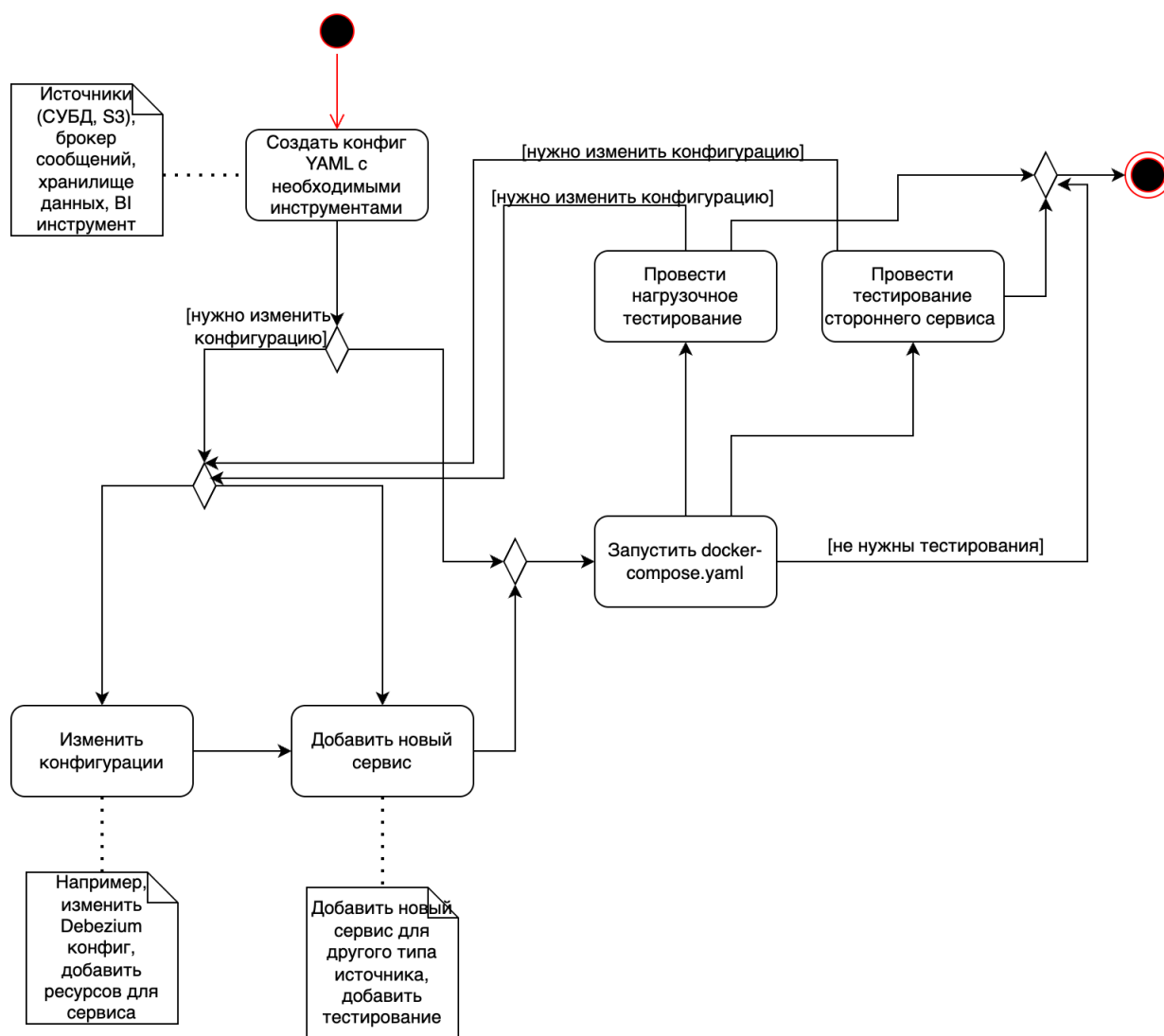


Рис.4.1. Диаграмма деятельности

В данном разделе будут рассмотрены практические примеры использования инструмента для развертывания полноценных платформ данных. На примере двух известных датасетов, Northwind[34] и Chinook[35], будет продемонстрирован весь цикл настройки и работы конвейера данных: от источников до систем хранения и визуализации. Эти примеры иллюстрируют, как с помощью декларативной конфигурации можно быстро построить и запустить сложную инфраструктуру для анализа данных.

#### 4.1. Пример Northwind: Связь клиентов с заказами

Данный пример демонстрирует полный цикл обработки данных с использованием платформы, сгенерированной инструментом `dpr` на основе простой конфигурации. В качестве источника используется классический датасет "Northwind"[34], загруженный в СУБД PostgreSQL.

Цель – показать, как данные из операционной базы данных проходят через систему потоковой обработки Kafka, сохраняются в аналитическом хранилище ClickHouse и визуализируются с помощью BI-инструмента Superset. Параллельно данные также архивируются в S3-совместимое хранилище Minio.

## 1. Конфигурация платформы

Для генерации инфраструктуры использовался следующий конфигурационный файл `config.yaml`:

```
project:
  name: data-platform-northwind
  version: 1.0.0
  description: Northwind end-to-end
sources:
  - type: postgres
    name: postgres_1
  - type: postgres
    name: postgres_2 # Дополнительный источник
  - type: s3
    name: s3_1      # S3-совместимое хранилище (Minio)
streaming:
  kafka:
    num_brokers: 6
  connect:
    name: connect-1
storage:
  clickhouse:
    name: clickhouse-1 # Аналитическое хранилище
bi:
  superset:
    name: superset-1 # BI-инструмент
```

## 2. Развертывание и статус сервисов

После запуска команды `dpd generate --config config.yaml` был создан файл `docker-compose.yml` и сопутствующие конфигурации. Платформа была развернута стандартной командой `docker compose up -d`. Все сервисы (PostgreSQL, Minio, Kafka-брокеры, Kafka Connect, Kafka

UI, ClickHouse, Superset) успешно запустились и работали в штатном режиме(рис.4.2).



























Name	Image	Port(s)
 data-platform-18	-	-
 data_platform_18__kafka_0	<a href="#">bitnami/kafka:latest</a>	<a href="#">59283:9092</a> 
 data_platform_18__kafka_4	<a href="#">bitnami/kafka:latest</a>	<a href="#">59289:9092</a> 
 data_platform_18__kafka_3	<a href="#">bitnami/kafka:latest</a>	<a href="#">59291:9092</a> 
 data_platform_18__kafka_5	<a href="#">bitnami/kafka:latest</a>	<a href="#">59284:9092</a> 
 data_platform_18__kafka_2	<a href="#">bitnami/kafka:latest</a>	<a href="#">59290:9092</a> 
 data_platform_18__kafka_1	<a href="#">bitnami/kafka:latest</a>	<a href="#">59282:9092</a> 
 data_platform_18__clickhouse_1	<a href="#">clickhouse/clickhouse-server</a>	<a href="#">1234:8123</a>  <a href="#">Show all ports (2)</a>
 data_platform_18__minio	<a href="#">minio/minio</a>	<a href="#">9000:9000</a>  <a href="#">Show all ports (2)</a>
 data_platform_18__postgres_2	<a href="#">postgres:15</a>	<a href="#">5433:5432</a> 
 data_platform_18__postgres_1	<a href="#">postgres:15</a>	<a href="#">5432:5432</a> 
 data_platform_18__akhq	<a href="#">tchiotludo/akhq</a>	<a href="#">8086:8080</a> 
 data_platform_18__minio_init	<a href="#">minio/mc:latest</a>	
 data_platform_18__superset_1	<a href="#">apache/superset</a>	<a href="#">8088:8088</a> 

Рис.4.2. Статус сервисов в Docker Desktop

### 3. Поток данных от источника до BI

#### – Источник данных (PostgreSQL)

База данных Northwind была предварительно загружена в экземпляр PostgreSQL. DDL базы Northwind изображена на рисунке 4.3

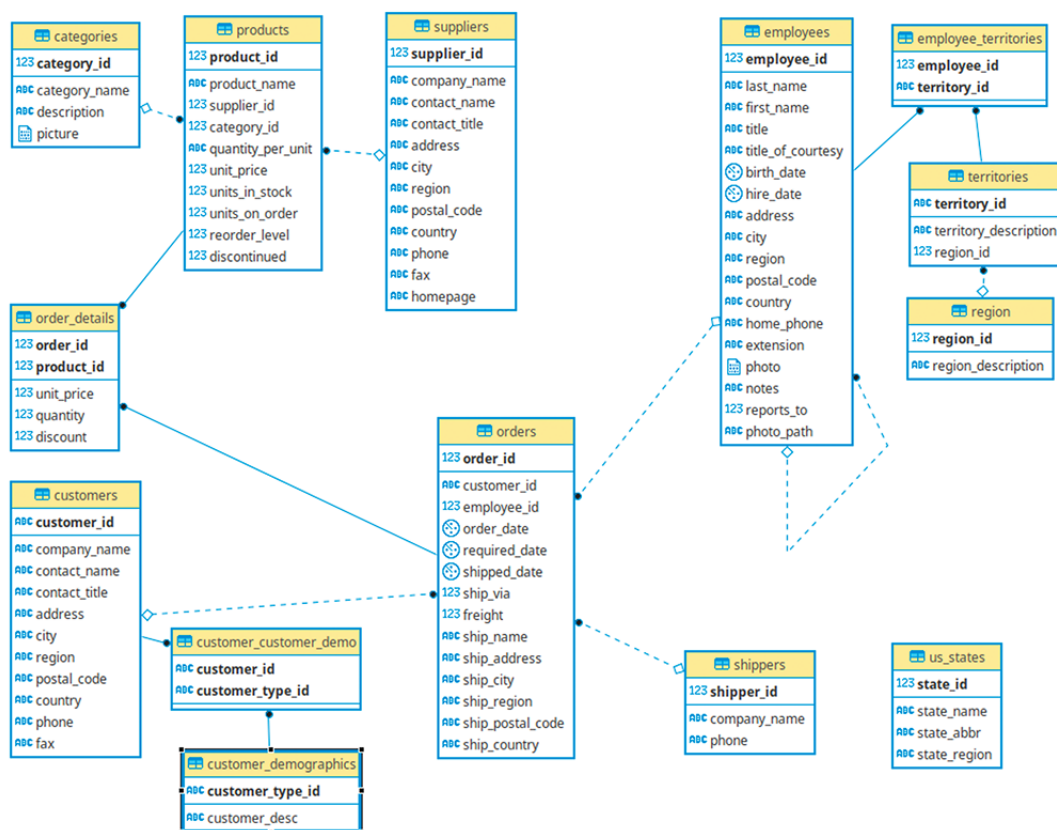


Рис.4.3. DDL схемы Northwind в PostgreSQL

– Захват изменений (Debezium + Kafka Connect):

Для отслеживания изменений (операций INSERT, UPDATE, DELETE) в таблицах PostgreSQL были автоматически созданные коннекторы Debezium PostgreSQL и S3SinkSourceConnector, работающий внутри сервиса Kafka Connect. Коннектор читает WAL (Write-Ahead Log) базы данных и публикует все изменения в виде сообщений в соответствующие топики Apache Kafka. Для каждой таблицы был автоматически создан свой топик.

