Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Кафедра информационных технологий и систем управления

ДОПУСТИТЬ К ЗАЩИТЕ ПЕРЕД ГЭК

Зав. кафедрой ИТиСУ

Е.В. Кислицын

(подпись) (Ф.И.О.)

«04» 06 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ИССЛЕДОВАНИЕ РАЗЛИЧНЫХ МЕХАНИЗМОВ ОРКЕСТРАЦИИ РАСЧЕТОВ АНАЛИТИЧЕСКИХ ВИТРИН ДАННЫХ

Научный руководитель: Корелин Иван Андреевич,
к.т.н., доцент

подпись

Нормоконтролер: Бредихина Наталья Сергеевна

подпись

Студент группы: РИМ-220906 Кожин Артём Вадимович

подпись

Екатеринбург
2024

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
**«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»**

Институт радиоэлектроники и информационных технологий - РТФ
Кафедра информационных технологий и систем управления
Направление подготовки 09.04.01 Информатика и вычислительная техника
Образовательная программа Инженерия искусственного интеллекта

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студента Кожина Артёма Вадимовича **группы** РИМ-220906
(фамилия, имя, отчество)

1. Тема выпускной квалификационной работы Исследование различных механизмов оркестрации расчетов аналитических витрин данных

Утверждена распоряжением по институту от «4» декабря 2023 г. № 33.02-05/298

2. Научный руководитель Корелин Иван Андреевич, доцент, кандидат технических наук
(Ф.И.О., должность, ученая степень, ученое звание)

3. Исходные данные к работе
ТЗ от работодателя

4. Перечень демонстрационных материалов *презентация в MS PowerPoint*

5. Календарный план

№ п/п	Наименование этапов выполнения работы	Срок выполнения этапов работы	Отметка о выполнении
1.	Глава 1. Анализ предметной области	до 23.03.2024 г.	выполнено
2.	Глава 2. Проектирование и разработка	до 29.04.2024 г.	выполнено
3.	Глава 3. Анализ работы программы для оркестрации	до 19.05.2024 г.	выполнено
4.	ВКР в целом	до 20.05.2024 г.	выполнено

Научный руководитель Корелин И.А. _____
Ф.И.О. (подпись)

Студент задание принял к исполнению 21.03.24 _____
дата (подпись)

6. Допустить Кожина Артёма Вадимовича к защите выпускной квалификационной работы в экзаменационной комиссии

Зав. кафедрой ИТиСУ _____ Е.В. Кислицын
(подпись) Ф.И.О.

РЕФЕРАТ

Выпускная квалификационная работа магистра 60 стр., 34 рис., 45 источников.

ОРКЕСТРАЦИЯ ДАННЫХ, APACHE AIRFLOW, ПРОЕКТИРОВАНИЕ DAG, РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.

Цель работы – анализ различных механизмов оркестрации данных и разработка программного обеспечения для оркестрации расчетов аналитических витрин данных под нужды компании АО «Банк СИНАРА».

Результатом работы стал выбор инструмента оркестрации расчетов аналитических витрин данных, проектирование и разработка программного обеспечения для оркестрации расчетов аналитических витрин данных.

Область применения полученных результатов – хранилище данных в банке.

Значимость работы заключается в практической реализации программного обеспечения для АО «Банк СИНАРА».

Выпускная квалификационная работа выполнена в текстовом редакторе Microsoft Word.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире данные являются одним из самых ценных ресурсов для любого бизнеса. Они позволяют компаниям принимать обоснованные решения, оптимизировать процессы, выявлять новые возможности для роста и конкурентного преимущества. Однако, объемы данных постоянно растут, и важно иметь систему, способную эффективно обрабатывать и анализировать их.

Обработка данных в информационных системах чаще всего проводится в три этапа: извлечение, трансформация и загрузка (Extract, Transform, Load – ETL). Применяя процесс извлечения, преобразования и загрузки, отдельные необработанные наборы данных могут быть подготовлены в формате и структуре, более пригодных для использования в аналитических целях, что позволяет получить более содержательные выводы. ETL автоматизирует повторяющиеся задачи обработки данных для эффективного анализа. Инструменты ETL автоматизируют процесс миграции данных. В результате инженеры по обработке данных могут больше времени уделять инновациям и меньше – решению таких утомительных задач, как перемещение и форматирование данных. ETL улучшает бизнес-аналитику и аналитику, делая этот процесс более надежным, точным, подробным и эффективным [1 – 3].

Для анализа организации используют аналитические витрины данных как инструмент для эффективного управления и принятия стратегических решений. Аналитические витрины данных стали неотъемлемой частью информационной инфраструктуры компаний, позволяя эффективно обрабатывать и анализировать огромные объемы информации. Витрина данных – это простая форма хранилища данных, ориентированная на одно направление деятельности или тему, например на продажи, финансы или маркетинг. С витриной данных сотрудники могут быстрее получать доступ к данным и статистическим показателям, потому что не нужно тратить время на поиск по более сложному хранилищу данных или вручную собирать данные

из разных источников [4 – 6]. Однако с увеличением данных и усложнением аналитических задач увеличивается и количество ETL-процессов, которые необходимо планировать, отслеживать и перезапускать в случае сбоев – возникает необходимость в оркестраторе [7].

Оркестрация расчетов в аналитических витринах данных представляет собой процесс управления выполнением задач обработки данных, их мониторингом выполнения и управлением возможными сбоями. Существует множество различных механизмов оркестрации расчетов аналитических витрин данных, каждый из которых имеет свои особенности и преимущества. Инструменты оркестрации данных сводят к минимуму ручное вмешательство за счет автоматизации перемещения данных внутри конвейеров данных, они собирают данные из разных мест и организуют их в удобный формат [8 – 9].

Актуальность темы исследования не может быть недооценена в виду стремительного развития информационных технологий и увеличивающихся объемов данных. Без эффективных инструментов для управления процессами обработки данных, компании рискуют потерять конкурентоспособность и возможность принимать обоснованные стратегические решения на основе анализа информации.

Целью выпускной квалификационной работы является исследование различных механизмов оркестрации расчетов аналитических витрин данных и разработка программного обеспечения, с использованием выбранного механизма, для оркестрации расчетов аналитических витрин данных под нужды компании АО «Банк СИНАРА». Для достижения поставленной цели предлагается выполнить следующие задачи:

1) провести обзор исследований связанных с темой работы;
2) выполнить изучение механизмов оркестрации;
3) спроектировать программное обеспечение для оркестрации расчетов аналитических витрин данных;
4) реализовать и протестировать программное обеспечение для оркестрации расчетов аналитических витрин данных;

5) провести анализ работы программного обеспечения для оркестрации, изучить, как улучшилась его работа, по сравнению со старой версией.

Работа состоит из введения, трех глав, заключения и списка литературы.

В первой главе приводится обзор работ, связанных с темой выпускной квалификационной работы, а также анализ и сравнение механизмов оркестрации.

Вторая глава посвящена проектированию программного обеспечения для оркестрации, его разработке и тестированию, содержит описание и анализ функциональных и нефункциональных требований к разрабатываемой программе, диаграмму компонентов, выбора подхода и архитектуры к разрабатываемому программному обеспечению и выбор инструмента для работы с данными.

В третьей главе описывается анализ работы программного обеспечения для оркестрации, происходит его анализ со старой версией.

В заключении приводятся основные результаты работы и рассматриваются дальнейшие пути развития программного обеспечения.

1. Анализ предметной области

В рамках выпускной квалификационной работы необходимо исследовать различные механизмы оркестрации расчетов аналитических витрин данных и разработать программное обеспечение для оркестрации, которое будет работать лучше старого. Для этого в данной главе необходимо познакомиться с архитектурой хранения данных и ознакомиться с используемыми инструментами построения конвейеров данных.

1.1. Обзор работ по тематике исследования

В книге «Data Pipelines with Apache Airflow» подробно рассказывается как разрабатывать конвейеры данных с применением Apache Airflow, также описываются компоненты этого инструмента для достижения поставленных целей [10].

Apache Airflow использует DAG (направленный ациклический граф), который объединяет задачи и указывает, как они должны выполняться.

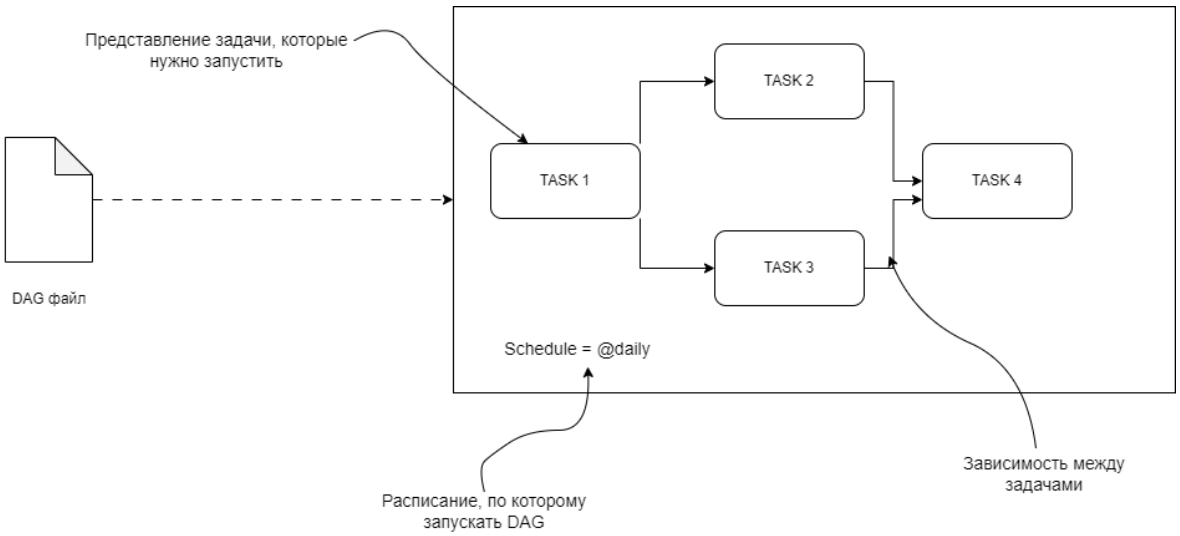

Рисунок 1 – Схема с представлением работы конвейеров в Airflow

На рисунке 1 можно увидеть, что создание конвейера идет в DAG файле, написанном на языке программирования Python. В этом файле описывается вся информация: время запуска, задачи, которые должен выполнить конвейер, связи между ними и др. На рисунке можно увидеть, как между собой связаны задачи, как видно, они последовательны, могут выполняться параллельно, но обязательно не должно быть циклов. На рисунке 2 можно увидеть, что цикличный граф не имеет точки выхода и никогда не закончит свою работу.

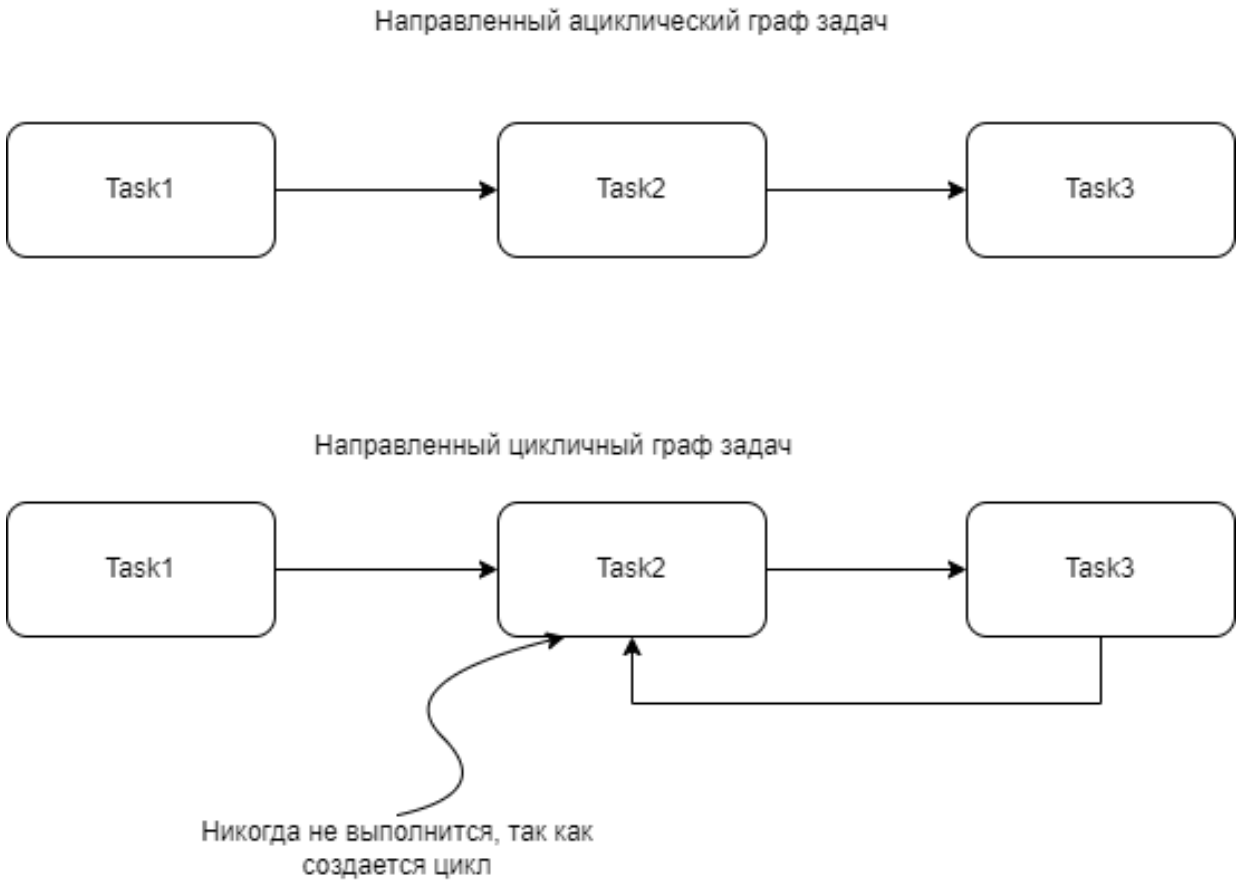

Рисунок 2 – Пример ацикличного и цикличного графов

После определения структуры DAG, Airflow даст возможность определить расписание, которое определяет, когда именно запускать все процессы. Можно ставить запуск единожды, каждую минусу, час и т.д., также можно задать расписание с помощью cron выражений.

В Apache Airflow есть 3 компонента, которые необходимы для работы DAG:

1) The Airflow scheduler – Производит парсинг DAG файлов, проверяет их время запуска и если их время пришло, то планирует задачи на передачу Airflow worker.

2) The Airflow worker – Собирает задачи, которые были запланированы и запускает их.

3) The Airflow webserver – Визуализирует DAG`и и обеспечивает основной интерфейс пользователям для мониторинга выполнения DAG`ов и их результатов.

Наглядную схему того, как происходит работа Apache Airflow можно увидеть на рисунке 3.