Topics			
Name	Count	Size	Last Record
connect-configs	≈ 34	14.607 KB	
connect-offsets	≈ 7	1.538 KB	
postgres_1.public.categories	≈ 8	4.393 KB	
postgres_1.public.customers	≈ 91	64.66 KB	
postgres_1.public.employee_territories	≈ 49	24.116 KB	8 minutes ago
postgres_1.public.employees	≈ 9	9.905 KB	8 minutes ago
postgres_1.public.order_details	≈ 2155	1.106 MB	
postgres_1.public.orders	≈ 830	617.563 KB	7 minutes ago
postgres_1.public.products	≈ 77	50.549 KB	
postgres_1.public.region	≈ 4	2.043 KB	
postgres_1.public.shippers	≈ 6	3.096 KB	
postgres_1.public.suppliers	≈ 29	21.338 KB	7 minutes ago
postgres_1.public.territories	≈ 53	27.391 KB	
postgres_1.public.us_states	≈ 51	26.549 KB	

Рис.4.4. Список активных Kafka-топиков в Kafka UI

Name	Type	Tasks
s3sink_postgres_1	S3SinkConnector	172.22.0.15-8083 (0) RUNNING
{"connector.class":"io.confluent.connect.s3.S3SinkConnector","partition.duration.ms":"86400000","topics.dir":"topics","flush.size":"1000","timezone":"UTC","store.url":"http://minio:9000","topics.regex":"postgres_1.*","locale":"ru-RU","aws.secret.access.key":"Ub480QPT046wSw3pkay2pLFnzBMfwu","key.converter.schemas.enable":"false","format.class":"io.confluent.connect.s3.format.json.JsonFormat","partitioner.class":"io.confluent.connect.storage.partitioner.TimeBasedPartitioner","name":"s3sink_postgres_1","value.converter.schemas.enable":"false","aws.access.key.id":"zvmgo3E7xF8m8J68","value.converter":"org.apache.kafka.connect.json.JsonConverter","storage.class":"io.confluent.connect.s3.storage.S3Storage","key.converter":"org.apache.kafka.connect.json.JsonConverter","path.format":"YYYY-MM-dd","timestamp.extractor":"Record","s3.bucket.name":"kafka-topics","rotate.schedule.interval.ms":"300000"}		
s3sink_postgres_2	S3SinkConnector	172.22.0.15-8083 (0) RUNNING
{"connector.class":"io.confluent.connect.s3.S3SinkConnector","partition.duration.ms":"86400000","topics.dir":"topics","flush.size":"1000","timezone":"UTC","store.url":"http://minio:9000","topics.regex":"postgres_2.*","locale":"ru-RU","aws.secret.access.key":"Ub480QPT046wSw3pkay2pLFnzBMfwu","key.converter.schemas.enable":"false","format.class":"io.confluent.connect.s3.format.json.JsonFormat","partitioner.class":"io.confluent.connect.storage.partitioner.TimeBasedPartitioner","name":"s3sink_postgres_2","value.converter.schemas.enable":"false","aws.access.key.id":"zvmgo3E7xF8m8J68","value.converter":"org.apache.kafka.connect.json.JsonConverter","storage.class":"io.confluent.connect.s3.storage.S3Storage","key.converter":"org.apache.kafka.connect.json.JsonConverter","path.format":"YYYY-MM-dd","timestamp.extractor":"Record","s3.bucket.name":"kafka-topics","rotate.schedule.interval.ms":"300000"}		
dbz_postgres_2	PostgresConnector	172.22.0.15-8083 (0) RUNNING
{"connector.class":"io.debezium.connector.postgresql.PostgresConnector","database.user":"postgres_2_admin","database.dbname":"postgres_2_db","slot.name":"debezium_slot","tasks.max":"1","publication.name":"debezium","time.precision.mode":"connect","database.port":"5432","plugin.name":"pgoutput","key.converter.schemas.enable":"false","tombstones.on.delete":"false","topic.prefix":"postgres_2","decimal.handling.mode":"double","replica.identity.autoset.values":"*.x:FULL","database.hostname":"postgres_2","database.password":"ehqTYB6Rc2mNgRum","name":"dbz_postgres_2","value.converter.schemas.enable":"false","value.converter":"org.apache.kafka.connect.json.JsonConverter","key.converter":"org.apache.kafka.connect.json.JsonConverter","snapshot.mode":"never"}		
dbz_postgres_1	PostgresConnector	172.22.0.15-8083 (0) RUNNING
{"connector.class":"io.debezium.connector.postgresql.PostgresConnector","database.user":"postgres_1_admin","database.dbname":"postgres_1_db","slot.name":"debezium_slot","tasks.max":"1","publication.name":"debezium","time.precision.mode":"connect","database.port":"5432","plugin.name":"pgoutput","key.converter.schemas.enable":"false","tombstones.on.delete":"false","topic.prefix":"postgres_1","decimal.handling.mode":"double","replica.identity.autoset.values":"*.x:FULL","database.hostname":"postgres_1","database.password":"Iqj5u8pJVK6QCC","name":"dbz_postgres_1","value.converter.schemas.enable":"false","value.converter":"org.apache.kafka.connect.json.JsonConverter","key.converter":"org.apache.kafka.connect.json.JsonConverter","snapshot.mode":"never"}		

Рис.4.5. Коннекторы в Kafka UI

### – Загрузка в аналитическое хранилище (ClickHouse):

Данные из Kafka доставлялись в ClickHouse с использованием встроенного движка KafkaEngine. Для каждой таблицы источника была создана связка:

1. Таблица на движке KafkaEngine[7], которая подписывается на соответствующий топик Kafka и читает из него сообщения "на лету".



Consumer Group: kafka-ch → default

Topics Members ACLS

Name	Partition	Member	Offset	Metadata	Lag
postgres_1.public.categories	0	-	8	-	0
postgres_1.public.customers	0	-	91	-	0
postgres_1.public.employee_territories	0	-	49	-	0
postgres_1.public.employees	0	-	9	-	0
postgres_1.public.order_details	0	-	2155	-	0
postgres_1.public.orders	0	-	-	-	-
postgres_1.public.products	0	-	77	-	0
postgres_1.public.region	0	-	4	-	0
postgres_1.public.shippers	0	-	6	-	0
postgres_1.public.suppliers	0	-	29	-	0
postgres_1.public.territories	0	-	53	-	0
postgres_1.public.us_states	0	-	51	-	0

Рис.4.6. Группы консьюмеров ClickHouse в Kafka UI

2. Целевая таблица на движке MergeTree[7] для эффективного хранения и аналитических запросов.
3. Материализованное представление, которое автоматически считывает данные из Kafka-таблицы и вставляет их в MergeTree-таблицу, выполняя при необходимости базовые преобразования.

Пример SQL кода для забора данных из Kafka в ClickHouse для таблицы orders находится в приложении 2

#### – Архивация данных в S3

Параллельно с основной обработкой, данные из Kafka-топиков архивировались в S3-хранилище (реализованное через Minio). Kafka Connect S3 Sink Connector считывал сообщения из топиков и сохранял их в виде файлов JSON в соответствующие директории внутри S3 бакета(рис 4.7). Это обеспечивает долговременное хранение сырых данных.

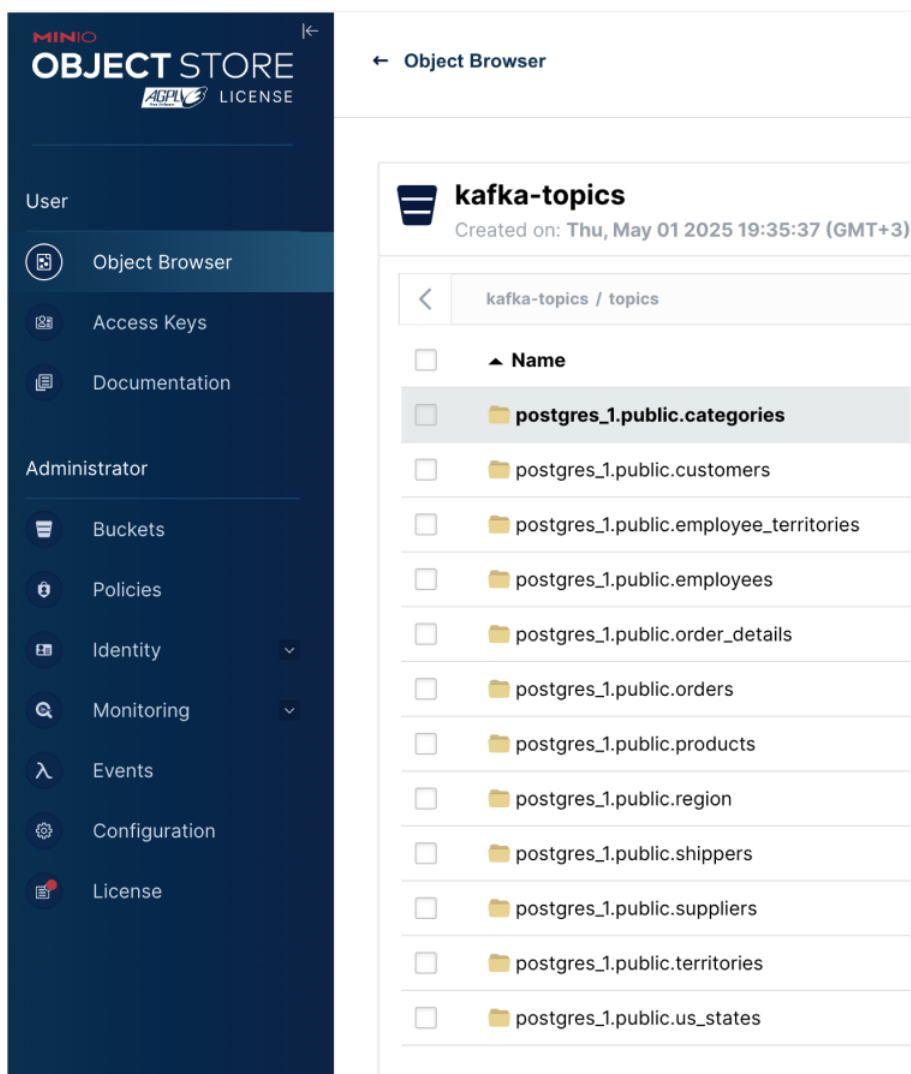


Рис.4.7. Список директорий в S3, соответствующих топикам Kafka

#### – Загрузка и хранение в ClickHouse

Данные о продажах и связанных сущностях доставлялись из Kafka в ClickHouse с использованием стандартного паттерна с предыдущего примера: Kafka Engine таблица для чтения из топика и Materialized View для переноса данных в целевую таблицу на движке MergeTree. Это позволило эффективно хранить данные для аналитических запросов.

#### – Проверка целостности данных

Было проведено сравнение количества записей в ключевых таблицах источника (PostgreSQL) и приемника (ClickHouse). Сравнение показало полное совпадение количества строк, что свидетельствует об успешной и полной доставке данных.

record_count_target			record_count_source		
table_name	record_count		table_name	record_count	
categories	8	1	categories	8	1
customer_customer_demo	0	2	customer_customer_demo	0	2
customer_demographics	0	3	customer_demographics	0	3
customers	91	4	customers	91	4
employee_territories	49	5	employee_territories	49	5
employees	9	6	employees	9	6
order_details	2155	7	order_details	2155	7
orders	830	8	orders	830	8
products	77	9	products	77	9
region	4	10	region	4	10
shippers	6	11	shippers	6	11
suppliers	29	12	suppliers	29	12
territories	53	13	territories	53	13
us_states	51	14	us_states	51	14

Рис.4.8. Сравнение количества строк в PostgreSQL и ClickHouse для Northwind

#### – Анализ и Визуализация (Superset)

Данные, загруженные в ClickHouse, были подключены как источник в Apache Superset. На основе этих данных был построен дашборд, включающий визуализацию, например, график среднего времени обработки заказа по месяцам(рис.4.9). Это демонстрирует готовность данных к анализу и построению отчетности.

## Среднее время обработки заказа

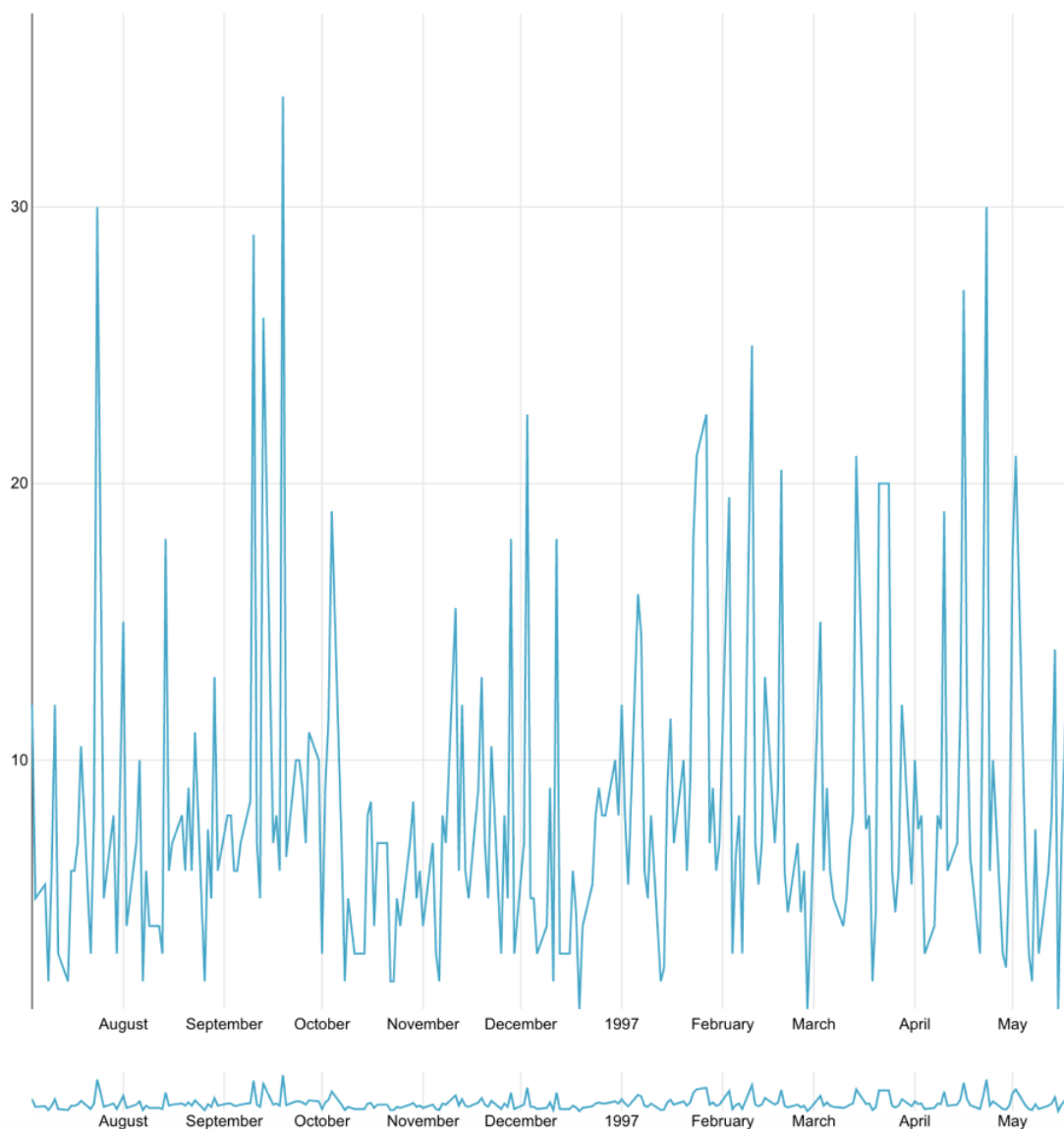


Рис.4.9. Чарт в Superset "Среднее время обработки заказа"

Данный пример успешно продемонстрировал возможность быстрого развертывания комплексной платформы данных с использованием инструмента автоматической генерации платформы данных и одного конфигурационного файла. Был реализован сквозной data pipeline: от захвата изменений в реляционной БД, через потоковую обработку в Kafka, с параллельной выгрузкой в S3, до загрузки в аналитическое хранилище ClickHouse и последующей визуализации в Superset. Проверка целостности данных подтвердила корректность работы всех компонентов пайплайна.

## 4.2. Пример Chinook: Анализ музыкальных продаж

Второй пример демонстрирует применение нашего инструмента для построения аналитического конвейера на основе датасета "Chinook"[35], который моделирует базу данных цифрового музыкального магазина. Цель — отследить поток данных о продажах от операционной базы данных через Kafka до аналитического хранилища ClickHouse и S3-архива, с последующей визуализацией ключевых метрик в Superset.

### 1. Конфигурация платформы

Для генерации инфраструктуры под этот сценарий использовался аналогичный по структуре конфигурационный файл, адаптированный под новый проект и источники:

```
project:
  name: chinehook
  version: 1.0.0
  description: This is a project for chinook
sources:
  - type: postgres
    name: postgres_chinook
  - type: postgres
    name: postgres_2
  - type: s3
    name: s3_1
streaming:
  kafka:
    num_brokers: 6
  connect:
    name: connect-1
storage:
  clickhouse:
    name: clickhouse-1
bi:
  superset:
    name: superset-1
```

## 2. Развертывание платформы

Аналогично первому примеру, команда `dpd generate -config config-chinook.yaml` создала необходимый `docker-compose.yml` и конфигурационные файлы. Запуск `docker compose up -d` успешно развернул все компоненты платформы.

## 3. Поток данных и артефакты

- Источник данных (PostgreSQL) База данных Chinook, DDL которой изображена на рисунке 4.10 была загружена в экземпляр PostgreSQL (`postgres_chinook`).

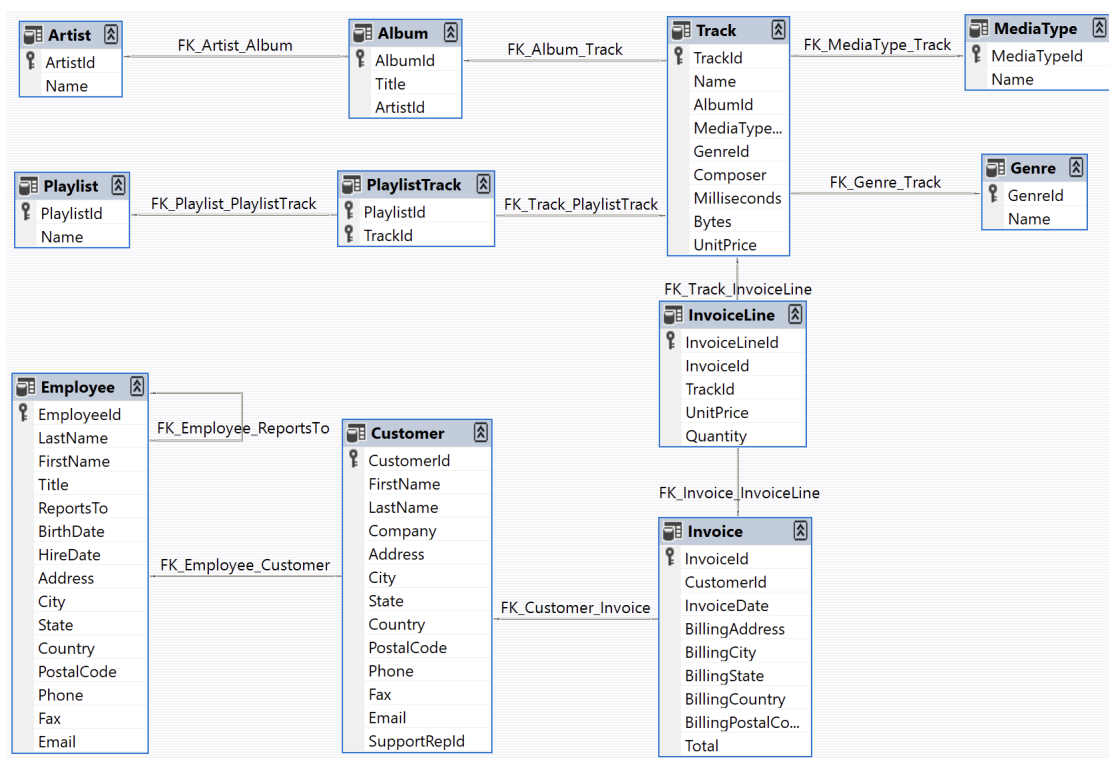


Рис.4.10. DDL схемы Chinook в PostgreSQL

- Захват изменений и публикация в Kafka

С помощью коннектора `Debezium PostgreSQL`, настроенного через `Kafka Connect`, все изменения в таблицах Chinook захватывались из WAL и публиковались в соответствующие топики Kafka (рис.4.11).

Topics			
Name	Count	Size	Last Record
connect-configs	≈ 18	10.259 KB	
connect-offsets	≈ 1	231 B	
postgres_1.public.album	≈ 347	178.561 KB	
postgres_1.public.artist	≈ 275	137.387 KB	
postgres_1.public.customer	≈ 59	42.881 KB	
postgres_1.public.employee	≈ 8	6.418 KB	
postgres_1.public.genre	≈ 25	11.941 KB	37 minutes ago
postgres_1.public.invoice	≈ 412	276.631 KB	37 minutes ago
postgres_1.public.invoice_line	≈ 2240	1.204 MB	37 minutes ago
postgres_1.public.media_type	≈ 5	2.588 KB	
postgres_1.public.playlist	≈ 18	8.922 KB	
postgres_1.public.playlist_track	≈ 8715	4.326 MB	
postgres_1.public.track	≈ 3503	2.139 MB	

Рис.4.11. Список Kafka-топиков в Kafka UI

#### – Архивация данных в S3

Параллельно с основной обработкой, данные из Kafka-топиков архивировались в S3-хранилище (рис.4.12). Kafka Connect S3 Sink Connector считывал сообщения из топиков и сохранял их в виде файлов JSON в соответствующие директории внутри S3 бакета. Это обеспечивает долговременное хранение сырых данных.