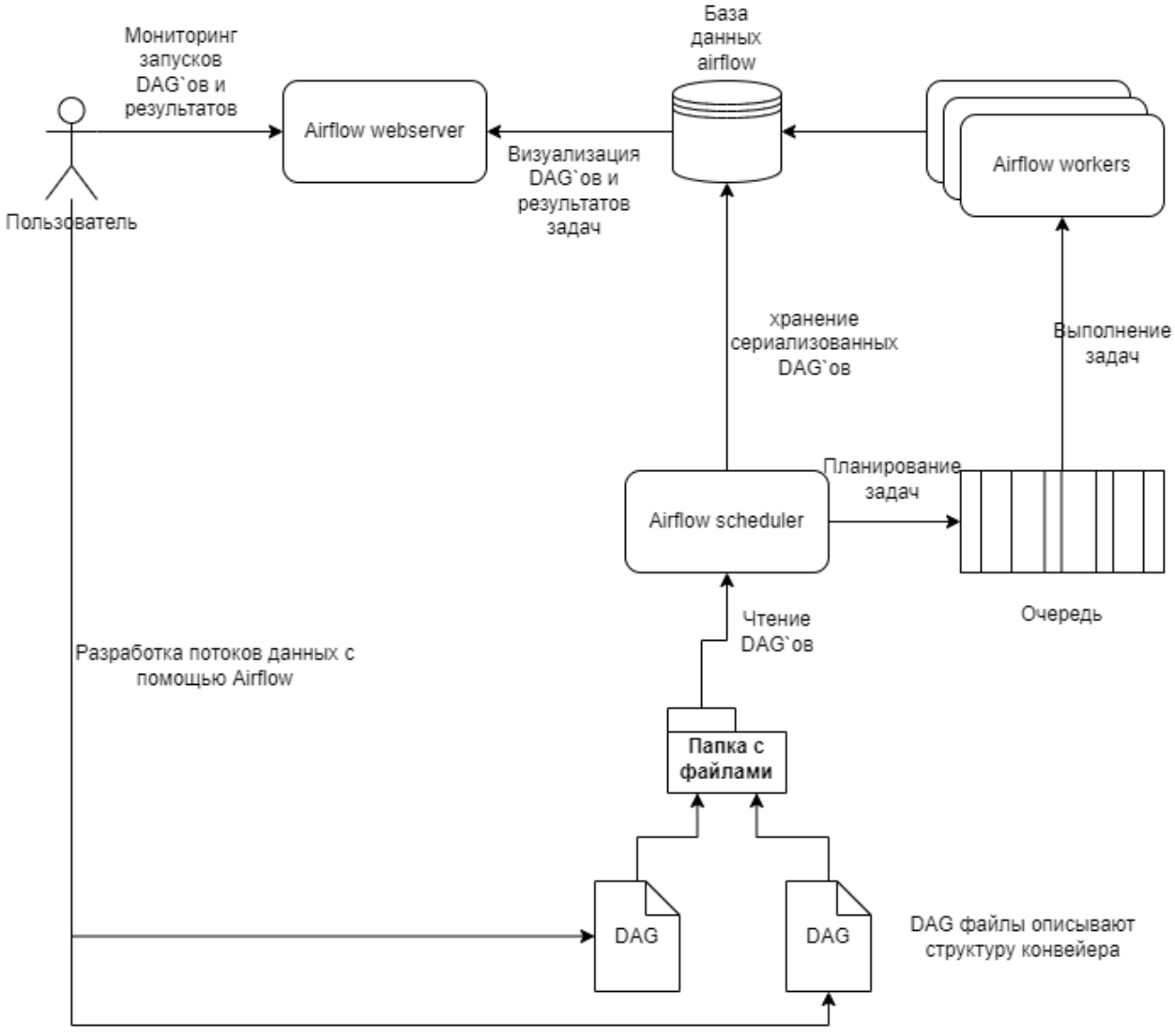

Рисунок 3 – Схема работы Apache Airflow

Одной из важных особенностей семантики планирования Airflow является то, что интервалы по расписанию не только запускают DAG в определенные моменты времени, но и предоставляют подробную информацию о последнем и следующем интервалах по расписанию. Это, по сути, позволяет разделить время на отдельные интервалы (например, каждый день, неделю и т.д.) и запускать DAG для каждого из этих интервалов.

Это свойство расписания интервалов Airflow имеет неоценимое значение для реализации эффективных конвейеров передачи данных, поскольку позволяет создавать инкрементные конвейеры передачи данных. В этих инкрементальных конвейерах при каждом запуске DAG обрабатываются только данные за соответствующий временной интервал (разница в данных) вместо того, чтобы каждый раз повторно обрабатывать весь набор данных. Особенно для больших наборов данных, это может обеспечить значительную экономию времени и средств за счет исключения дорогостоящих повторных вычислений существующих результатов.

Интервалы расписания становятся еще более эффективными в сочетании с концепцией обратной загрузки, которая позволяет создавать новый DAG для исторических интервалов расписания, которые возникали в прошлом. Эта функция позволяет легко создавать (или дополнять) новые наборы данных историческими данными, просто запустив DAG для этих прошлых интервалов расписания. Кроме того, очистив результаты прошлых запусков, также можно использовать эту функцию Airflow для легкого повторного запуска любых исторических задач, если внести изменения в код своей задачи, что позволит при необходимости легко повторно обработать весь набор данных.

В работе «An Overview of Data Vault Methodology and Its Benefits» описываются разные архитектуры в хранилищах данных, показаны этапы от сырых данных до конечных витрин и обзор компонентов Data Vault 2.0 [11].

Data Vault представляет собой альтернативу архитектурам, предложенным Биллом Инмоном и Ральфом Кимбаллом. Концепция

хранилища данных появилась в 1990 году, когда Дэн Линстедт опубликовал несколько статей, касающихся этого метода, и смог предложить окончательную версию архитектуры в 2020 году. В течение 10 лет автор корректировал метод и адаптировал его к текущим потребностям и шаблонам проектирования, создав адаптивный метод под названием Data Vault 2.0.

Каждая архитектура имеет соответствующее описание. Далее приводится обзор вышеупомянутых архитектур.

Архитектура Кимбалла, предложенная в 1996 году, состоит из двухуровневой модели. Исходные данные копируются в промежуточную область, и все преобразования происходят до того, как они попадают на второй уровень, в хранилище данных. Используя этот подход, конечные пользователи будут подключаться к одной и той же модели хранилища данных, и все измерения будут общими. Основное преимущество этого метода заключается в том, что его можно легко реализовать, но он также может быть довольно сложным, если требуется создать вторую модель, используя те же собранные данные, поскольку все нужно загружать заново.

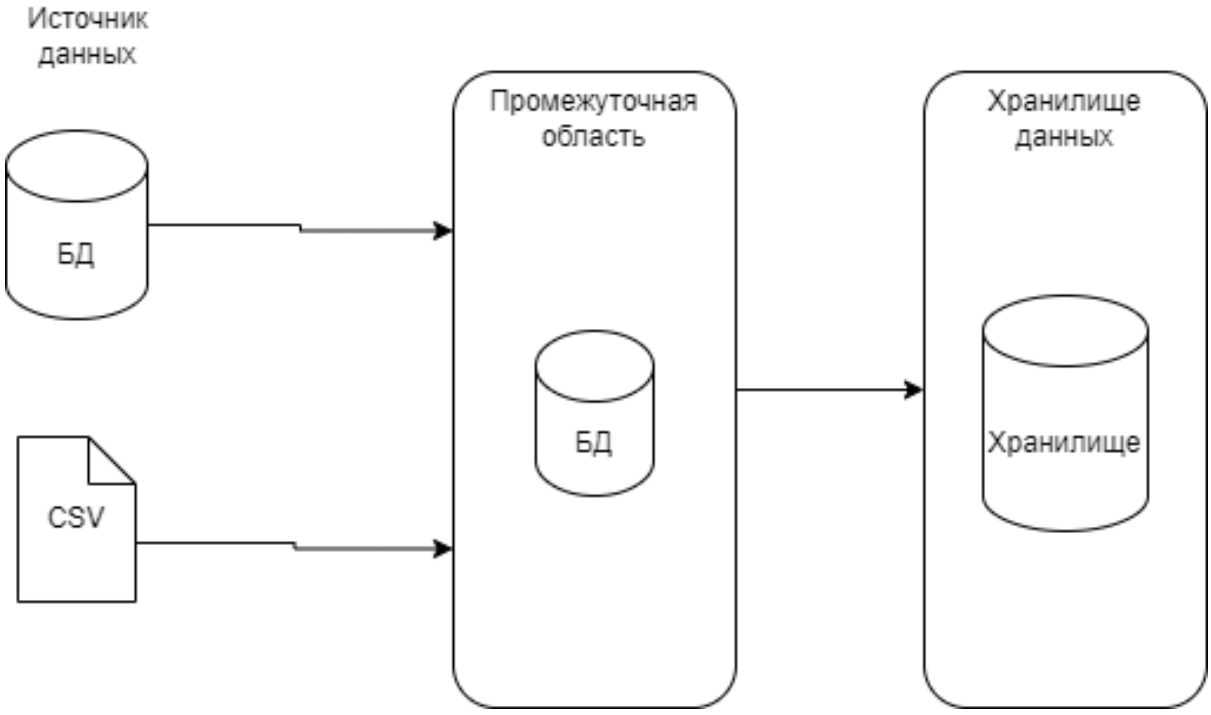

Рисунок 4 – Архитектура Кимбалла

Архитектура Инмона имеет три уровня для решения проблем, описанных ранее, где новый уровень состоит из витрин данных, которые представляют собой мини-хранилища данных, из которых пользователи могут извлекать необходимую информацию. Это также требует дополнительной обработки данных для создания всех витрин данных. Преимущество отделения хранилища данных от витрины данных выгодно в контексте моделей с несколькими вариантами использования. Это также обеспечивает масштабируемость в контексте интеграции новых требований или определения новых витрин данных.

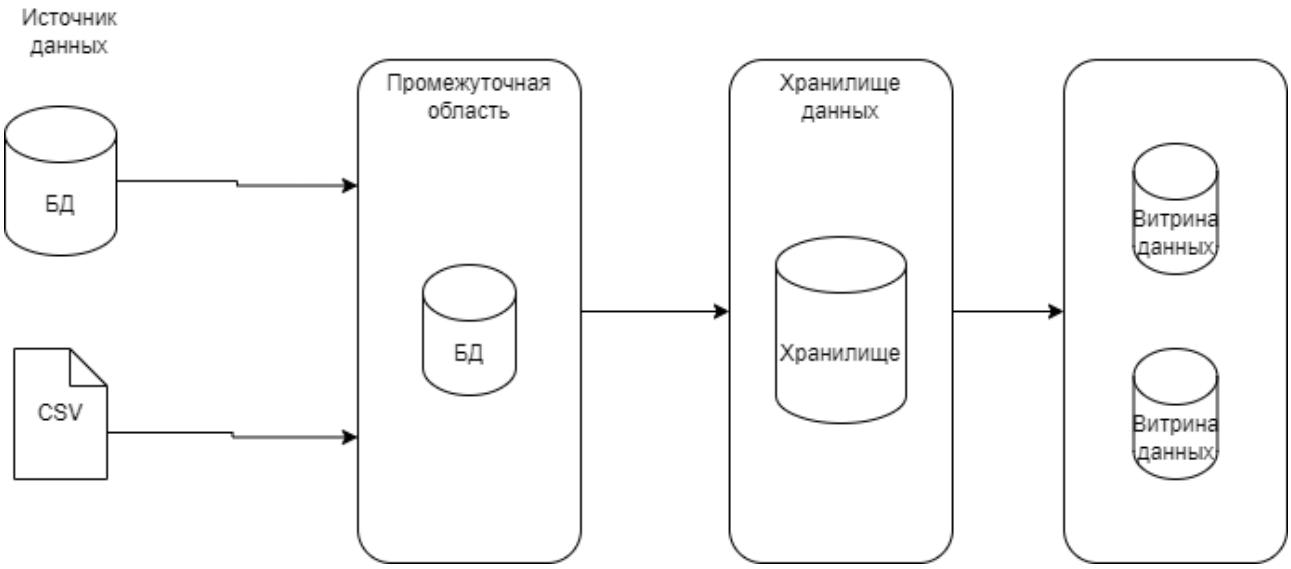

Рисунок 5 – Архитектура Инмона

Data Vault 2.0 был разработан в 2010 году Линстедтом в связи с необходимостью создания новой модели, масштабируемой в контексте больших данных, позволяющей обрабатывать неструктурированные данные и интегрироваться с большим количеством систем. Одним из основных отличий является изменение в использовании хэш-ключей, которые являются первичными ключами вместо последовательных номеров. Это изменение позволяет использовать все хабы и сателлиты, обрабатываться параллельно и устраняет все зависимости на стороне обработки данных. Хэш-ключ вычисляется на основе комбинации столбцов, составляющих бизнес-ключ

таблицы, с использованием различных методов шифрования, таких как SHA-1 или MD5. Более подробный разбор отличий представлен в таблице 1.

Таблица 1 – Сравнение Data Vault 1 и 2 версий

Data Vault 1.0	**Data Vault 2.0**
Порядковые номера для суррогатных ключей.	Использование хэш-ключей для суррогатных ключей.
Также использовался MPP, но только в определенной степени.	Используются платформы в стиле MPP и разработан он с учетом MPP.
Ограниченная поддержка загрузки в режиме реального времени.	Поддерживает работу в режиме реального времени, в облаке, с NoSQL и поддерживает работу с большими данными.
Не очень гибок в отношении автоматизации и виртуализации.	Большое внимание уделяется автоматизации и виртуализации.
Основное внимание уделялось моделированию, и многие концепции моделирования схожи.	Полноценная система бизнес-аналитики. В ней рассказывается обо всем, от концепции до реализации.

Основная цель Data Vault – обеспечить быструю адаптацию к изменениям в случае определения нового бизнес-объекта или использования новых источников. Архитектура имеет 3 уровня:

1) Staging area – промежуточное место в процессе загрузки данных. На этом уровне таблицы повторяют структуру исходных данных. Допускается применение различных бизнес правил, которые никак не влияют на значение данных, например, хэш-ключи, информация о времени загрузки и источнике данных.

2) Область корпоративного хранилища данных – второй уровень архитектуры, разработан с использованием 3 компонентов: хаб, сателлит и ссылка. Этот уровень разбивается на raw vault и business vault. В raw vault данные загружаются из staging area, разбиваясь на Хабы, Ссылки и Сателлиты. В процессе загрузки они никак не агрегируются и не пересчитываются.

Business vault – надстройка над Raw Data Vault. Содержит переработанные результаты. Оптимизирует данные для упрощения формирования витрин.

3) Зона предоставления информации – позволяет конечным пользователям получить доступ к обработанным данным с помощью витрин. Каждая витрина представляет из себя отдельную базу, схему или таблицу, предназначенную для решения задач различных пользователей или отделов.

Data Vault использует 3 компонента, которые упоминались выше: Хабы, Ссылки и Сателлиты.

1) Хаб – основное представление сущности (Клиент, Продукт, Заказ) с позиции бизнеса. Таблица-Хаб содержит одно или несколько полей, отражающих сущность в понятиях бизнеса. В совокупности эти поля называются «бизнес ключ». Идеальный кандидат на звание бизнес-ключа это ИНН организации или VIN номер автомобиля, а сгенерированный системой ID будет наихудшим вариантом. Бизнес ключ всегда должен быть уникальным и неизменным. Хаб так же содержит мета-поля load timestamp и record source, в которых хранятся время первоначальной загрузки сущности в хранилище и ее источник (название системы, базы или файла, откуда данные были загружены). В качестве первичного ключа Хаба рекомендуется использовать MD5 или SHA-1 хеш от бизнес ключа [12].

2) Таблицы-Ссылки связывают несколько хабов связью многие-ко-многим. Она содержит те же метаданные, что и Хаб. Ссылка может быть связана с другой Ссылкой, но такой подход создает проблемы при загрузке, так что лучше выделить одну из Ссылок в отдельный Хаб [12].