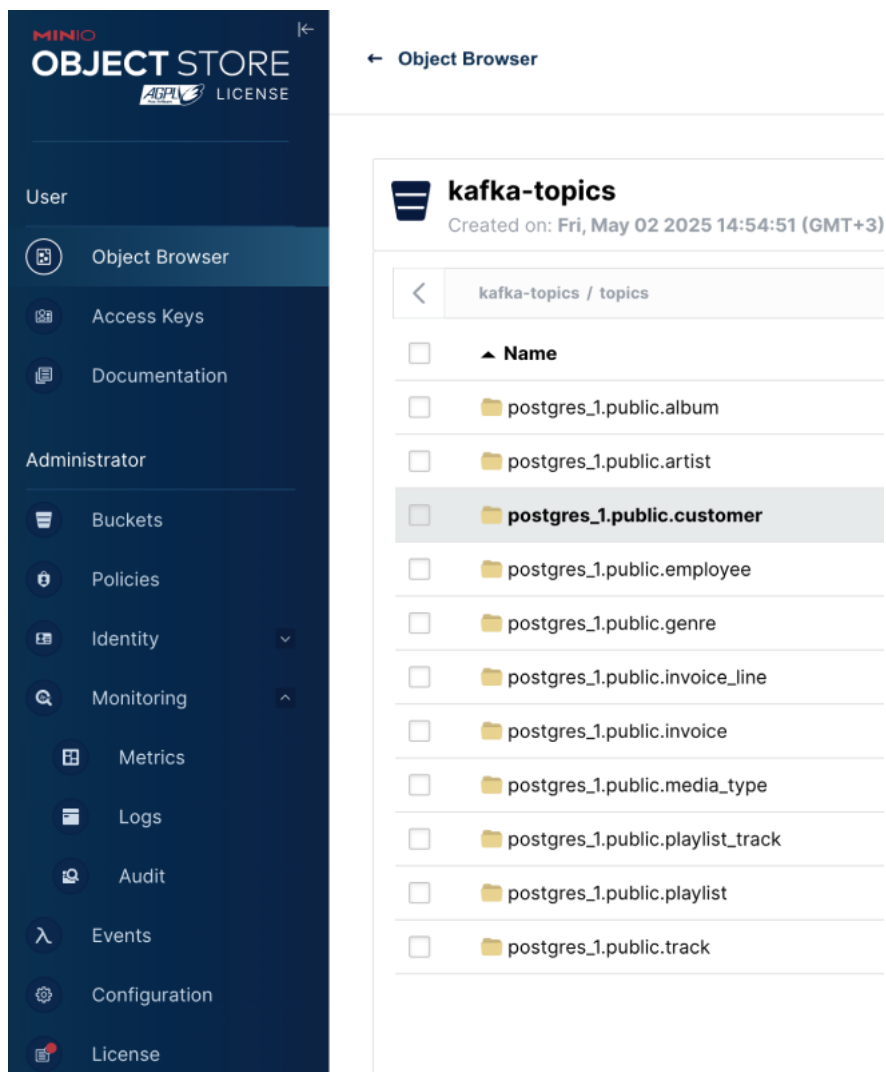


Рис.4.12. Список директорий в S3, соответствующих топикам Kafka

#### – Загрузка и хранение в ClickHouse

Данные о продажах и связанных сущностях доставлялись из Kafka в ClickHouse с использованием стандартного паттерна с предыдущего примера: Kafka Engine таблица для чтения из топика и Materialized View для переноса данных в целевую таблицу на движке MergeTree. Это позволило эффективно хранить данные для аналитических запросов.

#### – Проверка целостности данных

Для подтверждения корректности работы конвейера было выполнено сравнение количества записей в основных таблицах в исходной базе PostgreSQL и в целевых таблицах ClickHouse после завершения загрузки. Результаты сравнения показали идентичное количество строк(рис.4.13).



counts_source			counts_target		
table_name	record_count		table_name	record_count	
album	347	1	1 album	347	
artist	275	2	2 artist	275	
customer	59	3	3 customer	59	
employee	8	4	4 employee	8	
genre	25	5	5 genre	25	
invoice	412	6	6 invoice	412	
invoice_line	2240	7	7 invoice_line	2240	
media_type	5	8	8 media_type	5	
playlist	18	9	9 playlist	18	
playlist_track	8715	10	10 playlist_track	8715	
track	3503	11	11 track	3503	

Рис.4.13. Сравнение количества строк в PostgreSQL и ClickHouse для Chinook

- Анализ и Визуализация (Superset) ClickHouse был подключен как источник данных к Superset. На основе данных о продажах (таблица `invoice_line`), треках (таблица `track`) и жанрах (таблица `genre`), объединенных в ClickHouse, был построен дашборд. Один из ключевых чартов на дашборде отображает количество проданных треков (или сумму продаж) в разрезе музыкальных жанров.

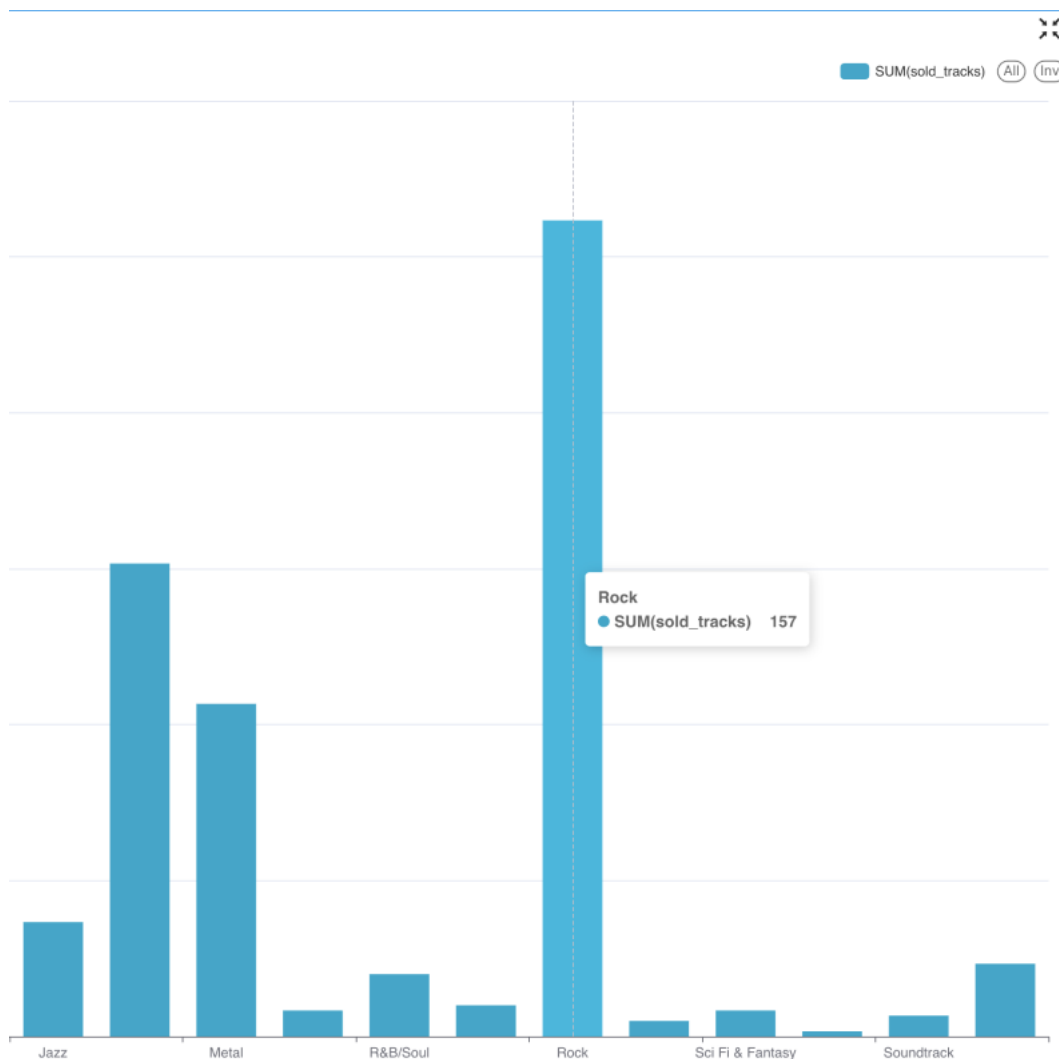


Рис.4.14. Чарт в Superset "Количество продаж по жанрам"

Пример с датасетом Chinoook подтверждает гибкость инструмента в развертывании платформ данных для различных сценариев. Была успешно создана инфраструктура и настроен конвейер для сбора, потоковой обработки, архивирования и аналитической обработки данных о музыкальных продажах. Финальная визуализация в Superset демонстрирует готовность платформы к решению реальных бизнес-задач по анализу данных.

## ГЛАВА 5. ИССЛЕДОВАНИЕ РАЗРАБОТАННОГО ПРОДУКТА

Целью данного раздела является исследование практической применимости и эффективности разработанного программного инструмента dpd (Data Platform Deployer) для автоматической генерации конфигураций элементов инфраструктуры программных систем для работы с большими данными. Учитывая, что

dpd представляет собой инструмент генерации конфигураций и автоматизации процесса развертывания, а не систему, подвергаемую нагрузкам в runtime, традиционные методы количественного нагрузочного тестирования самого инструмента не являются релевантными для оценки его ключевых преимуществ.

Вместо этого, для оценки эффективности и подтверждения выдвигаемых на защиту положений была выбрана методология, основанная на экспертной оценке и апробации инструмента в реальных условиях. Ключевым элементом исследования стала практическая апробация dpd в производственной среде компании ПАО «Магнит», где инструмент использовался для решения конкретных инженерных задач. Результаты данной апробации и экспертные заключения специалистов компании служат основной базой для подтверждения положений, выносимых на защиту. Официальным подтверждением успешной апробации является Акт внедрения (опытной эксплуатации), представленный в Приложении 3.

Исследование направлено на подтверждение следующих положений:

- Эффективность автоматической генерации конфигураций элементов инфраструктуры программных систем для работы с большими данными с помощью разработанного инструмента dpd.
- Значительное снижение трудоемкости и времени развертывания комплексной платформы данных за счет использования разработанного декларативного подхода и инструмента автоматизации dpd.
- Обеспечение корректности, согласованности и воспроизводимости конфигурации взаимосвязанных компонентов платформы данных посредством генерации единых и проверенных артефактов развертывания с помощью dpd.

### **5.1. Подтверждение эффективности автоматической генерации конфигураций**

Первое положение, выносимое на защиту, касается самой основной функции разработанного инструмента dpd – автоматической генерации конфигураций. Инструмент спроектирован таким образом, чтобы на основе единого декларативного файла в формате YAML, описывающего желаемый состав и параметры платформы данных, генерировать весь необходимый набор артефактов:

- Файл `docker-compose.yml` для оркестрации сервисов в контейнерах Docker.

- Скрипты инициализации для баз данных (например, создание пользователей, баз данных, таблиц в PostgreSQL и ClickHouse).
- Конфигурационные файлы для отдельных компонентов, такие как настройки брокера Kafka, параметры коннекторов Kafka Connect (включая Debezium, S3 Sink, и, как было апробировано, Iceberg Sink), конфигурации для Minio и подключения к источникам данных в Apache Superset.

В ходе апробации в ПАО «Магнит», инструмент `drpd` был использован для развертывания тестового стенда, включающего Apache Kafka, Minio и специфический набор коннекторов Kafka Connect, в частности, для тестирования IcebergSinkConnector. Эксперты компании подтвердили, что `drpd` успешно справился с задачей генерации всех необходимых конфигурационных файлов. Было отмечено, что сгенерированные конфигурации были полными, синтаксически корректными и соответствовали параметрам, указанным во входном YAML-файле. Это позволило инженерной команде быстро получить работоспособный стенд без необходимости ручного создания и отладки многочисленных конфигурационных файлов. Данный факт зафиксирован в Акте внедрения (Приложение 3).

Таким образом, практическое применение `drpd` в условиях реальной задачи подтвердило его способность эффективно автоматизировать процесс генерации конфигураций для сложной инфраструктуры больших данных.

## 5.2. Оценка снижения трудоемкости и времени развертывания

Второе положение утверждает, что использование `drpd` приводит к значительному снижению трудоемкости и времени, необходимых для развертывания платформы данных. Эта оценка проводилась на основе экспертного мнения специалистов ПАО «Магнит», участвовавших в апробации, путем сравнения предполагаемых трудозатрат при ручной настройке аналогичного стенда и фактических затрат при использовании `drpd`. Ручное развертывание многокомпонентной платформы данных, включающей PostgreSQL, ClickHouse, Kafka, Kafka Connect с различными коннекторами, Minio и Superset, является сложной и трудоемкой задачей. Она требует:

- Глубоких знаний конфигурационных особенностей каждого компонента.
- Ручного создания и редактирования десятков конфигурационных параметров в различных файлах.
- Тщательной настройки сетевых взаимодействий между сервисами.

- Отладки возможных конфликтов и ошибок, возникающих из-за опечаток или неверных настроек.

По оценкам экспертов ПАО «Магнит», ручная настройка стенда, аналогичного тому, что был развернут для тестирования IcebergSinkConnector, могла бы занять от нескольких часов до целого рабочего дня квалифицированного инженера, особенно если учесть необходимость отладки и проверки корректности связей.