3) Все описательные атрибуты Хаба или Ссылки (контекст) помещаются в таблицы-Сателлиты. Помимо контекста Сателлит содержит стандартный набор метаданных (load timestamp и record source) и один и только один ключ «родителя». В Сателлитах можно без проблем хранить историю изменения контекста, каждый раз добавляя новую запись при обновлении контекста в системе-источнике. Для упрощения процесса обновления большого сателлита в таблицу можно добавить поле hash diff: MD5 или SHA-

1 хеш от всех его описательных атрибутов. Для Хаба или Ссылки может быть сколько угодно Сателлитов, обычно контекст разбивается по частоте обновления. Контекст из разных систем-источников принято класть в отдельные Сателлиты [12].

В работе «Exploring Efficient Workflow Frameworks for Data Management» в одной из глав представлено сравнение 2 инструментов оркестрации: Apache Airflow и Dagster [13].

Apache Airflow описывается в данный статье, как надежный инструмент в управлении сложными потоками обработки данных, который имеет одно из преимуществ – масштабируемость. Но, с другой стороны, рассказывается, что инструмент может быть тяжелым для начинающих, т.к. требует хорошего понимания программирования на языке программирования Python и глубокого знания уникальных компонентов Airflow. Также обслуживание масштабируемых систем требует постоянного мониторинга и оптимизации некоторых компонентов, с которыми могут справиться опытные инженеры, а новые пользователи столкнутся с проблемами, требующих значительных затрат времени и ресурсов на их решение.

Dagster версии 0.13.0, напротив, предлагает более удобный подход к управлению потоками данных для новых пользователей. Инструмент делает упор на расширенном опыте локальной разработки и упрощении процесса разработки данных. Основное внимание уделяется простоте использования без ущерба для функциональности. С точки зрения масштабируемости, Dagster предоставляет достаточные возможности для управления растущими потоками обработки данных, хотя это может не соответствовать масштабируемости Airflow в более крупных и сложных средах.

При сравнении этих двух фреймворков становится очевидным, что каждый из них имеет свои явные преимущества.

В статье сравниваются только 2 инструмента оркестрации данных, однако есть и другие инструменты, которые можно использовать. В

следующем подразделе проведем свой анализ и сравнение механизмов оркестрации данных.

1.2.Анализ и сравнение механизмов оркестрации данных

Выбор механизма оркестрации для дальнейшей разработки программного обеспечения зависит от нескольких факторов, таких как наличие запусков по расписанию, присутствие обратной загрузки, чтобы загружать данные за прошедшие дни, использование языка программирования Python и возможность использовать инструмент на бесплатной основе. Ниже приведено описание некоторых из наиболее популярных механизмов оркестрации и их сравнение.

1.2.1. Apache airflow

В настоящее время существует несколько инструментов для автоматизации рабочих процессов.

Airflow– это платформа оркестрации рабочих процессов. Платформа Airflow содержит операторов для подключения ко многим технологиям и легко расширяется для подключения чего-то нового. Операторы – программы, которые выполняют конкретную задачу (например, выполняют bash команды), из них составляются графы, которые называются DAG.

DAG (направленный ациклический граф) – это основная концепция Airflow, объединяющая задачи , организованная с помощью зависимостей и отношений, указывающих, как они должны выполняться [14 – 15].

Богатый пользовательский интерфейс Airflow позволяет легко визуализировать рабочие конвейеры, отслеживать их ход и устранять проблемы при их возникновении. Более того, поскольку Airflow поддерживает python, он может реализовать сложную логику, которая часто требуется при

оркестрации данных в нескольких средах и системах. Использование Airflow языка программирования python также означает, что:

1) Код можно хранить в системе контроля версий, чтобы можно было вернуться к предыдущим версиям.

2) Код может разрабатываться несколькими людьми одновременно.

3) Могут быть написаны тесты для проверки функциональности.

4) Компоненты расширяемы, и вы можете использовать широкую коллекцию существующих компонентов.

Богатая семантика планирования и выполнения позволяет легко определять сложные конвейеры, работающие через регулярные промежутки времени. Обратное заполнение позволяет (повторно) запускать конвейеры на исторических данных после внесения изменений в вашу логику. А возможность перезапустить частичные конвейеры после устранения ошибки помогает максимизировать эффективность.

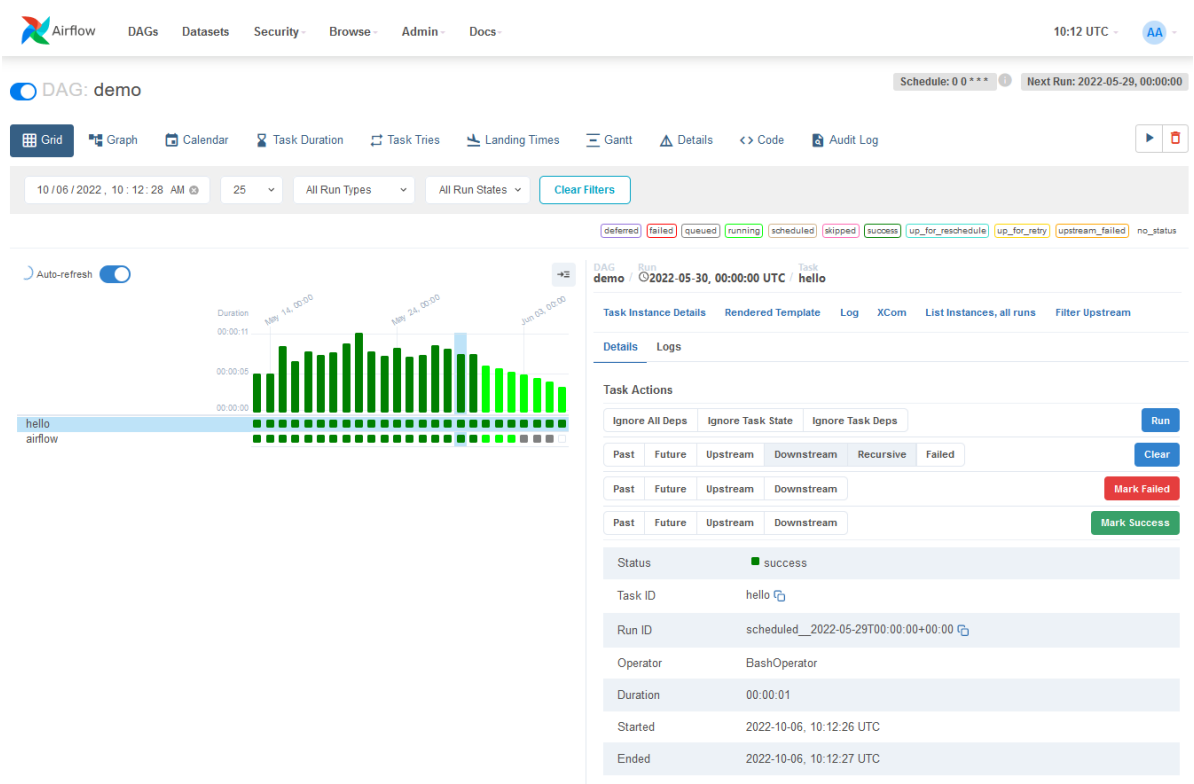

Рисунок 6 – Интерфейс Airflow

Плюсы Airflow:

1) Универсальный для технических пользователей.
2) Расширяемость: добавляйте собственные операторы и плагины.

Минусы Airflow:

1) Пользовательский интерфейс Airflow (UI) может быть сложным для менее технических пользователей.
2) Ресурсоемкость для крупных развертываний

1.2.2. Dagster

Dagster — это оркестратор, предназначенный для разработки и обслуживания данных, таких как таблицы, наборы данных, модели машинного обучения и отчеты.

Вы объявляете функции, которые хотите запустить, и ресурсы данных, которые эти функции создают или обновляют. Затем Dagster поможет запускать функции в нужное время и поддерживать актуальность данных.

Dagster предназначен для использования на каждом этапе жизненного цикла разработки данных, включая локальную разработку, модульные тесты, интеграционные тесты, промежуточные среды и производство [16 – 17].

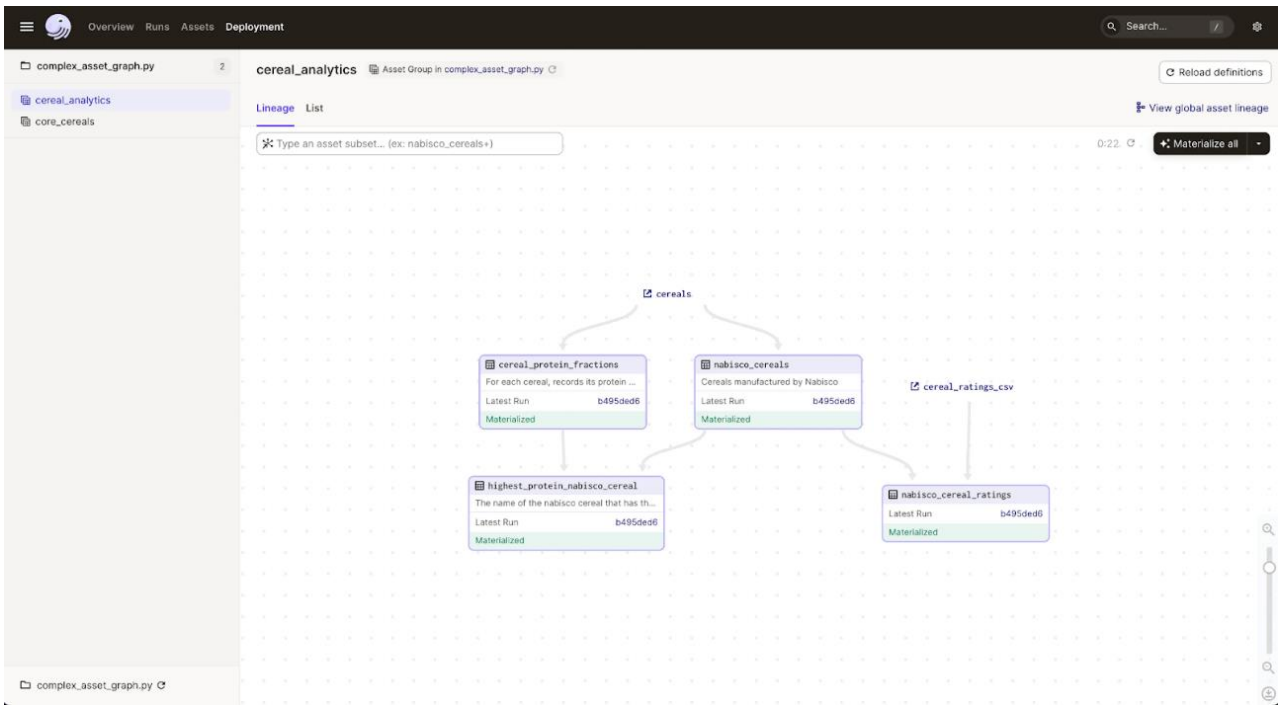

Рисунок 7 – Интерфейс Dagster

1.2.3. Luigi

Luigi — это модуль Python с открытым исходным кодом, который помогает объединить сложные рабочие процессы. Разработанный Spotify, он предназначен для управления взаимозависимыми задачами, фокусируясь на успехе целого, а не отдельных частей [18 – 19].

Плюсы Luigi:

1) Автоматически обрабатывает зависимости задач
2) Предоставляет веб-интерфейс для визуализации конвейера данных.
3) Легко расширяется с помощью пакетов Python.
4) Поддерживает создание динамических задач с параметрами.
5) Задачи также управляются через командную строку.

Минусы Luigi:

1) Требуется хорошее знание Python
2) Не подходит для очень больших и сложных рабочих процессов.
3) Ограниченное количество встроенных оповещений о проблемах с задачами.

4) Редкие обновления по сравнению с конкурентами

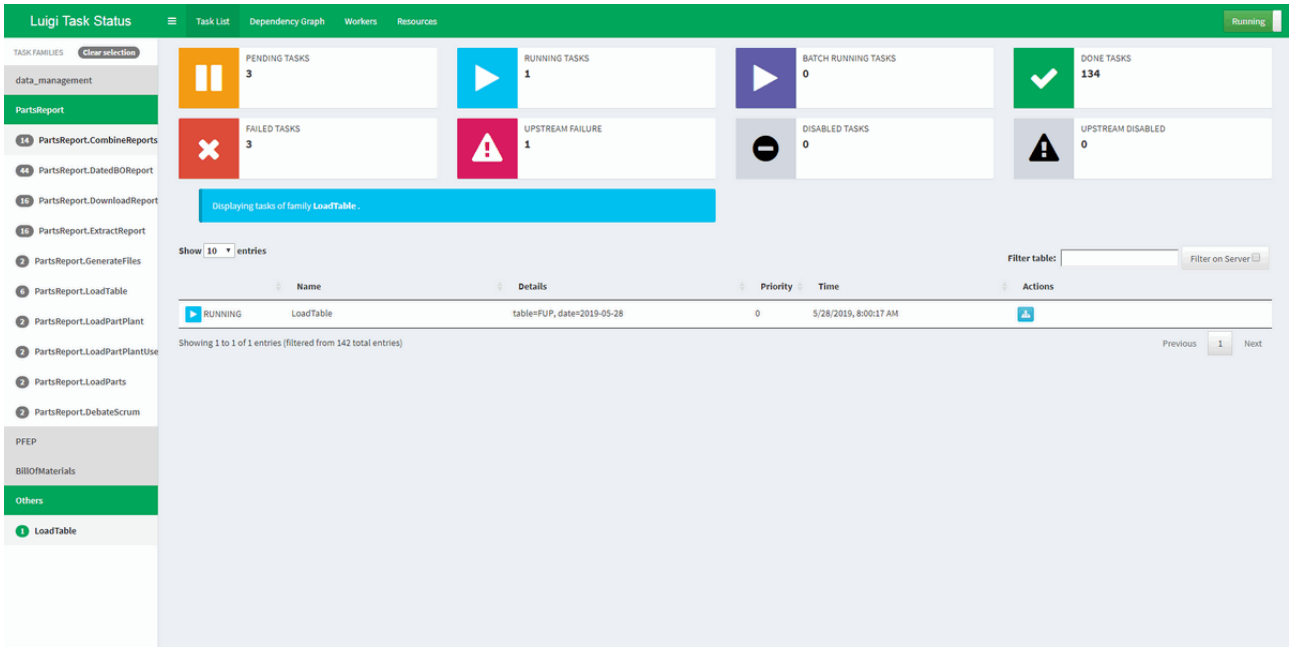

Рисунок 8 – Интерфейс Luigi

1.2.4. Apache NiFi

Решение для обработки потоков данных с открытым исходным кодом Apache NiFi предоставляет веб-интерфейс для проектирования, управления и мониторинга потоков данных. В дополнение к функционированию в качестве инструмента оркестровки данных Apache NiFi обеспечивает планирование заданий, мониторинг и маршрутизацию данных через веб-интерфейс [20].

Благодаря drag-and-drop интерфейсу члены команды обработки данных с небольшим опытом программирования или вообще без него (например, аналитики данных) могут счесть Apache NiFi удобным в использовании. А тот факт, что он также поддерживает широкий спектр протоколов и форматов передачи данных, может сделать его привлекательным для малого бизнеса, который имеет дело с широким спектром источников данных [21 – 22].

Плюсы Apache NiFi:

1) Хорошо подходит для проектирования популярных рабочих процессов.

Минусы Apache NiFi:

1) Плохая общая стабильность

1.2.5. Oozie

Apache Oozie — это решение с открытым исходным кодом, которое предоставляет несколько операционных функций для кластера Hadoop, в частности планирование задач кластера. Это позволяет менеджерам кластеров создавать сложные преобразования данных. Это дает больше контроля над заданиями и позволяет повторять их через заданные интервалы времени [23].

Oozie связан со стеком Hadoop, где YARN является его архитектурным центром, и поддерживает задания Apache MapReduce, Apache Pig, Apache Hive и Apache Sqoop. Он также может планировать специфичные для системы задачи, такие как приложения Java или shell скрипты [24].