При использовании инструмента `drd` процесс сводился к следующим шагам:

1. Описание требуемой конфигурации стенда в YAML-файле (занимает, в зависимости от сложности, от нескольких минут до получаса для опытного пользователя, знакомого с форматом).
2. Запуск `drd` для генерации артефактов (занимает несколько секунд).
3. Запуск развертывания с помощью сгенерированного `docker-compose.yml` (занимает несколько минут на скачивание образов и запуск контейнеров).

Таким образом, общее время на получение работоспособного стенда с помощью `drd` сократилось до десятков минут. Эксперты ПАО «Магнит» отметили, что основное преимущество `drd` заключается не только в прямом сокращении времени, но и в значительном снижении когнитивной нагрузки на инженера, которому больше не нужно держать в голове все детали конфигурации каждого сервиса. Это позволяет высвободить ресурсы для решения более высокоуровневых задач, таких как непосредственное тестирование функциональности (в данном случае IcebergSinkConnector). Снижение трудоемкости и экономия времени отражены в положительных отзывах, указанных в Акте внедрения (Приложение 3).

### **5.3. Обеспечение корректности, согласованности и воспроизводимости конфигураций**

Третье положение касается качества генерируемых конфигураций, а именно их корректности, согласованности и воспроизводимости.

- Корректность и согласованность: Инструмент `drd` реализует внутреннюю логику, которая обеспечивает согласованность настроек между различными компонентами. Например, адреса и порты сервисов, учетные данные, имена топиков Kafka, пути к хранилищам генерируются таким образом, чтобы все компоненты могли корректно взаимодействовать друг с другом без дополнительной ручной правки. В ходе апробации в ПАО «Магнит» было подтверждено, что развернутый с помощью `drd` стенд функционирует

вал корректно "из коробки": IcebergSinkConnector успешно подключался к Kafka и Minio, используя сгенерированные конфигурации, и корректно обрабатывал данные. Это свидетельствует о том, что drpd генерирует не просто набор файлов, а целостную, работоспособную систему с правильно настроенными взаимосвязями.

- **Воспроизводимость:** Декларативный подход, реализованный в drpd (описание всей инфраструктуры в одном YAML-файле), и автоматизация генерации конфигураций обеспечивают высокую степень воспроизводимости. При наличии одного и того же входного YAML-файла, drpd всегда будет генерировать идентичный набор конфигурационных артефактов. Это критически важно для создания идентичных окружений (например, для разработки, тестирования, демонстраций) и для исключения проблемы "работает на моей машине". Эксперты ПАО «Магнит» отметили, что возможность быстро и гарантированно воспроизводить тестовый стенд является значительным преимуществом, особенно при итеративном тестировании и отладке новых компонентов, таких как IcebergSinkConnector. Возможность многократного развертывания идентичной конфигурации была практически подтверждена в ходе апробации.

Уменьшение вероятности человеческой ошибки – еще один важный аспект, способствующий корректности. Автоматическая генерация на основе проверенной логики и шаблонов исключает опечатки, пропущенные параметры или неверные значения, которые часто возникают при ручной настройке. Эти аспекты также были положительно оценены специалистами ПАО «Магнит» (см. Приложение 3).

#### **5.4. Результаты апробации в ПАО «Магнит» и экспертная оценка**

Как уже упоминалось, ключевым этапом исследования стала апробация инструмента drpd в условиях реальной производственной задачи в компании ПАО «Магнит». Задачей апробации являлось тестирование нового коннектора IcebergSinkConnector для Apache Kafka, предназначенного для сохранения данных из Kafka в таблицы Apache Iceberg, хранящиеся в S3-совместимом хранилище (Minio).

Для решения этой задачи с помощью drpd был оперативно развернут тестовый стенд, включающий следующие компоненты, сконфигурированные для совместной работы:

- Apache Kafka (брокер сообщений).
- Minio (S3-совместимое объектное хранилище для таблиц Iceberg).
- Kafka Connect с развернутым IcebergSinkConnector.
- PostgreSQL как источник данных для Debezium, если бы тестировался полный пайплайн CDC -> Kafka -> Iceberg.

Использование dprd позволило:

1. Быстро создать необходимую инфраструктурную обвязку: Инженеры смогли сосредоточиться непосредственно на логике работы и настройках самого IcebergSinkConnector, а не на развертывании и конфигурировании базовых сервисов.
2. Легко модифицировать конфигурацию стенда: При необходимости изменения параметров Kafka, Minio или других компонентов для различных сценариев тестирования, достаточно было внести правки в YAML-файл и регенерировать конфигурацию, что занимало минимум времени.
3. Гарантировать корректность и изолированность тестового окружения: Это позволило получить надежные результаты тестирования самого коннектора.

По итогам апробации был составлен Акт внедрения (опытной эксплуатации) (Приложение 3), в котором специалисты ПАО «Магнит» подтвердили:

- Успешное применение инструмента dprd для развертывания тестового стенда.
- Значительное сокращение времени и трудозатрат на подготовку инфраструктуры.
- Высокое удобство использования декларативного подхода для описания конфигурации.
- Корректность и согласованность генерируемых конфигураций.
- Практическую пользу инструмента для решения инженерных задач, связанных с тестированием и развертыванием компонентов экосистемы больших данных.

Экспертная оценка со стороны специалистов ПАО «Магнит» однозначно подтвердила состоятельность и полезность разработанного инструмента dprd.



## ЗАКЛЮЧЕНИЕ

В рамках настоящей дипломной работы была поставлена и успешно решена актуальная задача автоматизации процесса развертывания и конфигурирования многокомпонентных платформ для работы с большими данными. Актуальность данной задачи обусловлена возрастающей сложностью современных стеков технологий Big Data, требующих значительных временных и экспертных затрат на их первоначальную настройку и последующую поддержку. Ручной процесс конфигурирования подвержен ошибкам, затрудняет воспроизводимость и масштабирование инфраструктуры.

Основной целью работы являлась разработка программного инструмента, способного автоматизировать генерацию конфигураций и артефактов развертывания для типовых компонентов платформ данных на основе декларативного описания.

В ходе выполнения дипломной работы были достигнуты следующие ключевые результаты:

1. Проведен анализ существующих подходов и инструментов для управления конфигурациями и развертывания инфраструктуры, выявлены их преимущества и недостатки применительно к задачам построения платформ больших данных. Это позволило обосновать необходимость разработки специализированного инструмента.
2. Спроектирован и разработан программный инструмент dpd (Data Platform Deployer). Инструмент использует декларативный подход: пользователь описывает желаемую конфигурацию платформы данных, включая такие компоненты, как PostgreSQL, ClickHouse, Apache Kafka, Kafka Connect с различными коннекторами типа Debezium и S3 Sink, объектное хранилище Minio и систему визуализации Apache Superset, в едином YAML-файле. На основе этого описания dpd автоматически генерирует:
  - Файл docker-compose.yml для оркестрации сервисов с корректной настройкой сетевых взаимодействий и передачей параметров между сервисами.
  - Необходимые конфигурационные файлы для каждого сервиса.
  - Скрипты инициализации баз данных и других компонентов.
3. Реализована поддержка расширяемости за счет модульной архитектуры и использования шаблонизатора, что позволяет в будущем добавлять



поддержку новых компонентов и кастомизировать генерируемые конфигурации.

4. Проведено исследование разработанного продукта, ключевым элементом которого стала апробация инструмента `dpr` в промышленной среде компании ПАО «Магнит». В ходе апробации инструмент использовался для оперативного развертывания тестового стенда с целью тестирования IcebergSinkConnector для Apache Kafka. Результаты апробации, зафиксированные в Акте внедрения (опытной эксплуатации) (Приложение 3), подтвердили практическую применимость и эффективность разработанного решения.

Все положения, выносимые на защиту, были подтверждены в ходе исследования:

- Автоматическая генерация конфигураций элементов инфраструктуры программных систем для работы с большими данными: продемонстрирована способность `dpr` генерировать полный набор корректных артефактов на основе декларативного описания, что было подтверждено в ходе апробации.
- Значительное снижение трудоемкости и времени развертывания комплексной платформы данных: экспертная оценка специалистов ПАО «Магнит» показала, что использование `dpr` существенно сокращает время и усилия, необходимые для подготовки инфраструктуры, по сравнению с ручным подходом.
- Обеспечение корректности, согласованности и воспроизводимости конфигурации взаимосвязанных компонентов платформы данных: практическое применение `dpr` показало, что сгенерированные конфигурации обеспечивают корректное взаимодействие всех компонентов "из коробки", а сам процесс развертывания становится полностью воспроизводимым.

Практическая значимость работы заключается в создании инструмента, который упрощает и ускоряет процесс развертывания платформ больших данных, снижает порог входа для инженеров, уменьшает количество ошибок, связанных с человеческим фактором, и способствует стандартизации конфигураций. Это особенно ценно в условиях динамично развивающихся проектов и при необходимости частого создания тестовых или демонстрационных окружений.

Разработанный инструмент `dpr` обладает потенциалом для дальнейшего развития, включая:

- Расширение списка поддерживаемых компонентов и облачных сервисов.

- Интеграцию с системами CI/CD для полной автоматизации жизненного цикла инфраструктуры.
- Разработку графического пользовательского интерфейса для упрощения описания конфигураций.
- Более глубокую кастомизацию генерируемых скриптов и конфигураций.

Таким образом, цели, поставленные в дипломной работе, были полностью достигнуты. Разработанный программный инструмент `drd` представляет собой законченное решение, обладающее как теоретической новизной в части подхода к автоматической генерации конфигураций на основе формализованной модели, так и высокой практической ценностью, подтвержденной результатами апробации в реальных условиях.

## СЛОВАРЬ ТЕРМИНОВ

**API (Application Programming Interface)** — интерфейс программирования приложений; набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах.

**ANTLR (ANother Tool for Language Recognition)** — генератор парсеров, который используется для создания компиляторов, интерпретаторов и других инструментов, работающих с языками программирования или формальными языками.

**Apache Kafka** — распределённая платформа для обработки потоковых данных, используемая для построения конвейеров данных реального времени и потоковых приложений.

**Apache Superset** — веб-приложение с открытым исходным кодом для исследования и визуализации данных.

**Arenadata** — российская компания, разрабатывающая платформу для сбора, хранения и обработки больших данных на основе технологий с открытым исходным кодом.

**Docker** — программная платформа для быстрой разработки, тестирования и развертывания приложений. Docker упаковывает программное обеспечение в стандартизированные блоки, называемые контейнерами, которые включают все необходимое для работы: библиотеки, системные инструменты, код и среду выполнения.

**Docker Compose** — инструмент для определения и запуска многоконтейнерных приложений Docker. С помощью Compose используется YAML-файл для настройки служб приложения.

**DSL (Domain-Specific Language)** — язык, специализированный для конкретной области применения. В контексте работы, это язык описания конфигурации платформы данных.

**ETL (Extract, Transform, Load)** — извлечение, преобразование, загрузка; один из основных процессов в управлении хранилищами данных, который включает извлечение данных из внешних источников, их преобразование и очистку для соответствия нуждам бизнес-модели и загрузку в хранилище данных.

**IaC (Infrastructure as Code)** — инфраструктура как код; подход к управлению и предоставлению компьютерных центров обработки данных через машиночи-

таемые файлы определений, а не через физическую конфигурацию оборудования или интерактивные инструменты настройки.

**JSON (JavaScript Object Notation)** — текстовый формат обмена данными, основанный на JavaScript. Легко читаем людьми и легко обрабатывается компьютерами.

**Kafka Connect** — фреймворк для надежной потоковой передачи данных между Apache Kafka и другими системами. Используется для создания коннекторов, которые перемещают большие наборы данных в Kafka и из Kafka.