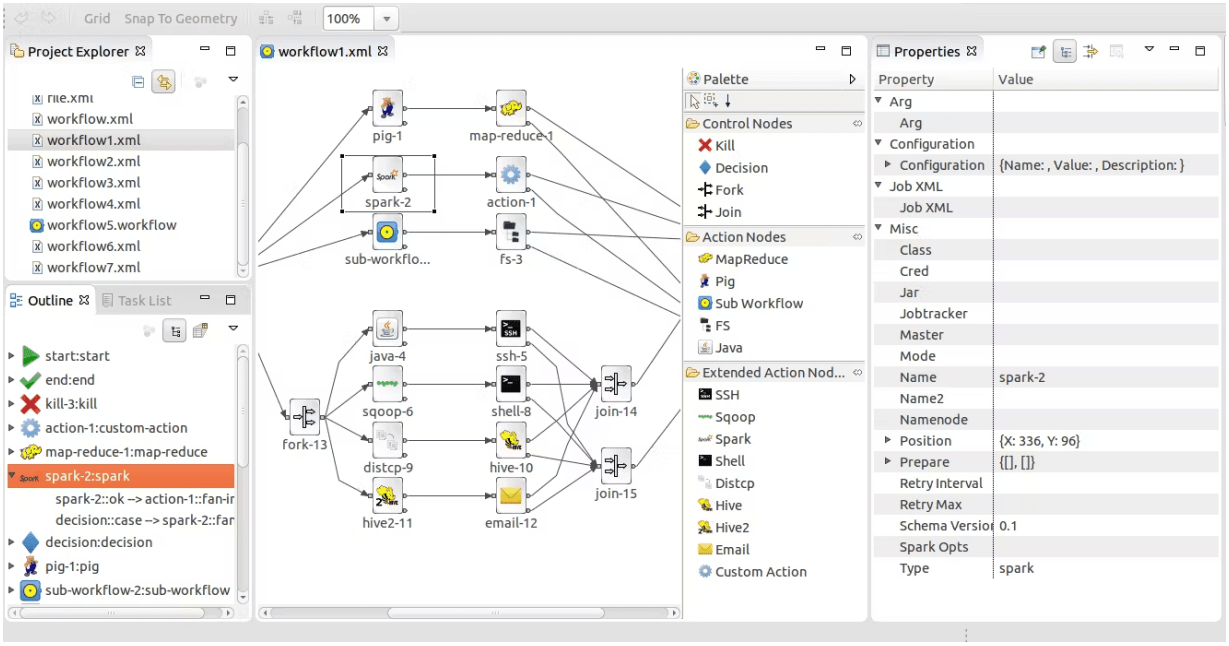

Рисунок 9 – Интерфейс Oozie

1.2.6. Сравнение инструментов

Сравнение инструментов оркестрации представлено в таблице 2.

Таблица 2 – Сравнение инструментов

	Airflow	Dagster	Luigi	NiFi	Oozie
Доступность	Имеет открытый исходный код и бесплатен	Бесплатен. Dagster cloud от 10$	Имеет открытый исходный код и бесплатен	Бесплатен.	Бесплатен
Создание пайплайна	Создание потока происходит с применением кода на языке Python	Конвейеры создаются на основе кода python	Использование Python в качестве языка описания DAG	Создание потока происходит с помощью веб-интерфейса, добавления готовых блоков (процессоров) и соединения их соответствующим образом	Задачи описываются на XML
Встроенное расписание	Есть	Есть	Отсутствует, но можно запускать из cron	Есть	Есть
Обратная загрузка	Есть	Есть	Есть	Нет	Есть

Так как нам необходим бесплатный инструмент, использующий язык программирования Python, имеющий встроенное расписание и обратную загрузку, то для задачи оркестрации расчета аналитических витрин данных лучше всего подходит Airflow и Dagster. Проведем более глубокое исследование этих двух инструментов и определимся с выбором.

1.2.7. Выбор инструмента

Проведем дополнительное исследование и сравним оба инструмента по следующим критериям: поддержка сообщества, тестирование и удобство использования.

Поддержка сообщества очень важно для различных инструментов, так как это позволяет легче и быстрее искать решение проблем, для бизнеса проще найти сотрудников со знаниями более популярных инструментов, а также более популярные инструменты зачастую чаще обновляются и исправляют проблемы инструмента.

По состоянию на июль 2022 года Apache Airflow скачали более 8 миллионов пользователей, график роста показан на рисунке 10 [25].

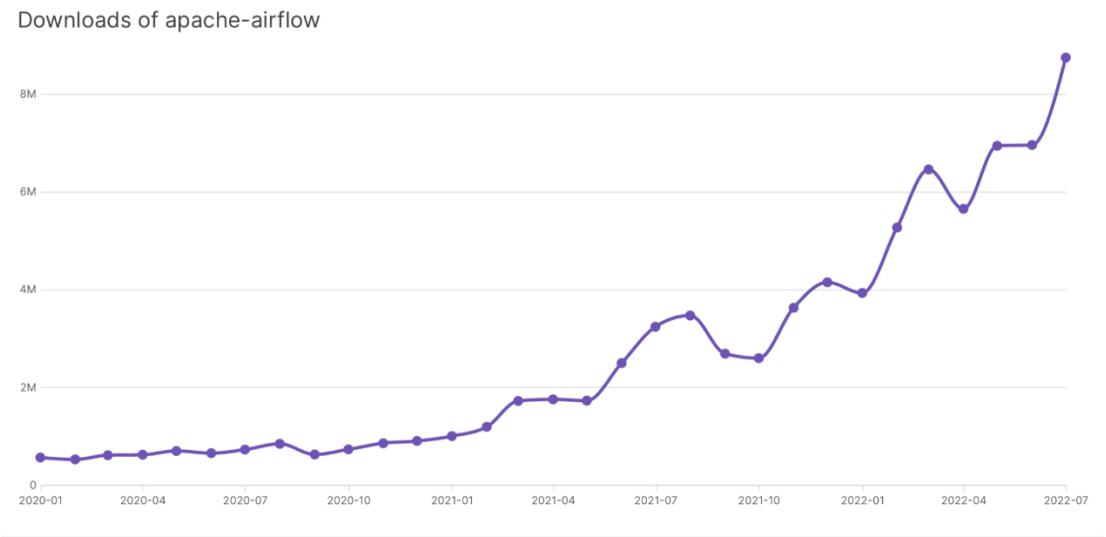

Рисунок 10 – Количество загрузок apache airflow

Такого же графика загрузок для dagster не существует, поэтому обратимся к анализу трафика официального сайта dagster, который приведен на рисунке 11 [26].

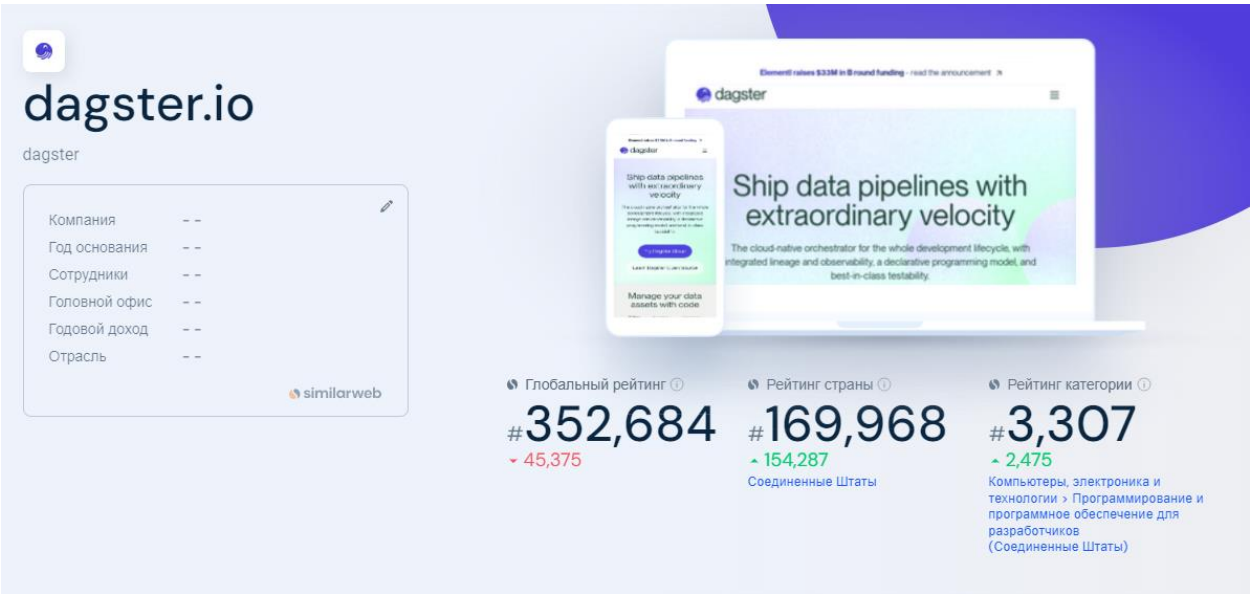

Рисунок 11 – Количество посещений официального сайта dagster

Как видно из рисунка, сайт dagster за месяца посетило около 350 тысяч человек, а скачало значит столько же или меньше. Что меньше, по сравнению с airflow в более, чем 22 раза, что означает, что airflow более популярен и искать решение проблем с ним будет проще.

Для дополнительной оценки проанализируем количество упоминаний airflow и dagster на github, которые показаны на рисунках 12 и 13 соответственно [27].

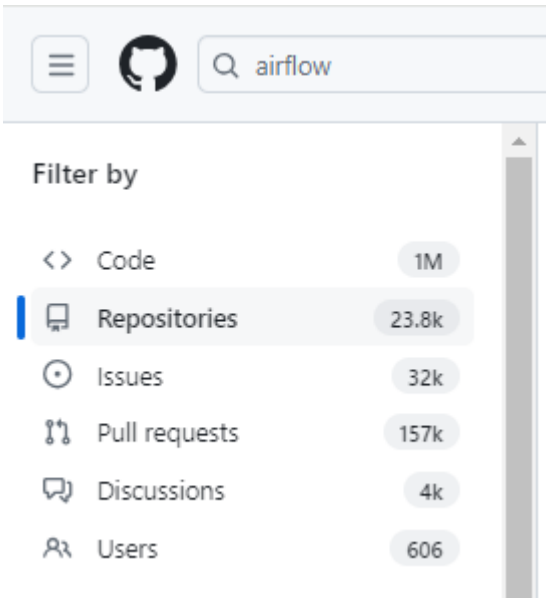

Рисунок 12 – Количество упоминаний в коде airflow

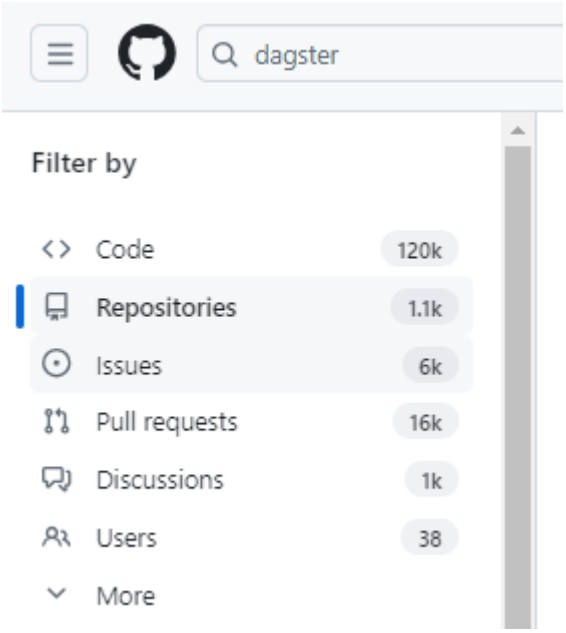

Рисунок 13 – Количество упоминаний в коде dagster

Как видно, airflow в коде упоминается около миллиона раз, а dagster около 120 тысяч. Данный анализ подтверждает, что airflow намного популярнее dagster.

Перед тем, как пустить код в эксплуатацию, его обязательно нужно протестировать, рассмотрим, какие средства тестирования предлагают оба инструмента.

В airflow можно проверить DAG на синтаксис до выгрузки его на сервер. Это можно сделать с помощью python <Путь к DAG`у>. Таким образом, будет выведена информация об синтаксических ошибках и предупреждения о изменении функционала в будущих версиях. Пример этого способа показан на рисунке 14.

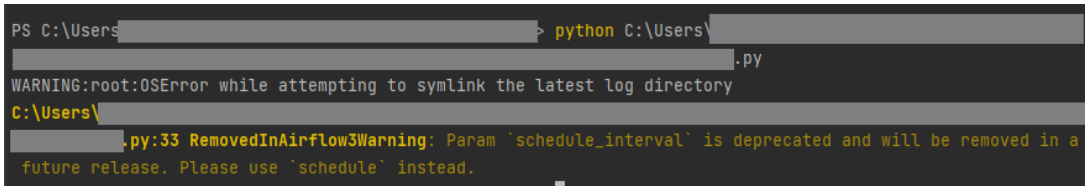

Рисунок 14 – Пример тестирования в airflow

Dagster предоставляет намного больший функционал для тестирования, чем airflow. Dagster позволяет создавать тестируемые и обслуживаемые приложения для работы с данными. Он предоставляет способы, позволяющие проводить модульное тестирование ваших приложений с данными [28]. Основной функцией модульного тестирования задач является JobDefinition.execute_in_process. Используя эту функцию, можно выполнить задание в процессе, а затем протестировать свойства выполнения, используя возвращаемый объект ExecuteInProcessResult. Пример кода модульного тестирования в dagster приведен на рисунке 15.

```
def test_job():
    result = do_math_job.execute_in_process()

    # return type is ExecuteInProcessResult
    assert isinstance(result, ExecuteInProcessResult)
    assert result.success
    # inspect individual op result
    assert result.output_for_node("add_one") == 2
    assert result.output_for_node("add_two") == 3
    assert result.output_for_node("subtract") == -1
```

Рисунок 15 – Пример кода тестирования в dagster

Таким образом, возможность тестирования предоставляют оба инструмента, но тестирование является разным. В airflow происходит тестирование синтаксиса, но, с другой стороны, dagster не нужно тестирование синтаксиса, так как он имеет идентичный синтаксис python, а airflow хоть и использует язык программирования Python, но некоторые функции языка переопределены и могут не всегда подсвечиваться средой программирования. Dagster предоставляет возможность модульного тестирования, что дает способ проверять данные и работу функций.

Создадим несколько DAG`ов и рассмотрим, как они работают в разных инструментах.

В airflow в одном файле создается один DAG. В dagster же части одного DAG могут лежать в разных файлах. Также в dagster есть сложности с

созданием большой группы DAG`ов, так как все они создаются в одном проекте и отображаются на одной доске, как показано на рисунке 16.

Рисунок 16 – Пример нескольких asset`ов в dagster

В airflow такой проблемы нет и можно добавлять DAG`ов столько, сколько нужно. Все они будут иметь отдельное отображение графа.

После дополнительного анализа двух инструментов, выбран Apache Airflow, так как он имеет большую поддержку сообщества и не имеет проблем с работой нескольких DAG`ов.

В данной главе был приведен обзор литературы, были найдены источники для более глубого понимания архитектуры хранения данных и также инструментов построения конвейеров данных. Был также проведен анализ разных инструментов оркестрации, их особенности, плюсы и минусы.

После изучения инструментов оркестрации данных был выбран оркестратор Airflow, как более подходящий под задачи расчета аналитических витрин.