**Minio** — высокопроизводительное распределенное объектное хранилище, совместимое с Amazon S3 API.

**PostgreSQL** — свободная объектно-реляционная система управления базами данных (СУБД).

**S3 (Simple Storage Service)** — сервис простого хранения данных, изначально разработанный Amazon Web Services; стандарт де-факто для API объектных хранилищ.

**SQL (Structured Query Language)** — язык структурированных запросов; декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных.

**Yandex Cloud** — облачная платформа, предоставляемая компанией Яндекс, включающая различные сервисы, в том числе для работы с большими данными.

**YAML (YAML Ain't Markup Language)** — рекурсивный акроним, «YAML — не язык разметки»; дружественный к человеку формат сериализации данных, часто используемый для конфигурационных файлов.

**Аналитика данных (Data Analytics)** — процесс инспектирования, очистки, преобразования и моделирования данных с целью извлечения полезной информации, формирования выводов и поддержки принятия решений.

**Большие данные (Big Data)** — совокупность подходов, инструментов и методов обработки структурированных и неструктурированных данных огромных объемов и значительного многообразия для получения воспринимаемых человеком результатов.

**Брокер сообщений (Message Broker)** — промежуточное программное обеспечение, которое преобразует сообщения из формального протокола обмена сообщениями отправителя в формальный протокол обмена сообщениями получателя.

**Визуализация данных (Data Visualization)** — представление данных в графическом формате для облегчения их восприятия и анализа.

**Генератор конфигураций (Configuration Generator)** — программный инструмент, который автоматически создает конфигурационные файлы для одной или нескольких систем на основе входных параметров или шаблонов.

**Декларативное описание (Declarative Description)** — способ описания системы или процесса, при котором указывается \*что\* должно быть достигнуто, а не \*как\* это сделать.

**Интегрированная платформа данных (Integrated Data Platform)** — комплексное решение, объединяющее различные инструменты и сервисы для сбора, хранения, обработки и анализа данных в единой среде.

**Инфраструктура (Infrastructure)** — совокупность взаимосвязанных обслуживающих структур или объектов, составляющих и/или обеспечивающих основу функционирования системы. В ИТ это физическое и виртуальное оборудование, сети, операционные системы, хранилища данных.

**Коннектор (Connector)** — в Kafka Connect, компонент, отвечающий за интеграцию с конкретным источником или приемником данных.

**Контейнеризация (Containerization)** — метод виртуализации на уровне операционной системы, при котором приложения запускаются в изолированных пространствах, называемых контейнерами.

**Конфигурационный файл (Configuration File)** — файл, используемый для настройки параметров компьютерной программы или операционной системы.

**Метаданные (Metadata)** — данные о данных; информация, описывающая свойства других данных.

**Метамодель (Metamodel)** — модель, описывающая структуру или правила построения других моделей.

**Облачные вычисления (Cloud Computing)** — модель обеспечения повсеместного и удобного сетевого доступа по требованию к общему пулу конфигурируемых вычислительных ресурсов (например, сетям передачи данных, серверам, устройствам хранения данных, приложениям и сервисам), которые могут быть оперативно предоставлены и освобождены с минимальными усилиями по управлению и необходимостью взаимодействия с провайдером.

**Объектное хранилище (Object Storage)** — архитектура хранения данных, которая управляет данными как объектами, в отличие от других архитектур хранения, таких как файловые системы, которые управляют данными как иерархией

файлов, и блочные хранилища, которые управляют данными как блоками внутри секторов и дорожек.

**Оркестрация контейнеров (Container Orchestration)** — автоматизация развертывания, масштабирования и управления контейнеризированными приложениями.

**Парсер (Parser)** — часть компилятора или интерпретатора, отвечающая за синтаксический анализ входной последовательности символов (например, исходного кода) с целью построения структуры данных, обычно дерева разбора или абстрактного синтаксического дерева.

**Платформа данных (Data Platform)** — интегрированный набор технологий, используемых для сбора, хранения, обработки, анализа и управления данными.

**Потоковая обработка данных (Stream Processing)** — парадигма обработки данных, при которой данные обрабатываются непрерывно по мере их поступления, а не пакетами.

**Развертывание (Deployment)** — процесс установки, настройки и активации программного обеспечения или инфраструктуры в целевой среде, делая его доступным для использования.

**Репозиторий (Repository)** — место, где хранятся и поддерживаются какие-либо данные. Часто используется в контексте систем управления версиями (например, Git-репозиторий).

**Скрипт (Script)** — программа или последовательность инструкций, которая автоматизирует выполнение задач.

**СУБД (Система Управления Базами Данных)** — совокупность программных и языковых средств, предназначенных для создания, ведения и совместного использования баз данных многими пользователями.

**Топик (Topic)** — в Apache Kafka, именованная категория или канал, в который продюсеры публикуют сообщения и из которого консьюмеры читают сообщения.

**Управляемый сервис (Managed Service)** — сервис, предоставляемый облачным провайдером, который берет на себя задачи по управлению, обслуживанию и масштабированию базовой инфраструктуры этого сервиса.

**Хранилище данных (Data Warehouse)** — предметно-ориентированная, интегрированная, привязанная ко времени и неизменяемая совокупность данных, предназначенная для поддержки принятия управленческих решений.

**CDC (Change Data Capture)** — захват изменений данных; процесс отслеживания изменений в источнике данных (например, базе данных) и доставки этих изменений в другие системы или хранилища.

**ClickHouse** — быстрая аналитическая СУБД с открытым исходным кодом, работающая на основе столбцового хранения данных.

**dpd (Data Platform Deployer)** — разрабатываемый инструмент для автоматической генерации конфигураций и развертывания платформы данных.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Майер-Шенбергер В., Кузьер К.* Большие данные: Революция, которая изменит как мы живем, работаем и мыслим / пер. с англ. И. Гайдюка. — Москва: МИФ, 2015. — 240 с. — ISBN 978-5-00057-011-7.
2. *Барр М., Эллиот Г.* Данные – новая нефть: как управлять цифровыми активами и извлекать из них максимальную пользу: пер. с рус. — Эксмо, 2022. — 288 с. — ISBN 978-5-04-119234-5.
3. *Таненбаум Э., Стеен М.* Распределенные системы: принципы и парадигмы: пер. с англ. — ДМК Пресс, 2020. — 584 с. — ISBN 978-5-97060-708-4.
4. *Моррис М., Киф М.* Инфраструктура как код: управление серверами в облаке: пер. с англ. — Санкт-Петербург: Питер, 2017. — 336 с. — ISBN 978-5-496-02653-1.
5. *Хамбл Д., Фарли П.* Непрерывная поставка: Надежная автоматизация сборки, тестирования и развертывания программного обеспечения: пер. с англ. — Москва: Вильямс, 2011. — 432 с. — ISBN 978-5-8459-1739-3.
6. Документация PostgreSQL. — URL: <https://www.postgresql.org> (дата обращения: 08.05.2025).
7. Документация ClickHouse [Электронный ресурс]. — URL: <https://clickhouse.com> (дата обращения: 04.05.2025).
8. Amazon Web Services: Что такое объектное хранилище [Электронный ресурс]. — URL: <https://aws.amazon.com/ru/what-is/object-storage/> (дата обращения: 04.05.2025).
9. Документация Kafka [Электронный ресурс]. — URL: <https://kafka.apache.org/intro> (дата обращения: 04.05.2025).
10. Документация Kafka Connect. — URL: <https://docs.confluent.io/platform/current/connect/index.html> (дата обращения: 04.05.2025).
11. Документация Superset [Электронный ресурс]. — URL: <https://superset.apache.org> (дата обращения: 04.05.2025).
12. *Narkhede N., Shapira G., Palino T.* Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale: trans. from the English by. — 2nd. — Sebastopol, CA: O'Reilly Media, 2021. — 464 p. — ISBN 978-1492042218.
13. *Organization J. S.* JSON Schema. — URL: <https://json-schema.org/specification.html> (visited on 04.05.2025).



14. Облачная платформа Yandex Cloud. — URL: <https://yandex.cloud/ru/data-platform> (дата обращения: 25.05.2025).
15. Облачная платформа VK Cloud. — URL: <https://cloud.vk.com/cloud-platform/> (дата обращения: 25.05.2025).
16. Платформа данных Arenadata. — URL: <https://arenadata.tech/products/> (дата обращения: 25.05.2025).
17. Облачный провайдер аналитики CedrusData. — URL: <https://www.cedrusdata.ru> (дата обращения: 25.05.2025).
18. Документация Kubernetes [Электронный ресурс]. — URL: <https://kubernetes.io/docs/home/> (дата обращения: 04.05.2025).
19. *Laney D.* 3D Data Management: Controlling Data Volume, Velocity, and Variety: trans. from the English by. — META Group (Gartner), 2001.
20. *White T.* Hadoop: The Definitive Guide: trans. from the English by. — 4th. — O'Reilly Media, 2015.
21. *Siegel E.* Predictive Analytics: The Power to Predict Who Will Click, Buy, Lie, or Die: trans. from the English by. — Revised and Updated. — 2016.
22. *Кореньков В.* Распределенная система для обработки, хранения и анализа экспериментальных данных Большого адронного коллайдера. — 2012.
23. *Davenport T. H., Patil D. J.* Data Scientist: The Sexiest Job of the 21st Century // Harvard Business Review. — 2012. — Vol. 90, no. 10. — P. 70–76.
24. SmartData 2024: The State of Data, RU Edition. — URL: <https://smartdataconf.ru/archive/2024/talks/726ec3f588224d9998a189429f4dcebe> (дата обращения: 09.09.2025).
25. Википедия ACID. — URL: <https://en.wikipedia.org/wiki/ACID> (дата обращения: 04.05.2025).
26. Википедия Change Data Capture. — URL: [https://en.wikipedia.org/wiki/Change\\_data\\_capture](https://en.wikipedia.org/wiki/Change_data_capture) (дата обращения: 04.05.2025).
27. Документация Debezium. — URL: <https://debezium.io/documentation/reference/stable/index.html> (дата обращения: 04.05.2025).
28. Azure Microsoft: Что такое озеро данных. — URL: <https://azure.microsoft.com/ru-ru/resources/cloud-computing-dictionary/what-is-a-data-lake> (дата обращения: 04.05.2025).
29. Википедия OLAP. — URL: [https://en.wikipedia.org/wiki/Online\\_analytical\\_processing](https://en.wikipedia.org/wiki/Online_analytical_processing) (дата обращения: 04.05.2025).
30. *Новиков Ф. А.* Грамматики и автоматы. — 2023.

31. Ульман Д., Сети Р., Ахо А. Компиляторы: принципы, технологии и инструменты: пер. с англ. — Москва: Вильямс, 2003. — 768 с. — ISBN 5-8459-0189-8.
32. Википедия YAML. — URL: <https://en.wikipedia.org/wiki/YAML> (дата обращения: 04.05.2025).
33. ANTLR Lab: learn, test, and experiment with ANTLR grammars online! — URL: <http://labantlr.org> (visited on 29.05.2024).
34. Northwind sample database for postgres. — URL: [https://github.com/pthom/northwind\\_psql](https://github.com/pthom/northwind_psql) (дата обращения: 04.05.2025).
35. Chinook sample database for postgres. — URL: <https://github.com/josemguerra/Postgres-sql-Chinook> (дата обращения: 04.05.2025).

## Приложение 1

## Грамматика языка DPD

Таблица П1.1

Грамматика языка DPD в форме ANTLR

Грамматика ANTLR4	Комментарий
grammar ConfigDSL;	
configFile : projectDef sourcesDef streamingDef storageDef biDef EOF ;	Корневое правило: определяет общую структуру конфигурационного файла, состоящего из последовательных блоков
projectDef : PROJECT COLON NAME COLON STRING VERSION COLON STRING DESCRIPTION COLON STRING ;	Правило для секции "project": описывает метаданные проекта
sourcesDef : SOURCES COLON sourceItem+ ;	Правило для секции «sources»: определяет список источников данных.

<pre>sourceItem : DASH NAME COLON STRING TYPE COLON sourceType ( PORT COLON NUMBER )? ( USERNAME COLON STRING )? ( PASSWORD COLON STRING )? ( ACCESS_KEY COLON STRING )? ( SECRET_KEY COLON STRING )? ( REGION COLON STRING )? ( BUCKET COLON STRING )? ;</pre>	<p>Правило для описания одного источника данных: имя, тип (Postgres/S3) и опциональные параметры (порт, учётные данные, детали S3).</p>
<pre>sourceType : POSTGRES   S3 ;</pre>	<p>Правило для определения типа источника данных (PostgreSQL или S3).</p>
<pre>streamingDef : STREAMING COLON (kafkaDef   connectDef)+ ;</pre>	<p>Правило для секции «streaming»: задаёт компоненты потоковой обработки (Kafka или Kafka Connect).</p>
<pre>kafkaDef : KAFKA COLON NUM_BROKERS COLON NUMBER ;</pre>	<p>Правило для конфигурации Kafka: число брокеров.</p>
<pre>connectDef : CONNECT COLON NAME COLON STRING ;</pre>	<p>Правило для конфигурации Kafka Connect: имя инстанса.</p>

<pre>storageDef : STORAGE COLON clickhouseDef ;  clickhouseDef : CLICKHOUSE COLON NAME COLON STRING ;</pre>	<p>Правило для секции «storage»: задаёт компонент хранения данных и его параметры (ClickHouse).</p>
<pre>biDef : BI COLON supersetDef ;  supersetDef : SUPERSET COLON NAME COLON STRING ( USERNAME COLON STRING )? ( PASSWORD COLON STRING )? ;</pre>	<p>Правило для секции «bi»: задаёт инструмент BI (Apache Superset) и его опциональные параметры.</p>

PROJECT : 'project';	
SOURCES : 'sources';	
STREAMING : 'streaming';	
STORAGE : 'storage';	
BI : 'bi';	
KAFKA : 'kafka';	
CONNECT : 'connect';	
CLICKHOUSE : 'clickhouse';	
SUPERSET : 'superset';	
NAME : 'name';	
VERSION : 'version';	
DESCRIPTION : 'description';	
TYPE : 'type';	
PORT : 'port';	
USERNAME : 'username';	
PASSWORD : 'password';	
ACCESS_KEY : 'access_key';	
SECRET_KEY : 'secret_key';	
REGION : 'region';	
BUCKET : 'bucket';	
NUM_BROKERS : 'num_brokers';	
POSTGRES : 'postgres';	
S3 : 's3';	
COLON : ':';	
DASH : '-';	
STRING :	
'"' ( ~[\\"]   '\\ ' . ) * ? '"' ;	
NUMBER : [0-9]+ ;	
WS :	
[ \\t\\r\\n]+ -> skip ;	
	Лексемы: ключевые слова, разделители, строковые и числовые литералы, пробельные символы.

## Приложение 2

## SQL код для забора данных из Kafka в ClickHouse

В приложении приведен SQL код для забора данных из Kafka топика в ClickHouse для таблицы orders

Листинг П2.1

## SQL код для забора данных из Kafka в ClickHouse

```

1  CREATE TABLE clickhouse_1_db.kafka_orders (data String)
2  ENGINE = Kafka()
3  SETTINGS kafka_broker_list = 'kafka-0:9092,kafka-1:9092,
   kafka-2:9092,kafka-3:9092,kafka-4:9092,kafka-5:9092',
4      kafka_topic_list    = 'postgres_1.public.orders',
5      kafka_group_name    = 'kafka-ch',
6      kafka_format        = 'JSONAsString';
7
8  CREATE TABLE clickhouse_1_db.orders
9  (
10     order_id            Int16,
11     customer_id         Nullable(String),
12     employee_id         Nullable(Int16),
13     order_date          Nullable(Date),
14     required_date       Nullable(Date),
15     shipped_date         Nullable(Date),
16     ship_via            Nullable(Int16),
17     freight             Nullable(Float32),
18     ship_name           Nullable(String),
19     ship_address        Nullable(String),
20     ship_city           Nullable(String),
21     ship_region         Nullable(String),
22     ship_postal_code    Nullable(String),
23     ship_country        Nullable(String)
24 )
25 ENGINE = MergeTree()
26 ORDER BY order_id;
27
28 CREATE MATERIALIZED VIEW clickhouse_1_db.kafka_orders_mv
29 TO clickhouse_1_db.orders AS
30 SELECT
31     CAST(JSON_VALUE(data, '$.after.order_id')
           Int16)                AS order_id,

```

```

32  CAST(JSON_VALUE(data, '$.after.customer_id')           AS
      Nullable(String)) AS customer_id,
33  CAST(JSON_VALUE(data, '$.after.employee_id')           AS
      Nullable(Int16)) AS employee_id,
34  CAST(toDate(CAST(JSON_VALUE(data, '$.after.order_date')
      AS Nullable(Int64))) AS Nullable(Date)) AS order_date,
35  CAST(toDate(CAST(JSON_VALUE(data, '$.after.required_date')
      AS Nullable(Int64))) AS Nullable(Date)) AS
      required_date,
36  CAST(toDate(CAST(JSON_VALUE(data, '$.after.shipped_date')
      AS Nullable(Int64))) AS Nullable(Date)) AS
      shipped_date,
37  CAST(JSON_VALUE(data, '$.after.ship_via')              AS
      Nullable(Int16)) AS ship_via,
38  CAST(JSON_VALUE(data, '$.after.freight')              AS
      Nullable(Float32)) AS freight,
39  CAST(JSON_VALUE(data, '$.after.ship_name')            AS
      Nullable(String)) AS ship_name,
40  CAST(JSON_VALUE(data, '$.after.ship_address')          AS
      Nullable(String)) AS ship_address,
41  CAST(JSON_VALUE(data, '$.after.ship_city')            AS
      Nullable(String)) AS ship_city,
42  CAST(JSON_VALUE(data, '$.after.ship_region')          AS
      Nullable(String)) AS ship_region,
43  CAST(JSON_VALUE(data, '$.after.ship_postal_code')      AS
      Nullable(String)) AS ship_postal_code,
44  CAST(JSON_VALUE(data, '$.after.ship_country')         AS
      Nullable(String)) AS ship_country
45  FROM clickhouse_1_db.kafka_orders;

```



## Апробация в компанию МАГНИТ

### ПРОТОКОЛ

внедрения и тестирования Kafka Connect Iceberg Sink Connector  
в тестовой среде, развернутой с использованием инструмента dpd

г. Краснодар

«\_\_» \_\_\_\_\_ 2025 г.

### 1. Введение

Настоящий протокол составлен по результатам выполнения рабочей задачи в рамках технологической инфраструктуры ПАО «Магнит».

Исполнитель: \_\_\_\_\_ (Ф.И.О., должность)

Организация: Публичное акционерное общество «Магнит» (ПАО «Магнит»)

### 2. Постановка задачи

В рамках стратегии развития аналитической платформы и внедрения современных технологий хранения и обработки больших данных, перед Исполнителем была поставлена задача по исследованию и тестированию интеграции потоковых данных из Apache Kafka в озеро данных на базе Apache Iceberg.

Основная цель:

Протестировать и оценить работоспособность, производительность и потенциальную применимость коннектора Kafka Connect Iceberg Sink для организации непрерывной доставки данных из корпоративной шины данных Apache Kafka в таблицы формата Apache Iceberg. Это включает в себя настройку передачи данных, проверку их корректности в целевом хранилище и оценку стабильности работы решения.

### 3. Используемый инструментарий и среда тестирования

Для оперативного развертывания необходимой тестовой инфраструктуры был применен разработанный Исполнителем инструмент Data Platform Deployer (далее dpd). С помощью dpd была сгенерирована и развернута тестовая среда, включающая следующие ключевые компоненты:

- Apache Kafka: Кластер из нескольких брокеров для имитации производственной шины данных.
- Kafka Connect: Распределенная платформа для запуска коннекторов, включая тестируемый Iceberg Sink Connector.

- S3-совместимое хранилище (Minio): Для хранения данных таблиц Apache Iceberg, так как Iceberg часто использует объектные хранилища в качестве основного бэкенда.
- Инструменты мониторинга (AKHQ): Для отслеживания состояния топиков Kafka и коннекторов.
- Источник данных (например, PostgreSQL с Debezium): Для генерации тестового потока данных в Kafka, имитирующего бизнес-события.

Использование инструмента dpd позволило значительно сократить время на подготовку и конфигурацию тестового стенда, обеспечив стандартизированную и воспроизводимую среду.

#### 4. Ход выполнения работ

Генерация и развертывание инфраструктуры:

Был подготовлен конфигурационный файл для dpd, описывающий необходимые компоненты (Kafka, Kafka Connect, Minio).

С помощью команды `dpd generate --config <config_file.yaml>` был сгенерирован `docker-compose.yml` и сопутствующие файлы. Платформа была развернута командой `docker compose up -d`.

Настройка Kafka Connect Iceberg Sink Connector:

Тестируемый Iceberg Sink Connector (от Tabular) был загружен на узлы Kafka Connect.

Была создана конфигурация для Iceberg Sink Connector. В конфигурации были указаны:

- Адреса Kafka-брокеров.
- Имена топиков Kafka для чтения данных.
- Параметры подключения к Iceberg-каталогу (например, REST-каталог или Hive Metastore, если используется).
- Параметры S3-хранилища (Minio): эндпоинт, ключи доступа, бакет для хранения данных Iceberg.
- Схема данных и параметры целевых таблиц Iceberg.
- Политики коммита данных в Iceberg по времени

Конфигурация была отправлена в Kafka Connect REST API для запуска коннектора.

Генерация тестового потока данных:

В один или несколько Kafka-топиков, указанных в конфигурации коннектора, подавался тестовый поток сообщений в формате JSON, имитирующий реальные бизнес-события.

Мониторинг и проверка:

Отслеживалось состояние Iceberg Sink Connector через Kafka Connect REST API и/или AKHQ (статус, наличие ошибок, лаг).

Проверялось создание и наполнение таблиц Apache Iceberg в S3-хранилище (Minio) в соответствии с ожидаемой структурой и форматом (например, Parquet или ORC файлы данных, файлы метаданных Iceberg).

Выполнялись тестовые запросы к данным в таблицах Iceberg с использованием совместимого Spark SQL, чтобы убедиться в корректности и доступности записанных данных.

## 5. Результаты тестирования

Развертывание инфраструктуры: Тестовая среда с использованием dprl была успешно и быстро развернута, все компоненты (Kafka, Kafka Connect, Minio) функционировали корректно.