Важным преимуществом Apache Airflow является активное сообщество разработчиков, поддерживающих проект, что обеспечивает стабильность и постоянное обновление инструмента. Кроме того, Airflow предоставляет широкий набор интеграций с различными источниками данных и инструментами обработки данных, что делает его универсальным инструментом для оркестрации сложных рабочих процессов.

2. Проектирование и разработка

В данной главе представлены требования к разрабатываемому программному обеспечению, выбор подходов и архитектуры, а также реализация самого программного обеспечения с помощью выбранного механизма оркестрации.

2.1. Проектирование

2.1.1. Функциональные и нефункциональные требования

Функциональные требования описывают поведение системы, т.е. ее действия (вычисления, преобразования, проверки, обработку и т.д.) [29 – 31].

Можно определить следующий набор функциональных требований к системе, представленный ниже.

1. Выполнение новой системы должно приводить к такому же результату, как и старая система;
2. DAG не должен содержать триггеров.
3. Генератор кода должен создавать все DAG файлы, которые имеют конфигурацию в базе данных.

Нефункциональные требования описывают свойства системы (удобство использования, безопасность, надежность, расширяемость и т.д.), которыми она должна обладать при реализации своего поведения [30 – 32].

Разрабатываемая система должна удовлетворять следующим нефункциональным требованиям:

1. Система должна быть реализована с помощью языка программирования Python;
2. DAG должен создаваться с помощью кодогенератора;
3. Кодогенератор должен быть реализован с помощью библиотеки шаблонизатора jinja2 [33];

4. DAG должен поддерживать версию Apache Airflow 2.6 и ниже.

2.1.2. Выбор подхода и архитектуры к разрабатываемой системе

В старой версии системы используются Triggers и Sensors.

Датчики или сенсоры – это специальный вид операторов, которые просто ждут, пока произойдет определенное действие, например копирование файла или завершение рабочей нагрузки в Kubernetes [34].

Триггеры – это условия или события, которые определяют, когда задача должна быть выполнена в направленном ациклическом графе (DAG).

В версии 2.4 появились наборы данных(Dataset). Они являются заменой логической группировки данных. Наборы данных могут обновляться вышестоящими задачами, а обновления наборов данных способствуют планированию последующих групп DAG.

Набор данных определяется унифицированным идентификатором ресурса (URI), пример кода определения экземпляра класса Dataset представлен ниже.

```
example_dataset = Dataset("s3://dataset-bucket/example.csv")
```

Airflow не делает никаких предположений о содержании или расположении данных, представленных URI.

Наборы данных позволяют определять явные зависимости между DAG и обновлениями данных. Это поможет:

1) Стандартизировать общение между командами. Наборы данных могут функционировать как API для связи, когда данные в определенном месте обновлены и готовы к использованию.

2) Сократить объем кода, необходимого для реализации межгрупповых зависимостей DAG. Даже если группы DAG не зависят от обновлений данных, можно создать зависимость, которая активирует группу обеспечения

доступности баз данных после того, как задача в другой DAG обновит набор данных.

3) Получить лучшее представление о том, как подключены DAG и как они зависят от данных. На вкладке «Наборы данных» в пользовательском интерфейсе Airflow показан график всех зависимостей между DAG и наборами данных в вашей среде Airflow.

4) Сократить затраты, поскольку наборы данных не используют рабочий слот в отличие от датчиков или других реализаций зависимостей между группами DAG.

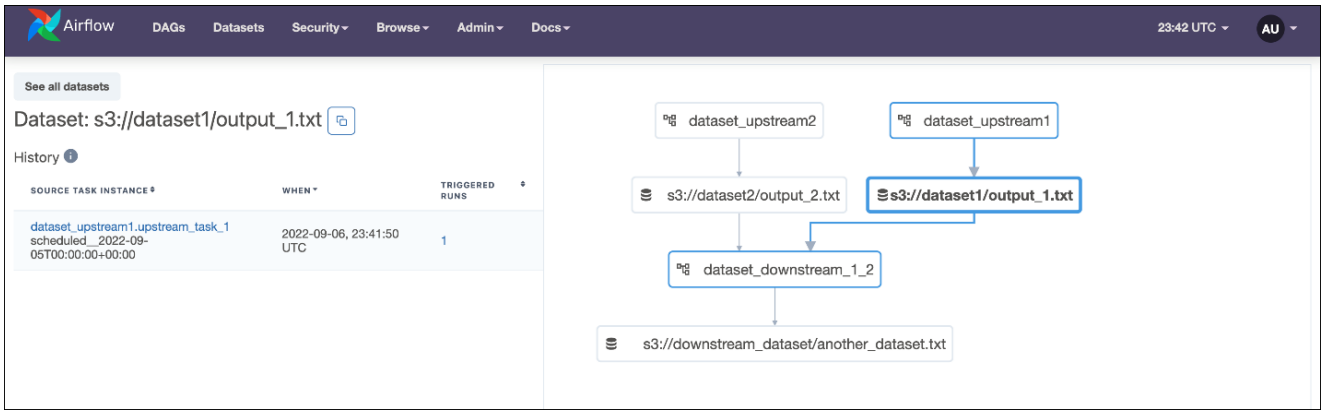

Рисунок 17 – Зависимость DAG и наборов данных в интерфейсе. Рисунок взят из источника [20]

Для того, чтобы связать DAG, необходимо сылаться на набор данных в задаче, передав его в параметр задачи outlets. Outlets является частью BaseOperator, поэтому доступен каждому оператору Airflow. Для запуска DAG после выполнения задачи, необходимо в schedule передать URI, такое же, как в outlets [35].

После выполнения задач write_instructions_to_file и write_info_to_file, код которых представлен в приложении А будут запущены дочерние DAG`и, которые ожидают эти наборы данных. Код дочерних DAG`ов представлен в приложении Б.

Как видно из кода, schedule ожидает выполнения 2 наборов данных. На рисунке 18 показано, как выглядит при этом интерфейс дочернего DAG.

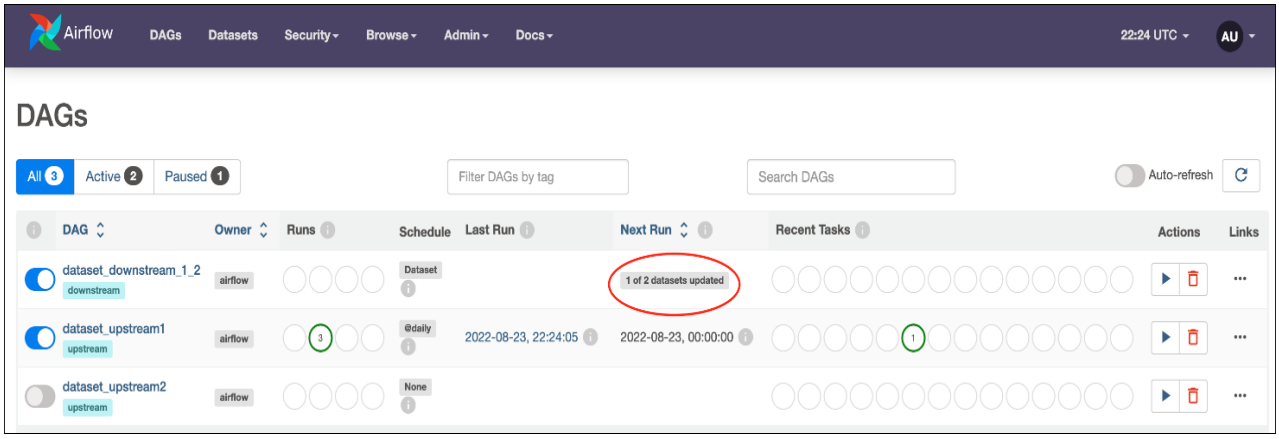

Рисунок 18 – Ожидание выполнения наборов данных в интерфейсе. Рисунок взят из источника [20]

Реализация нового программного обеспечения для оркестрации будет разработано через наборы данных, это должно позволить сократить код и убрать из кода триггеры, тем самым освободить рабочий слот worker. Новая система в такой реализации должна работать быстрее старой.

2.1.3. Проектирование программного обеспечения для оркестрации

Программное обеспечение для оркестрации будет состоять из нескольких DAG файлов и предполагать, что DAG одного слоя будет запускаться после выполнения необходимых этапов на другом. На рисунке 19 изображена диаграмма компонентов системы оркестрации расчетов аналитических витрин данных.

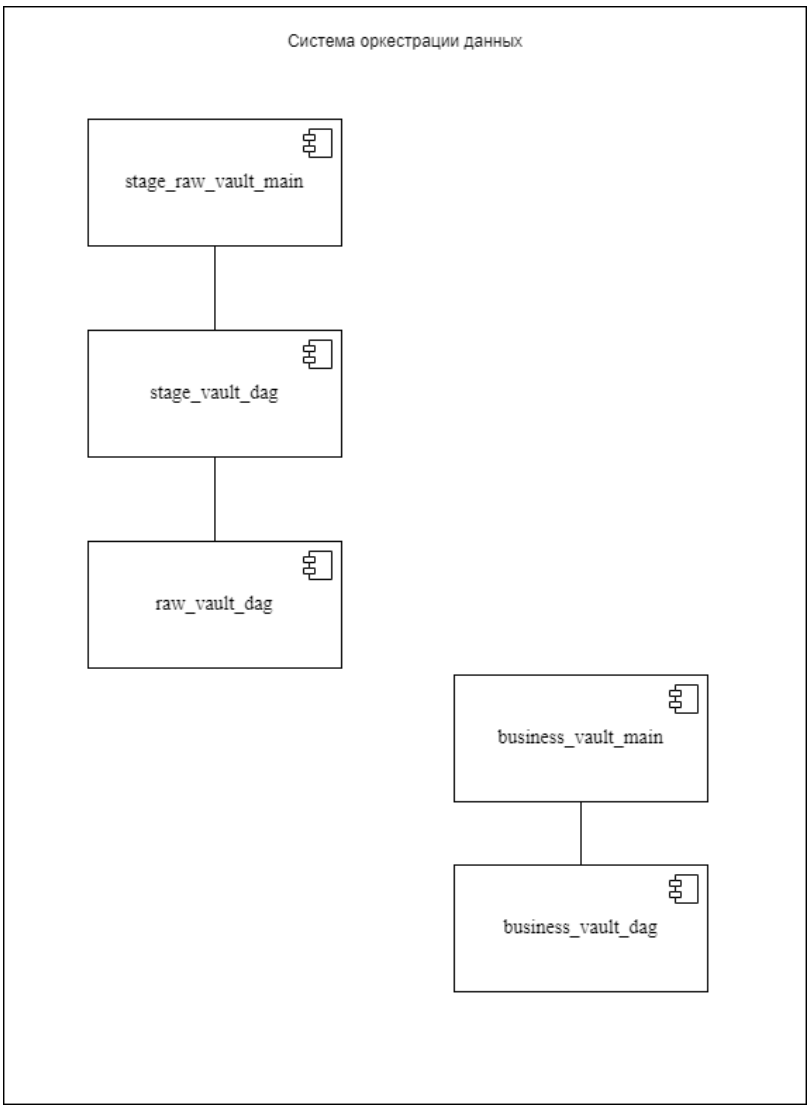

Рисунок 19 – Диаграмма компонентов системы оркестрации данных

На диаграмме каждый компонент – это отдельный DAG. Как видно из диаграммы, stage_raw_vault_main, который запускается по расписанию и пишет данных в лог о запуске и завершении работы, вызывает stage_vault_dag, который в свою очередь запускает следующий raw_vault_dag. Конвейер stage_vault_dag, выполняет задачу загрузки данных в stage слой, а raw_vault_dag, выполняет загрузку в слой raw. Последний конвейер больше ничего не запускает, но это не значит, что система закончила работу, по расписанию запускается business_vault_main, который использует сенсор и ожидает выполнения всех raw_vault_dag, после чего разрешает запуск business_vault_dag. Такая логика нужна из-за того, что business_vault_dag

может ожидать выполнения нескольких raw_vault_dag. Далее рассмотрим подробнее каждый конвейер и какие задачи он выполняет.

На рисунке 20 изображен конвейер stage_raw_vault_main, который сначала записывает данные о начале работы в лог, далее идет ожидание готовности данных в источнике, работа заканчивается записью в лог об окончании работы, после выполнения последней задачи создается набор данных, после чего запускаются все конвейеры stage_vault_dag. Новая версия имеет всего 3 шага, в то время, как старая 8, если считать все триггеры за один шаг.

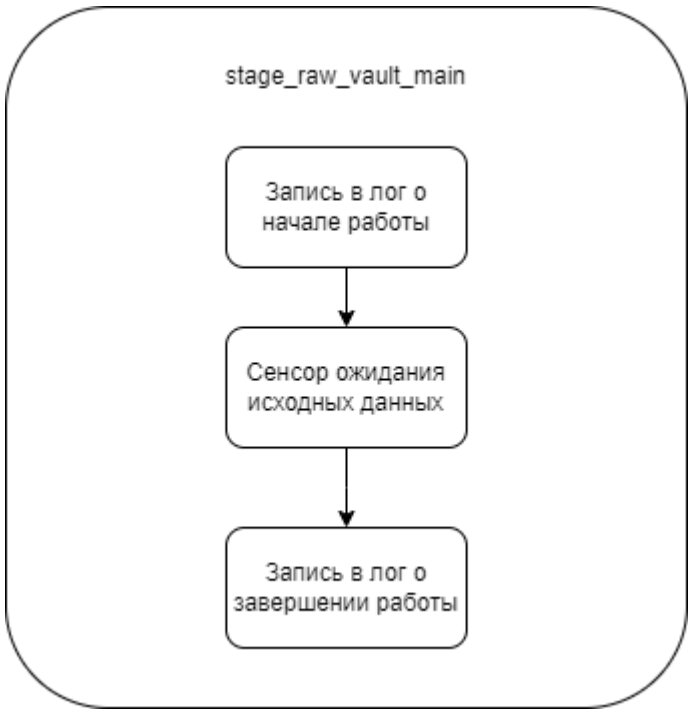

Рисунок 20 – Граф конвейера stage_raw_vault_main

На рисунке 21 изображен конвейер stage_vault_dag, который начинает работу с записи в лог данных о старте конвейера, далее проверяет есть ли для источника специальный номер данных, если нет, то создает. После создается внешняя таблица из который происходит загрузка сначала в промежуточную таблицу, где хранятся данные только с последней загрузки, а потом происходит создание и переключение партиций в таблицу, где хранятся данные с нескольких загрузок таблицы. Потом создаются суррогатные ключи и проверка

на их дублирование, если такие есть, то они удаляются, последним шагом происходит запись в лог об окончании работы конвейера, который также сообщает о готовности данных и запускает raw_vault_dag.