**Containers** [Give feedback](#) [View all your running containers and applications. Learn more](#)

Container CPU usage: 32.64% / 800% (8 CPUs available) Container memory usage: 6.53GB / 7.47GB [Show charts](#)

Q Search [Only show running containers](#)

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	iceberg-data-platform	-	-	-	32.64%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	iceberg_data_platform_ch	aa734e86ac09	<a href="#">clickhouse/clickhouse-server</a>	1234:8123 <a href="#">Show all ports (2)</a>	11.1%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	iceberg_data_platform_kafka_0	4f33ad2f0a92	<a href="#">bitnami/kafka:latest</a>	56337:9092 <a href="#">Show all ports (2)</a>	4.13%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	iceberg_data_platform_superset	a012af49e119	<a href="#">apache/superset</a>	8088:8088 <a href="#">Show all ports (2)</a>	3.31%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	postgres	6f9d3a38a80b	<a href="#">postgres:13</a>		3.13%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	iceberg_data_platform_kafka_1	ae24e38d3dfc	<a href="#">bitnami/kafka:latest</a>	56338:9092 <a href="#">Show all ports (2)</a>	2.62%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	trino	e1160b949807	<a href="#">trinodb/trino</a>	8080:8080 <a href="#">Show all ports (2)</a>	2.41%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	iceberg_data_platform_kafka_2	1c2bbd22f809	<a href="#">bitnami/kafka:latest</a>	56332:9092 <a href="#">Show all ports (2)</a>	2.37%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	iceberg_data_platform_kafka_connect	fa1328e96cfc	<a href="#">debezium/connect:3.0.0.Final</a>	8083:8083 <a href="#">Show all ports (4)</a>	1.86%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	spark-iceberg	284d71a57e0b	<a href="#">tabulario/spark-iceberg:3.5.5-1.8.1</a>	10000:10000 <a href="#">Show all ports (4)</a>	1.43%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	iceberg_data_platform_akhq	0cd5d1abf100b	<a href="#">lchiotudo/akhq</a>	8086:8080 <a href="#">Show all ports (4)</a>	0.13%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	iceberg-rest	14b893a2ef99	<a href="#">tabulario/iceberg-rest</a>	8181:8181 <a href="#">Show all ports (4)</a>	0.11%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	iceberg_data_platform_psql_platform	ec38f97b237	<a href="#">postgres:15</a>	5433:5432 <a href="#">Show all ports (4)</a>	0.04%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>
<input type="checkbox"/>	iceberg_data_platform_minio	a5b38d8daacf	<a href="#">minio/minio</a>	9000:9000 <a href="#">Show all ports (4)</a>	0%	23 minutes ago	<a href="#">Stop</a> <a href="#">Restart</a> <a href="#">Delete</a>

Showing 16 items

Рис. 1 Развернутая инфраструктура в Docker Desktop

Работа коннектора: Kafka Connect Iceberg Sink Connector был успешно запущен и настроен. Коннектор стабильно потреблял данные из указанных Kafka-топиков.

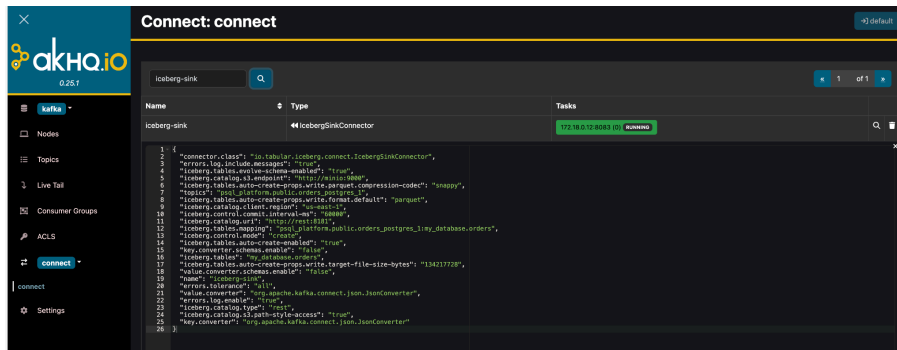


Рис. 2 Конфигурация коннектора IcebergSinkConnector в АКНҚ (Kafka UI)

Передача данных: Данные из Kafka успешно передавались и записывались в таблицы Apache Iceberg. Наблюдалась корректная сериализация/десериализация сообщений и их преобразование в формат, пригодный для Iceberg.

Структура в Iceberg: В S3-хранилище (Minio) корректно создавались директории и файлы, соответствующие структуре таблиц Iceberg (файлы данных, манифесты, файлы метаданных).

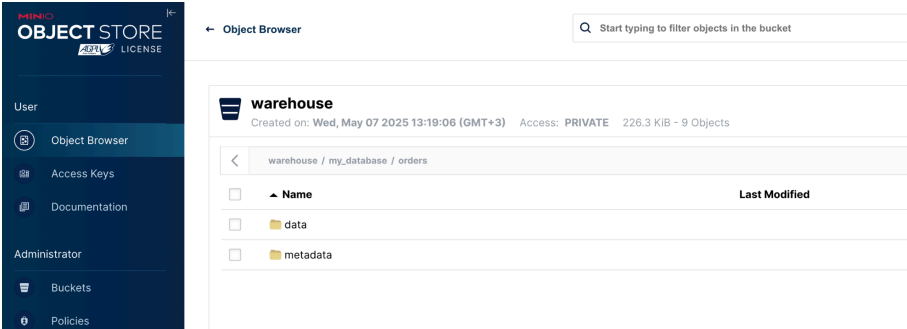


Рис. 3 Структура директории таблицы orders в Minio UI

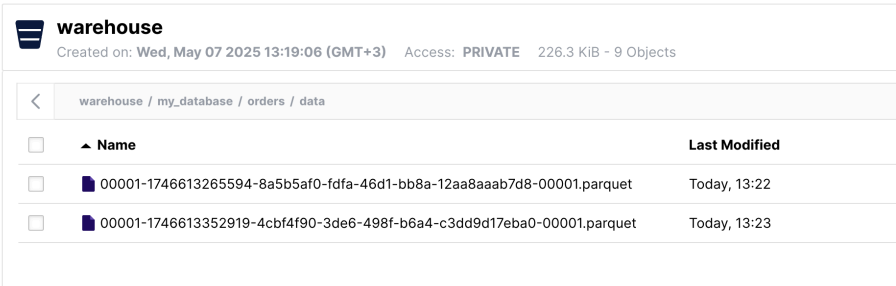


Рис. 4 Содержимое директории data в Minio UI

warehouse

Created on: Wed, May 07 2025 13:19:06 (GMT+3)    Access: PRIVATE    226.3 KiB - 9 Objects

<

warehouse / my\_database / orders / metadata

<input type="checkbox"/>	Name	Last Modified
<input type="checkbox"/>	00000-383492e3-af96-4a91-a8c5-05a5887ca342.metadata.json	Today, 13:21
<input type="checkbox"/>	00001-8273470f-2138-41fa-9bf1-e1356bb73d82.metadata.json	Today, 13:22
<input type="checkbox"/>	00002-8067c36c-2c95-4784-a175-23d19a37702f.metadata.json	Today, 13:23
<input type="checkbox"/>	2c1da788-0088-421f-8e1f-72eeb4598cb7-m0.avro	Today, 13:22
<input type="checkbox"/>	b3c30e59-333e-407e-99a2-85b9e55b65ed-m0.avro	Today, 13:23
<input type="checkbox"/>	snap-6345797814554116362-1-2c1da788-0088-421f-8e1f-72eeb4598cb7.avro	Today, 13:22
<input type="checkbox"/>	snap-8002465153226173367-1-b3c30e59-333e-407e-99a2-85b9e55b65ed.avro	Today, 13:23

Рис. 5 Содержимое директории metadata в Minio UI

Целостность и доступность данных: Выборочная проверка данных, записанных в Iceberg, показала их соответствие исходным данным в Kafka. Данные были успешно прочитаны с помощью аналитических инструментов, совместимых с Iceberg.

[12]:

Visual

SELECT \* FROM my\_database.orders

__lsn	order_date	__table	__db	ship_mode	__deleted	id	customer_id	__op	sales	__source_ts_ms
27245272	1402272000000	orders_postgres_1	psql_platform_db	Standard	false	115812	BH-11710	c	3714.304	1746613350950
27245376	1414886400000	orders_postgres_1	psql_platform_db	Standard	false	115889	SH-20395	c	409.304	1746613350950
27245480	1416787200000	orders_postgres_1	psql_platform_db	First	false	115973	NG-18430	c	2.624	1746613350950
27245576	1405382400000	orders_postgres_1	psql_platform_db	Standard	false	115980	VW-21775	c	9.51	1746613350950
27245680	1406332800000	orders_postgres_1	psql_platform_db	Standard	false	116190	SG-20470	c	256.48	1746613350950
27245784	1393891200000	orders_postgres_1	psql_platform_db	Same Day	false	116239	CL-12565	c	354.9	1746613350950
27245888	1410480000000	orders_postgres_1	psql_platform_db	Second	false	116246	LW-17215	c	3785.292	1746613350950
27245984	1416009600000	orders_postgres_1	psql_platform_db	Standard	false	116407	JF-15190	c	362.176	1746613350950
27246088	1418515200000	orders_postgres_1	psql_platform_db	Standard	false	116568	BM-11785	c	186.304	1746613350950
27246192	1399507200000	orders_postgres_1	psql_platform_db	First	false	116666	KT-16480	c	1799.97	1746613350950

[13]:

Visual

SELECT count(\*) FROM my\_database.orders

[13]:

count(\*)

10018

Рис. 6 Запросы в Iceberg таблицу в SparkSQL

Производительность (качественная оценка): При тестовой нагрузке коннектор демонстрировал приемлемую производительность, лаг потребления из Kafka оставался в допустимых пределах. Учитывая параметер iceberg.control.commit.interval-ms лаг был ненулевой в пределах 3 минут, при лаге 60000 сообщений.

Замечания и наблюдения:

- при использовании catalog.type = hive возникала ошибка

```
lang.IllegalArgumentException: Cannot initialize Catalog
mentation org.apache.iceberg.hive.HiveCatalog: Cannot find
ructor for interface org.apache.iceberg.catalog.Catalog
ng org.apache.iceberg.hive.HiveCatalog
.lang.ClassNotFoundException:
pache.iceberg.hive.HiveCatalog]
g.apache.iceberg.CatalogUtil.loadCatalog(CatalogUtil.java:2
```

Что говорит о конфликтах при работе с hive, поэтому текущая реализацию использует rest

- при попытке посмотреть данные Iceberg из под Trino возникала ошибка

```
rino. spi. TrinoException: Error processing metadata for ta
tabase. orders java. lang. IllegalArgumentException: No fac
ocation: s3:// warehouse/ my_database/ orders/ metadata/
8002465153226173367-1-b3c30e59-333e-407e-99a2-85b9e55b65ed.
```

Чтобы обойти эту ошибку нужно изменить настройки подключения к Iceberg со стороны Trino, конфигурация `trino/iceberg.properties`, реализация с SparkSQL не потребовала точной настройки.

## 6. Заключение и выводы

По результатам проведенного внедрения и тестирования можно сделать следующие выводы:

Технология Kafka Connect Iceberg Sink Connector является работоспособным и перспективным решением для организации потоковой загрузки данных из Apache Kafka в хранилище данных Apache Iceberg.

Протестированная связка компонентов (Kafka -> Kafka Connect -> Iceberg Sink -> Iceberg (на S3)) продемонстрировала стабильную работу и корректную передачу данных в рамках тестовой среды.

Использование разработанного инструмента dpd существенно упростило и ускорило процесс подготовки и развертывания необходимой инфраструктуры для тестирования, что подтверждает его практическую ценность для решения подобных инженерных задач.

Полученные результаты позволяют рекомендовать дальнейшее, более углубленное исследование и пилотирование Kafka Connect Iceberg Sink Connector для возможного применения в продуктивных системах ПАО «Магнит» с целью построения современных озер данных на базе Apache Iceberg.

Настоящий протокол подтверждает факт выполнения поставленной рабочей задачи и успешную апробацию разработанного инструмента dpd в условиях, приближенных к реальным задачам компании.

Подписи:

Исполнитель:

\_\_\_\_\_ / (Ф.И.О.) /  
(подпись)

Ответственное лицо от ПАО «Магнит» (Руководитель):

\_\_\_\_\_ / (Ф.И.О., Должность) /  
(подпись)

«\_\_» \_\_\_\_\_ 2025 г.

М.П. (если применимо)