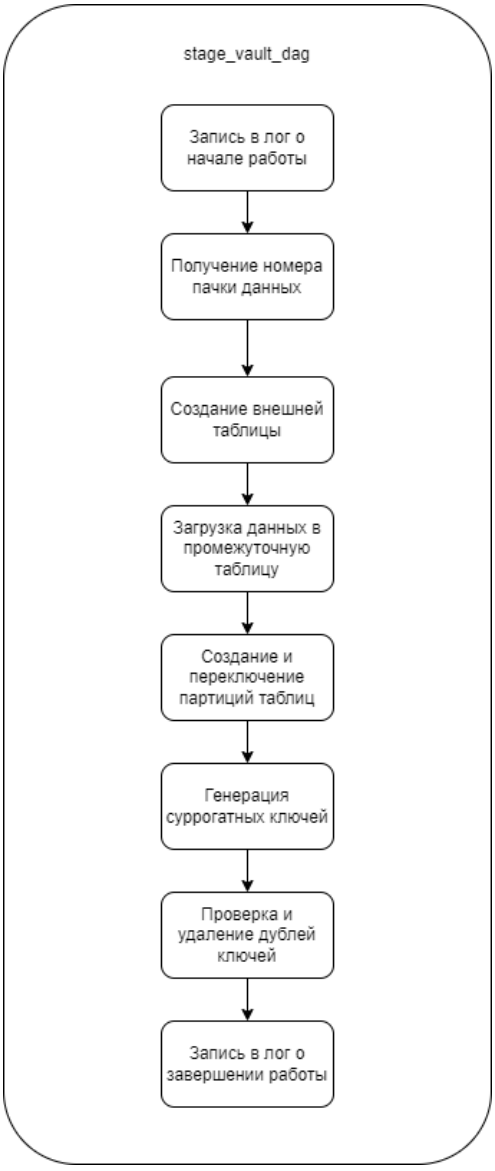

Рисунок 21 – Граф конвейера stage_vault_dag

На рисунке 22 изображен конвейер raw_vault_dag, который начинает работу с записи в лог данных о старте конвейера, далее собирает режим загрузки, например, полная загрузка данных или частичная. После проверяется есть ли полностью дублирующие данные в таблице, из который происходит загрузка, если есть, то происходит их удаление. Далее параллельно заполняются таблицы-хабы и таблицы-саттелиты. После их заполнения

происходит удаление устаревших данных и обновление режима загрузки, последним шагом происходит запись в лог об окончании работы конвейера.

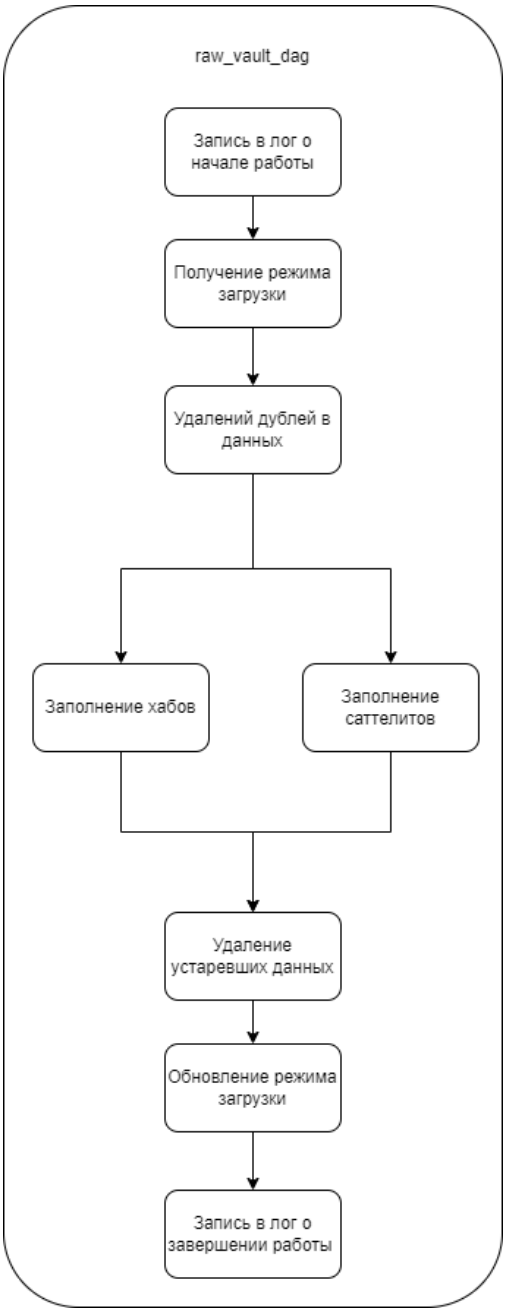

Рисунок 22 – Граф конвейера raw_vault_dag

Конвейер business_vault_main запускается по расписанию, его задачи такие же, как и у stage_raw_vault_main, только он запускает слой business и ожидает готовности данных от raw_vault_dag. В старой версии было 4 шага, если считать триггеры конвейеров business_vault_dag за 1 один шаг, в новой

версии этот шаг уйдет и задач будет только 3. Последняя задача будет сообщать о готовности данных и запускать конвейеры business_vault_dag.

Business_vault_dag необходим для трансформации данных в отчеты, конвейеры на данном слое имеют разные задачи по функциям и количеству, информация о проводимых трансформациях для отчетов описана в специальной таблице на базе данных. В новой версии все зависимые конвейеры будут запускать друг друга и ожидать выполнения завершения других зависимых DAG`ов. Для этого необходимо разработать генератор кода, который будет находить зависимости между DAG`ами.

2.1.4. Проектирование кодогенератора

Создавать вручную DAG файлы – это трудоемкая, долгая и рутинная работа, которую можно автоматизировать. Чтобы облегчить этот процесс предлагается создать кодогенератор, который будет создавать DAG файлы. На рисунке 23 можно увидеть диаграмму компонентов кодогенератора, который будет состоять из модуля сбора информации, базы данных, откуда будут собирать информацию и модуля генерации.

Модуль сбора информации осуществляет сбор данных с базы данных для конкретного DAG файла. База данных содержит в себе всю освновную информацию для каждого DAG файла, который необходимо создать. Модуль генерации принимает данные от модуля сбора данных и начинает генерацию DAG файлов.

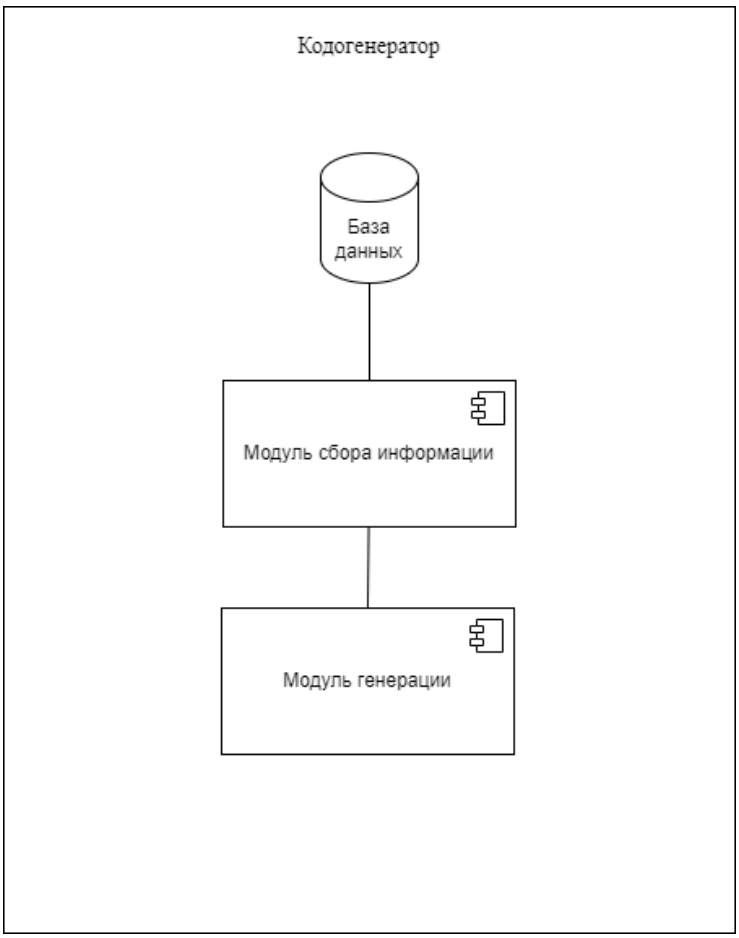

Рисунок 23 – Диаграмма компонентов кодогенератора

Чтобы было удобно работать и читать данные с базы необходимо определиться с инструментом, который будет это делать.

2.1.5. Выбор инструмента для работы с данными

Для удобной работы с данными необходимо выбрать библиотеку, которая работает с DataFrame.

1. Pandas – это быстрый, мощный, гибкий и простой в использовании инструмент для анализа и обработки данных с открытым исходным кодом, построенный на основе языка программирования Python [36]. Он построен на основе библиотеки NumPy и предоставляет несколько удобных функций для работы с табличными данными.

Pandas DataFrames можно создавать из различных источников данных, включая файлы CSV, электронные таблицы Excel, базы данных SQL и объекты JSON. После создания DataFrame им можно манипулировать с помощью различных функций, таких как фильтрация, группировка, объединение и поворот. Pandas также предоставляет мощные инструменты для визуализации данных и анализа временных рядов.

2. Polars – это библиотека DataFrame, предназначенная для обработки данных с быстрым временем освещения за счет реализации языка программирования Rust и использования Arrow в качестве основы. Предпосылка Polars – дать пользователям более быстрый опыт по сравнению с пакетом Pandas. Идеальная ситуация для использования пакета Polars – это когда у вас есть данные, которые слишком велики для Pandas, но слишком малы для использования Spark [37].

В Polars есть много доступных функций, которые выполняют ту же функцию, что и функции Pandas, и есть возможность условного выбора.

Основным аргументом в пользу использования Polars по-прежнему является более быстрое время выполнения.

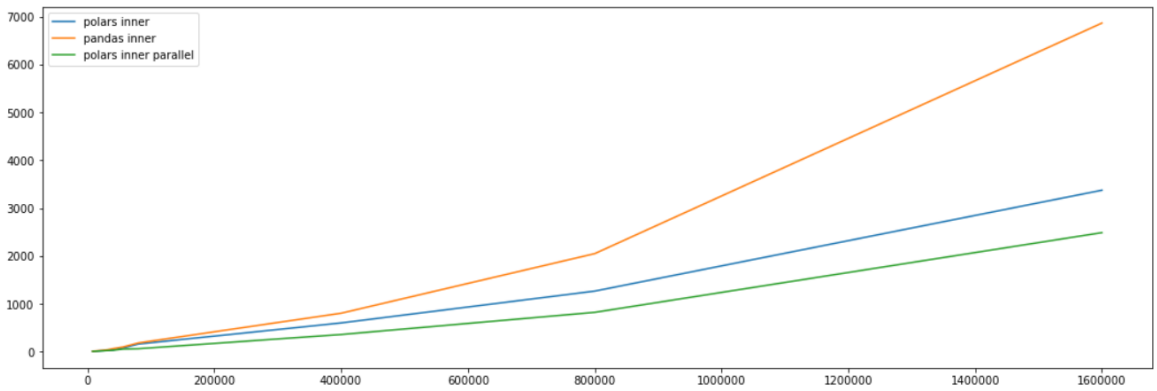

Рисунок 24 – Сравнение Polars и Pandas. Рисунок взят из источника [22]

На изображении выше показано сравнение времени трех разных методов: Polars Inner Join, Pandas Inner Join и Polars Inner Join Parallel. Как видно, время выполнения Pandas медленнее, когда размер данных становится больше по сравнению с Polars. Может не быть такой большой разницы во

времени выполнения при меньшем размере данных, но это становится более ясным, когда размер данных больше [38].

3. Vaex – это более новый API DataFrame на Python, предназначенный для обработки больших наборов данных, которые не помещаются в памяти. Он предоставляет высокоуровневый интерфейс для работы с данными с синтаксисом, аналогичным Pandas, но с несколькими оптимизациями для производительности и использования памяти [39].

Одной из ключевых особенностей Vaex является использование отложенных вычислений. Это означает, что Vaex не вычисляет результаты, пока они не будут явно запрошены, что позволяет оптимизировать использование памяти и скорость обработки. Vaex также поддерживает распределённые вычисления и ускорение графического процессора, что позволяет масштабировать его до ещё больших наборов данных [40]. Также он имеет набор удобных средств визуализации данных, упрощающих изучение набора данных.

Так как данных, с которыми необходимо работать, не много и необходим популярный инструмент, чтобы генератор кода можно было легко поддерживать, то выбор пал на Pandas, анализ показал, что хоть он и проигрывает по скорости на больших данных, но он достаточно известный, чтобы можно было легко поддерживать код и на малом количестве данных его скорости вполне хватает для его использования.

2.2. Программная реализация

2.2.1. Реализация программного обеспечения для оркестрации

Для реализации программного обеспечения для оркестрации данных необходимо создавать DAG файлы и так как было решено их генерировать, то был создан jinja2 шаблон DAG файла, по которому будут формироваться все DAG файлы для каждого из этапов программы для оркестрации. Часть кода шаблона представлена в листинге ниже.

```
@task(pool='default_pool', queue='default',
outlets=[DATASET_START])
def dag_func(log, dags):
     {%- if stage_vault_dags|length > 0 %}
     stage_vault_dags_list = ({%- for dag in stage_vault_dags
%}'{{dag.stage_vault_dags_name }}',{%- endfor %})
     reset_stage_vault_dags(stage_vault_dags_list ,'{{
dag_table_name }}')
     {%- endif %}
     write_log(log, dags)
     make_report('{{ src_name }}', 'start_stage_vault_dags')
```

В коде представлена функция одного из шаблонов DAG файла. После выполнения этой функции будет запущен DAG, который ожидает создания набора данных DATASET_START. URI dataset`a указан в параметре декоратора task, который называется outlets. Ниже приведена часть кода, где показано, как собирается URI набора данных и как DAG ожидает выполнение задачи из другого DAG.

```
DATASET_START = Dataset("run_stage_dags_dataset_{{ database }}")
STAGE_VAULT_DAG_DATASET = Dataset("table://{{ database }}/{{
source_stage_schema }}.{{ table_name_stage_vault }}")
with DAG("{{ dag_id }}",
          default_args={ 'depends_on_past': {{ depends_on_past
}},
          'email': {{ email }},
},
        start_date=datetime(2024, 1, 1),
        schedule=[DATASET_START],
        …,
        tags={{ tags }}) as dag:
```

Как видно из кода в листинге выше, набор данных DATASET_START содержит постоянное название, изменяется только название базы. Этот набор запускает все DAG`и, которые в параметре schedule имеют название этого набора данных. Набор данных STAGE_VAULT_DAG_DATASET уже изменяется в зависимости от базы, схемы и имени таблицы. Помимо наборов данных в шаблон подставляются также и другие данные. Все, что подставлять в шаблон указано в двойных фигурных скобках.

2.2.2. Реализация кодогенератора

Для реализации генератора кода были использован язык программирования Python и его библиотеки: шаблонизатор jinja2, os , который необходим для создания папок и файлов, sqlalchemy, для подключения к базе данных и pandas, который нужен для удобной работы с данными и их сбора с базы данных [41 – 42].

Для генерации файлов необходимо собрать из базы данных информацию о DAG, этим занимается модуль сбора информации, в листинге ниже представлена часть кода, где создается подключение к базе и чтение информации. На базе данных содержится вся информация, необходимая для создания DAG (имя дага, время запуска, тэги и др.).

```
engine = create_engine(connection_string)
df_dags = pd.read_sql_query(sql_query, con=engine, dtype=object)
```

После того, как информация собрана, запускается модуль генерации, который в цикле добавляет дополнительную информацию о даге, такую как имя базы, имя схемы и тэги. А потом создает DAG файл, подставляя данные в шаблон.

2.3. Тестирование системы

Тестирование приложения проводилось вручную, путем запуска генератора кода, сверкой файлов с конфигурацией на базе данных и запуском DAG`ов на сервере.

Для тестирования системы использовалось функциональное тестирование, в ходе которого проверяется работоспособность всех описанных ранее в функциональных требованиях задач.

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям [43].

В таблице 3 приведены функциональные тесты для разрабатываемого программного обеспечения.

Таблица 3 – Функциональные тесты

№	Название теста	Действия	Ожидаемый результат	Тест пройден?
1	Генератор кода создает столько DAG файлов, сколько конфигураций на базе данных	Сгенерировать файлы и проверить, соответствует ли их количество конфигурациям на базе данных	Количество файлов соответствует количество конфигураций	Да
2	Зависимые DAG`и вызываются без использования триггера	Запустить stage_raw_vault_main и посмотреть, что в коде отсутствуют триггеры и дочерние DAG`и запускаются и успешно отрабатывают	В коде отсутствуют триггеры и все DAG`и успешно отрабатывают	Да
3	Новая система приводит к такому же результаты, что и старая	Запустить stage_raw_vault_main и посмотреть, что в базу грузятся такие же данные, как и в старой версии	Данные идентичны старой версии	Да

Все тесты из таблицы были пройдены успешно.

В данной главе были сформулированы функциональные и нефункциональные требования к системе.

Была выбрана архитектура и подходы к DAG.

Были спроектированы программное обеспечение для оркестрации данных и генератор кода для DAG файлов, построены диаграммы компонентов, выбран инструмент для работы с данными.

Была описана реализация программы для оркестрации и кодогенератора, приведена часть кода.

Проведено тестирование программного обеспечения для оркестрации расчетов аналитических витрин данных и генератора кода к ней. Было проведено тестирование основных функций системы. В следствии чего можно сделать вывод, что результаты тестирования приложения соответствует выделенным ранее требованиям.

3. Анализ работы программы для оркестрации

Проведем анализ новой версии программы для оркестрации данных со старой и выясним, произошли ли улучшения и на сколько. Для этого запустим по 20 раз DAG`и старой и новой версии, которые запускают 332, 168 и 80 DAG`ов. У DAG`ов есть возможность отключить выполнение основного кода и в этих тестах будут писаться только данные в лог и начале и завершении и вызываться зависимые DAG`и, это значит, что анализ не будет зависеть от загружаемых данных. Для анализа построим гистограммы для каждого конвейера, который запускает другие. Для отображения гистограммы будет использована библиотека seaborn и matplotlib для отображения титула и наименований осей [44 – 45].

Сравним гистограммы на рисунках 25 и 26, которые показывают старую и новую версии для DAG`а, которые запускает 332 других DAG`а.

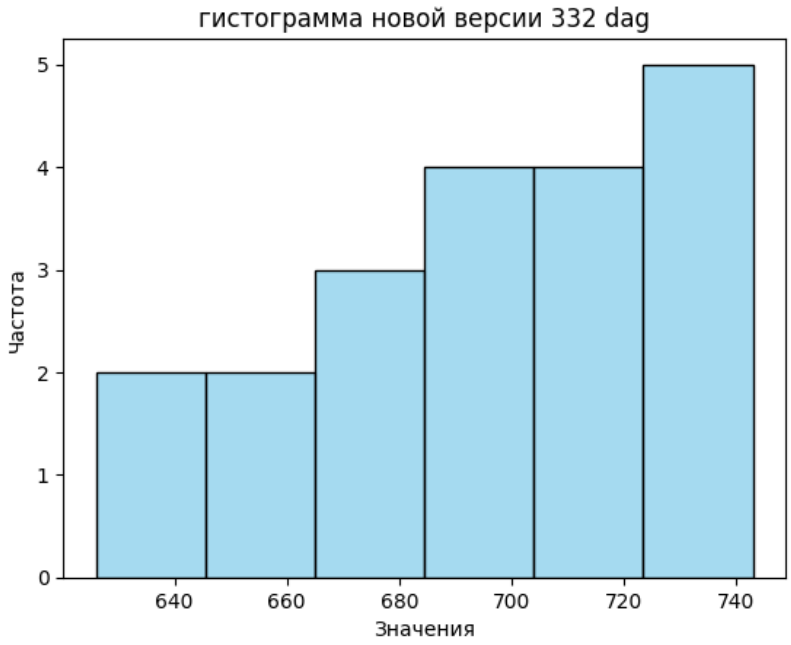

Рисунок 25 – Гистограмма для DAG`а новой версии, запускающего другие 332 DAG`а

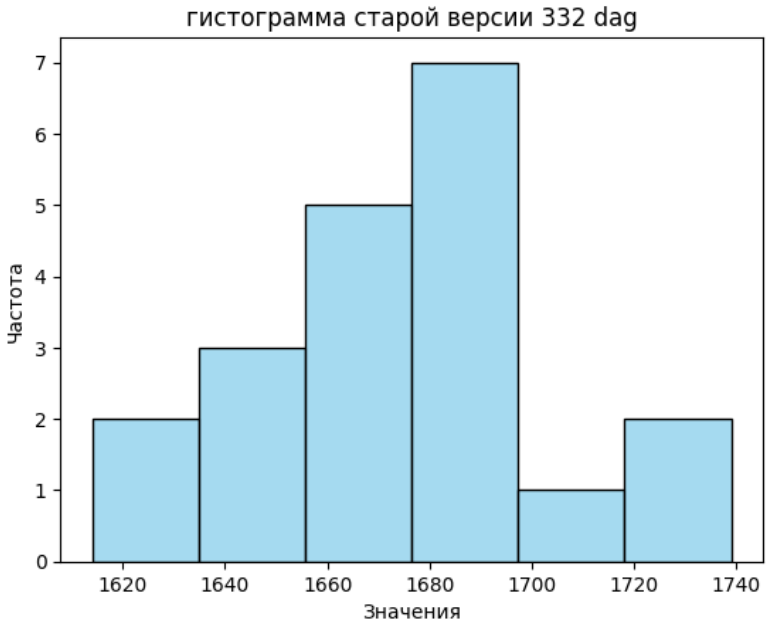

Рисунок 26 – Гистограмма для DAG`а старой версии, запускающего другие 332 DAG`а

На рисунке 25 видно, что гистограмма перекошена в сторону более больших значений, в то время как на рисунке 26 гистограмма близка к нормальному распределению. DAG на рисунках 25 и 26 имеет зависимые DAG`и, которые в ходе выполнения ждут завершения нескольких предыдущих DAG`ов. Возможно, на рисунке 25 выражена эта ситуация, планировщик запускал DAG`и не последовательно, а мог запустить все DAG`и одной ветки, а потом перейти к другим. В то время, как старая версия выполняла задачи более последовательно, поэтапно и не запускала DAG`и.

Рассмотрим гистограммы на рисунках 27 и 28. DAG на этих гистограммах запускает весь слой stage или raw в старой версии и в новой запускает загрузку слоев после отработки предыдущего DAG`а, эти DAG`и не имеют ожидания выполнения нескольких DAG`ов, как в рисунках 25 и 26. Обе гистограммы имеют нормальное распределение.

Рисунок 27 – Гистограмма для DAG`а старой версии, запускающего другие 168 DAG`а

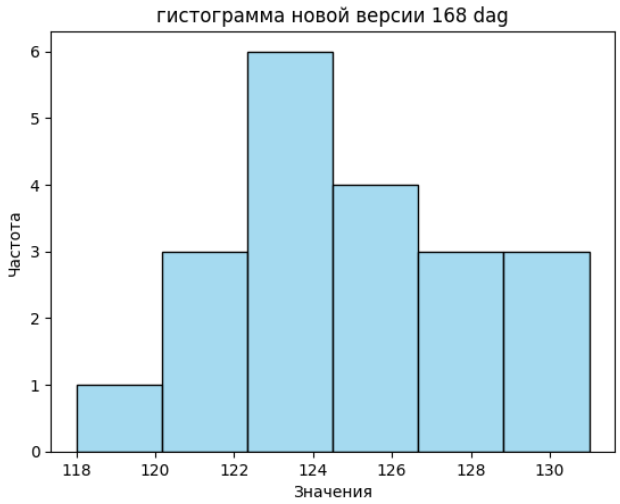

Рисунок 28 – Гистограмма для DAG`а новой версии, запускающего другие 168 DAG`а

Гистограммы на рисунках 29 и 30 близки к нормальному распределению. DAG`и на этих гистограммах работают также, как и на рисунках 27 и 28, но вызываемых DAG`ов здесь меньше.

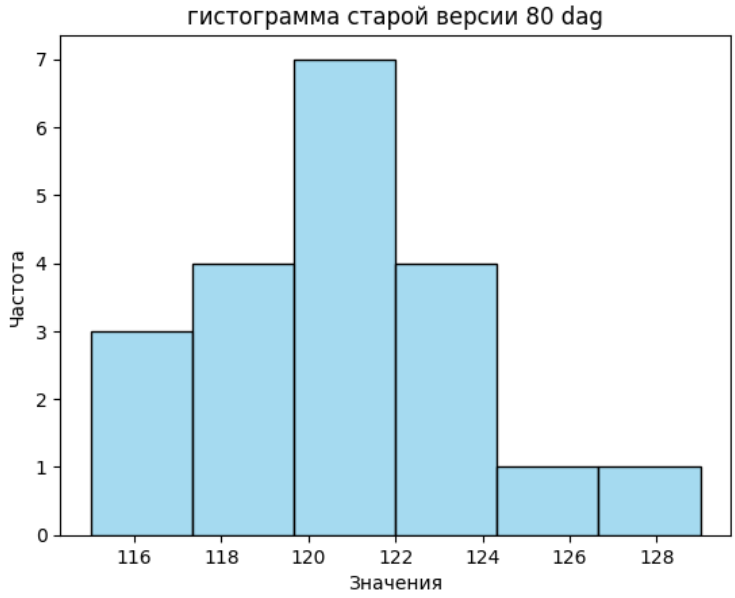

Рисунок 29 – Гистограмма для DAG`а старой версии, запускающего другие 80 DAG`ов

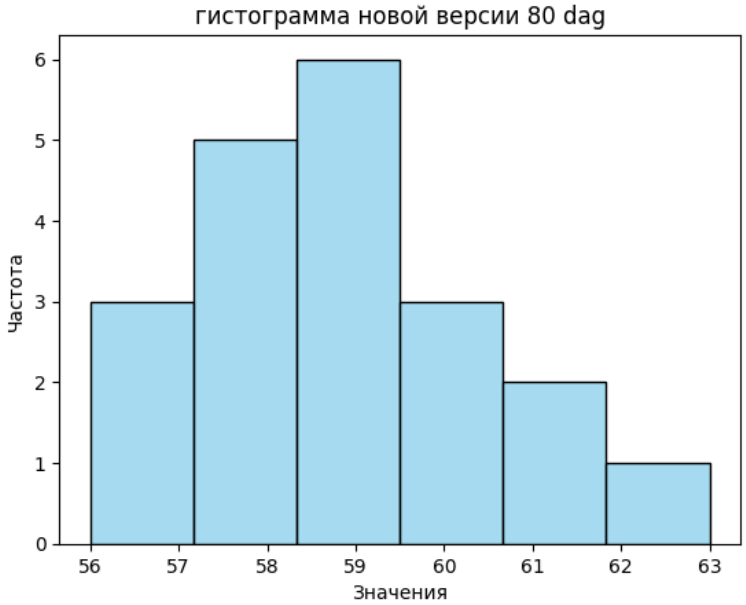

Рисунок 30 – Гистограмма для DAG`а новой версии, запускающего другие 80 DAG`ов

На всех рисунках значения выполнения DAG`а указаны в секундах. Код отображения одной из гистограмм представлен в листинге. В данном коде, для рисования гистограммы используется библиотека seaborn, а для написания титула и именования осей используется matplotlib.

```
sns.histplot(df['new_80'], bins=6, kde=False, color='skyblue')
plt.title('гистограмма новой версии 80 dag')
plt.xlabel('Значения')
plt.ylabel('Частота')
plt.show()
```

На рисунке 31 приведена таблица с описательной статистикой, проанализируем её.

	old_332	new_332	old_168	new_168	old_80	new_80
count	20.000000	20.000000	20.000000	20.000000	20.000000	20.00000
mean	1676.200000	696.600000	263.550000	124.450000	120.650000	58.90000
std	30.632972	34.705376	4.817457	3.300319	3.360373	1.68273
min	1614.000000	626.000000	252.000000	116.000000	115.000000	56.00000
25%	1659.500000	680.000000	262.000000	123.000000	118.000000	58.00000
50%	1673.500000	699.500000	265.000000	124.000000	121.000000	59.00000
75%	1691.000000	724.750000	266.000000	126.250000	122.250000	60.00000
max	1739.000000	743.000000	273.000000	132.000000	129.000000	63.00000

Рисунок 31 – Таблица с описательной статистикой данных

Как видно из таблицы, стандартное отклонение больше всего у DAG`ов, которые имеют несколько зависимостей на запуск из-за чего время завершения загрузки может иметь больший разброс, по сравнению, с DAG`ами, у которых нет ожидания загрузки нескольких таблиц.

Сравним время выполнения старого и нового DAG`ов на графике. На рисунке 32 показано время выполнения DAG`ов, которые запускают 332 зависимых DAG`а.

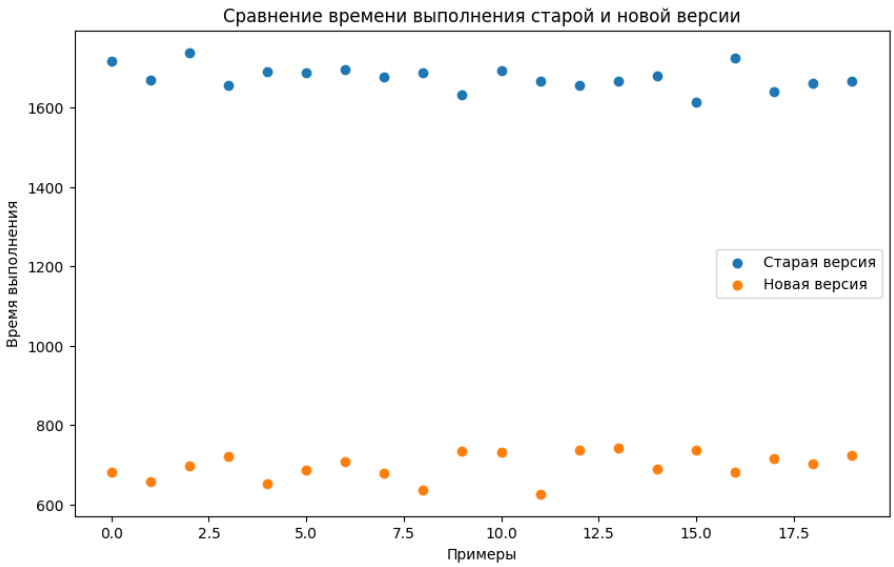

Рисунок 32 – График времени выполнения DAG`ов с 332 зависимыми DAG`ами

Из рисунка видно, что время выполнения разных версий не пересекается и новая версия работает быстрее, что означает, улучшение DAG`ов сработало. На рисунках 33 и 34 приведены графики для других конвейеров.

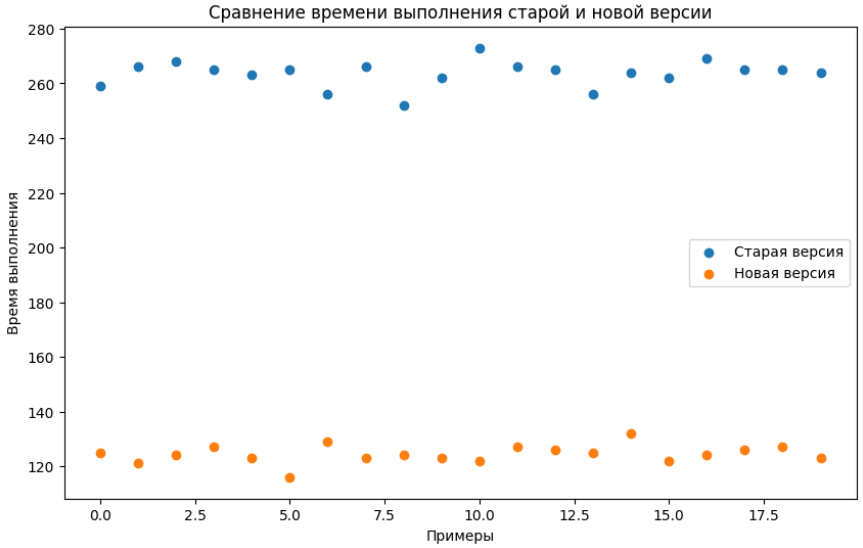

Рисунок 33 – График времени выполнения DAG`ов с 168 зависимыми DAG`ами

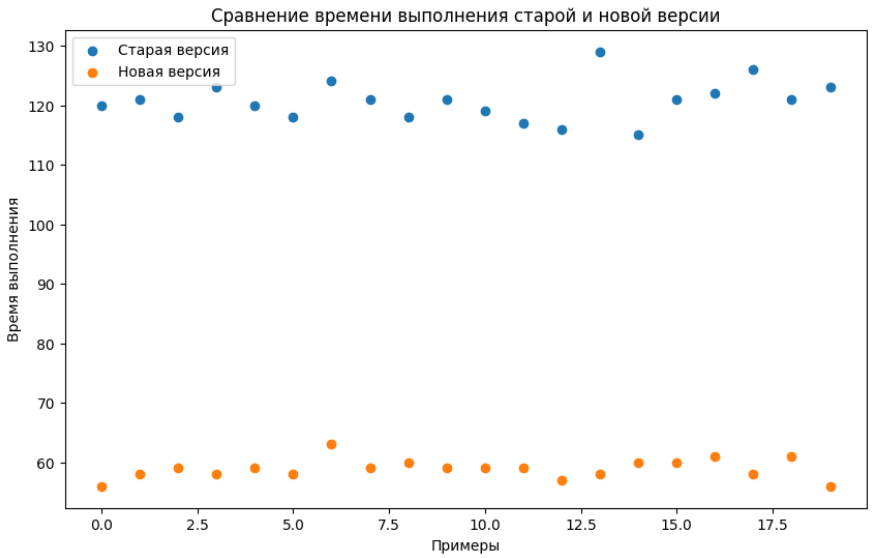

Рисунок 34 – График времени выполнения DAG`ов с 80 зависимыми DAG`ами

Исходя из гистограмм и графиков времени выполнения, можно сделать вывод, что улучшение системы прошло удачно и время выполнения уменьшилось, что помимо более ранних расчетов витрин дает возможность добавлять больше DAG`ов.

В данной главе было проведено сравнение старого и нового механизмов оркестрации. Были приведены гистограммы и графики времени выполнения DAG`ов. В ходе сравнения и анализа стало видно, что новый механизм работает лучше, а именно быстрее.

ЗАКЛЮЧЕНИЕ

Целью выпускной квалификационной работы являлось исследование различных механизмов оркестрации расчетов аналитических витрин данных и разработка программного обеспечения, с использованием выбранного механизма, для оркестрации расчетов аналитических витрин данных под нужды компании АО «Банк СИНАРА».

Поставленную цель, можно считать выполненной, так как продукт был разработан и внедрен в механизм оркестрации расчетов аналитических витрин данных компании АО «Банк СИНАРА».

В теоретической части работы была рассмотрена некоторая литература, связанная по тематике исследования, где рассматривались архитектуры хранения данных, механизм работы Apache Airflow, также было выполнено исследование различных механизмов оркестрации данных, таких как, Apache Airflow, Luigi, Apache NiFi, Dagster, Oozie. Были приведены их описание, особенности, плюсы и минусы. Был выбран Apache Airflow, как наиболее подходящий оркестратор. В практической части работы было спроектировано и реализовано программное обеспечение для оркестрации расчетов аналитических витрин данных. А также проведено тестирование, в ходе которого все тесты были успешно пройдены. Третий раздел был посвящен анализу работы разработанного программного обеспечения со старой версией. Были приведены гистограммы времени выполнения DAG`ов, всего было по 20 запусков для каждой версии. DAG`и не выполняли основной функционал, сравнение шло только на выполнении записи данных в лог на базу данных и запусках зависимых конвейеров. Были сделаны выводы, что в ходе данной работы было значительно уменьшено время выполнения оркестрации расчетов аналитических витрин данных, что может свидетельствовать об успешности разработанного программного обеспечения.

Разработанный продукт полностью удовлетворяет поставленным функциональным и нефункциональным требованиям.

Для достижения всех целей работы были решены следующие задачи:

1. произведен обзор некоторой литературы по тематике исследования и выполнено изучение механизмов оркестрации, по итогам которого был выбран наиболее подходящий инструмент оркестрации;
2. спроектирована архитектура DAG и кодогенератора. Приведены диаграммы компонентов, выбран инструмент для работы с данными;
3. разработано программное обеспечение для оркестрации расчетов аналитических витрин данных;
4. разработан кодогенератор для генерации DAG;
5. выполнено тестирование программного обеспечения, в ходе которого все тесты были успешно пройдены;
6. выполнено сравнение и анализ новой и старой версии DAG. Были приведены гистограммы и графики времени выполнения. Был сделан вывод, что новый механизм работает быстрее, чем старый.

Главными преимуществами новой версии продукта является его уменьшение времени выполнения и уменьшение количества кода за счет удаления триггеров.

В будущем планируется дальше развивать данный продукт. Планируется спроектировать и разработать DAG в Airflow, который будет иметь зависимости между Raw vault и business vault, что должно позволить уменьшить скорость работы всей системы, возможность запускать нужные конвейеры, в зависимости от того, какие данные пришли для рассчета аналитической витрины и добавить функцию, чтобы конвейер с набором данных не запускал зависимые, на случай, если нужно запустить только его.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое ETL (извлечение, преобразование, загрузка)? [Электронный ресурс] URL: https://aws.amazon.com/ru/what-is/etl/ (дата обращения: 18.03.2024 г.).

2. Что такое ETL? [Электронный ресурс] URL: https://www.oracle.com/cis/integration/what-is-etl/ (дата обращения: 18.03.2024 г.).

3. Что такое ETL и с какими задачами поможет [Электронный ресурс] URL: https://yandex.cloud/ru/blog/posts/2023/02/extract-transform-load?utm_referrer=https%3A%2F%2Fwww.google.com%2F (дата обращения: 18.03.2024 г.).

4. Что такое витрина данных? [Электронный ресурс] URL: https://www.oracle.com/cis/autonomous-database/what-is-data-mart/ (дата обращения: 18.03.2024 г.).

5. Простор для данных [Электронный ресурс] URL: https://habr.com/ru/articles/650237/ (дата обращения: 18.03.2024 г.).

6. Data Mart: основы и преимущества витрины данных [Электронный ресурс] URL: https://yandex.cloud/ru/docs/glossary/datamart (дата обращения: 18.03.2024 г.).

7. AirFlow: что это, как работает и при чем здесь облака. [Электронный ресурс] URL: https://cloud.vk.com/blog/airflow-what-it-is-how-it-works (дата обращения: 18.03.2024 г.).

8. Зачем дата-инженеру нужен оркестратор? [Электронный ресурс] URL: https://datafinder.ru/services/zachem-data-inzheneru-nuzhen-orkestrator (дата обращения: 18.03.2024 г.).

9. What Is Data Orchestration? [Электронный ресурс] URL: https://www.digitalroute.com/resources/glossary/data-orchestration/ (дата обращения: 18.03.2024 г.).

10. Harenslak B., de Ruiteri J. Data Pipelines with Apache Airflow. MANNING, 2021. – C. 2 – 27.

11. VINEȘ A., SAMOILĂ R. An Overview of Data Vault Methodology and Its Benefits. Informatica Economică, C. 15 – 20.

12. Введение в Data Vault [Электронный ресурс] URL: https://habr.com/ru/articles/348188/ (дата обращения: 20.04.2024 г.).

13. Petraitytė E. Exploring Efficient Workflow Frameworks for Data Management, 2024. – C. 30 – 31.

14. What is Airflow? [Электронный ресурс] URL: https://airflow.apache.org/docs/apache-airflow/stable/index.html (дата обращения: 18.03.2024 г.).

15. Как Apache Airflow помогает дирижировать данными компаний [Электронный ресурс] URL: https://practicum.yandex.ru/blog/apache-airflow/ (дата обращения: 18.03.2024 г.).

16. Welcome to Dagster! [Электронный ресурс] URL: https://docs.dagster.io/getting-started (дата обращения: 20.03.2024 г.).

17. Dagster | Туториал [Электронный ресурс] URL: https://habr.com/ru/articles/690342/ (дата обращения: 20.03.2024 г.).

18. What is Luigi? [Электронный ресурс] URL: https://censius.ai/mlops-tools/luigi (дата обращения: 20.03.2024 г.).

19. Обзор фреймворка Luigi для построения последовательностей выполнения задач [Электронный ресурс] URL: https://clck.ru/3AvAcY (дата обращения: 20.03.2024 г.).

20. Apache NiFi [Электронный ресурс] URL: https://nifi.apache.org/ (дата обращения: 20.03.2024 г.).

21. Что такое Apache NiFi и как он устроен [Электронный ресурс] URL: https://bigdataschool.ru/wiki/nifi (дата обращения: 20.03.2024 г.).

22. Apache NiFi: что это такое и краткий обзор возможностей [Электронный ресурс] URL:

https://habr.com/ru/companies/rostelecom/articles/432166/ (дата обращения: 20.03.2024 г.).

23. Data Orchestration Tools: Ultimate Review [Электронный ресурс] URL: https://lakefs.io/blog/data-orchestration-tools-2023/#h-7-apache-oozie (дата обращения: 20.03.2024 г.).

24. Apache Oozie Workflow Scheduler for Hadoop [Электронный ресурс] URL: https://oozie.apache.org/ (дата обращения: 20.03.2024 г.).

25. How We Track the Growth of Apache Airflow [Электронный ресурс] URL: https://www.astronomer.io/blog/how-we-track-the-growth-of-apache-airflow/ (дата обращения: 11.05.2024 г.).

26. Аналитика количества посещений сайта dagster [Электронный ресурс] URL: https://www.similarweb.com/ru/website/dagster.io/#overview/ (дата обращения: 11.05.2024 г.).

27. Официальный сайт github [Электронный ресурс] URL: https://github.com/ (дата обращения: 11.05.2024 г.).

28. Документация dagster [Электронный ресурс] URL: https://docs.dagster.io/concepts/testing (дата обращения: 11.05.2024 г.).

29. Как писать функциональные требования к ПО [Электронный ресурс] URL: https://clck.ru/3AvAdg (дата обращения: 11.05.2024 г.).

30. Куликов С.С. Тестирование программного обеспечения. Базовый курс. 2 издание. – Издательство Четыре Четверти, 2017. – 312 с.

31. Функциональные и нефункциональные требования [Электронный ресурс] URL: https://visuresolutions.com/ru/requirements-management-traceability-guide/functional-vs-non-functional-requirements/ (дата обращения: 04.05.2024 г.).

32. Как писать НЕфункциональные требования к ПО [Электронный ресурс] URL: https://vc.ru/u/1397572-rabotayu-v-it/629888-kak-pisat-nefunkcionalnye-trebovaniya-k-po (дата обращения: 04.05.2024 г.).

33. Официальная документация jinja [Электронный ресурс] URL: https://jinja.palletsprojects.com/en/3.1.x/ (дата обращения: 04.05.2024 г.).

34. Краткий ликбез по программным компонентам Apache AirFlow [Электронный ресурс] URL: https://bigdataschool.ru/blog/airflow-key-concepts-from-dag-to-hook.html (дата обращения: 04.05.2024 г.).

35. Datasets and data-aware scheduling in Airflow [Электронный ресурс] URL: https://docs.astronomer.io/learn/airflow-datasets (дата обращения: 04.05.2024 г.).

36. Официальная документация pandas [Электронный ресурс] URL: https://pandas.pydata.org/ (дата обращения: 07.05.2024 г.).

37. Polars — DataFrames for the new era [Электронный ресурс] URL: https://pola.rs/ (дата обращения: 07.05.2024 г.).

38. Топ-3 альтернативных пакета Python для Pandas [Электронный ресурс] URL: https://dev-gang.ru/article/top-alternativnyh-paketa-python-dlja-pandas-ir5j0ru3e3/ (дата обращения: 07.05.2024 г.).

39. Python Pandas против Vaex Dataframes: сравнительный анализ [Электронный ресурс] URL: https://clck.ru/3AvAd7 (дата обращения: 07.05.2024 г.).

40. Vaex.io: An ML Ready Fast DataFrame for Python [Электронный ресурс] URL: https://vaex.io/ (дата обращения: 07.05.2024 г.).

41. Документрация модуля os [Электронный ресурс] URL: https://docs.python.org/3/library/os.html (дата обращения: 07.05.2024 г.).

42. The Python SQL Toolkit and Object Relational Mapper [Электронный ресурс] URL: https://www.sqlalchemy.org/ (дата обращения: 07.05.2024 г.).

43. Лайза Криспин, Джанет Грегори. Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд. – М.: Издательский дом «Вильямс», 2016. – 466 с.

44. seaborn: statistical data visualization [Электронный ресурс] URL: https://seaborn.pydata.org/ (дата обращения: 07.05.2024 г.).

45. Matplotlib: Visualization with Python [Электронный ресурс] URL: https://matplotlib.org/ (дата обращения: 07.05.2024 г.).

ПРИЛОЖЕНИЕ А

```
from pendulum import datetime
from airflow.datasets import Dataset
from airflow.decorators import dag, task

API = "https://www.thecocktaildb.com/api/json/v1/1/random.php"
INSTRUCTIONS=Dataset("file://localhost/airflow/include/cocktail_
instructions.txt")
INFO =
Dataset("file://localhost/airflow/include/cocktail_info.txt")

@dag(
    start_date=datetime(2022, 10, 1),
    schedule=None,
    catchup=False,
)
def datasets_producer_dag():
    @task
    def get_cocktail(api):
        import requests
        r = requests.get(api)
        return r.json()

    @task(outlets=[INSTRUCTIONS])
    def write_instructions_to_file(response):
        cocktail_name = response["drinks"][0]["strDrink"]
        cocktail_instructions =
response["drinks"][0]["strInstructions"]
        msg = f"See how to prepare {cocktail_name}:
{cocktail_instructions}"
        f = open("include/cocktail_instructions.txt", "a")
        f.write(msg)
        f.close()
```

Продолжение ПРИЛОЖЕНИЯ А

```
    @task(outlets=[INFO])
    def write_info_to_file(response):
        import time
        time.sleep(30)
        cocktail_name = response["drinks"][0]["strDrink"]
        cocktail_category = response["drinks"][0]["strCategory"]
        alcohol = response["drinks"][0]["strAlcoholic"]
        msg = f"{cocktail_name} is a(n) {alcohol} cocktail from
category {cocktail_category}."
        f = open("include/cocktail_info.txt", "a")
        f.write(msg)
        f.close()

    cocktail = get_cocktail(api=API)

    write_instructions_to_file(cocktail)
    write_info_to_file(cocktail)

datasets_producer_dag()
```

ПРИЛОЖЕНИЕ Б

```
from pendulum import datetime
from airflow.datasets import Dataset
from airflow.decorators import dag, task

INSTRUCTIONS =
Dataset("file://localhost/airflow/include/cocktail_instructions.txt")
INFO = Dataset("file://localhost/airflow/include/cocktail_info.txt")

@dag(
    dag_id="datasets_consumer_dag",
    start_date=datetime(2022, 10, 1),
    schedule=[INSTRUCTIONS, INFO],  # ожидает оба DAG
    catchup=False,
)
def datasets_consumer_dag():
    @task
    def read_about_cocktail():
        cocktail = []
        for filename in ("info", "instructions"):
            with open(f"include/cocktail_{filename}.txt", "r") as f:
                contents = f.readlines()
                cocktail.append(contents)

        return [item for sublist in cocktail for item in sublist]

    read_about_cocktail()

datasets_consumer_dag()
```