**АРХИТЕКТУРА РЕШЕНИЯ.**

Исполнитель: Асет Иманкулов

Должность: разработчик

Дата: 31.07.2023

Версия документа: 3.0

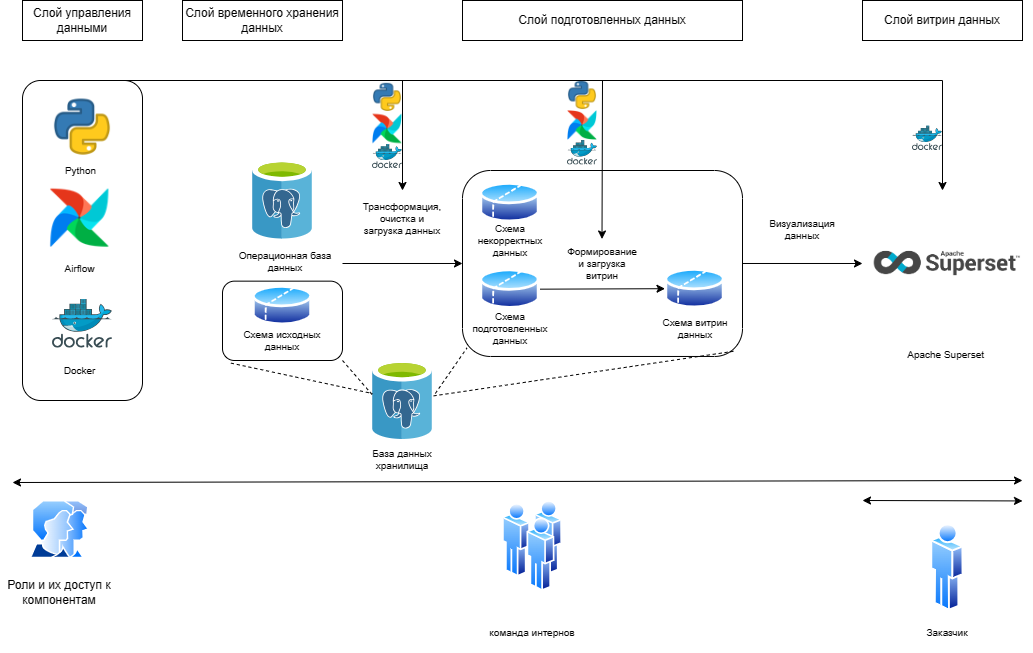
Оглавление

[1. СХЕМА АРХИТЕКТУРНОГО РЕШЕНИЯ. 3](#_Toc140263898)

[2. Описание компонентов. 4](#_Toc140263899)

[3. Описание слоев. 6](#_Toc140263900)

# 1. СХЕМА АРХИТЕКТУРНОГО РЕШЕНИЯ.



# 2. Описание компонентов.

2.1. Наименование баз данных и их элементов в информационной системе

|  |  |
| --- | --- |
| **База данных** | **Наименование** |
| Операционная база данных | internship\_sources |
| База данных хранилища | internship\_7\_db |

|  |  |  |
| --- | --- | --- |
| **База данных** | **Схема** | **Наименование** |
| Операционная база данных | Схема исходных данных | sources |
| База данных хранилища | Схема исходных данных | sources |
| Схема подготовленных данных | dds |
| Схема некорректных данных | data\_quality |
| Схема витрин данных | datamarts |

2.2. Версии используемого ПО.

|  |  |
| --- | --- |
| **Программное обеспечение** | **Версия программного обеспечения** |
| PostgreSQL | 12.10 |
| Python | 3.10 |
| Docker (клиент) | 24.0.2 |
| Docker (сервер) | 24.0.2 |
| Apache Airflow | 2.6.2 |
| Apache Superset | 2.1.0 |

2.3. Параметры подключения к компонентам.

Подключение к базам данных:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **База данных** | **Хост** | **Порт** | **Пользователь** | **Схема** |
| internship\_sources | 10.1.108.29 | 5432 | interns\_7 | sources |
| internship\_7\_db | 10.1.108.29 | 5432 | interns\_7 | sources |
| dds |
| data\_quality |
| datamarts |

Подключение к Apache Airflow:

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| URL вэб-интерфейса | http://localhost:8080/home |
| Логин | interns\_7 |
| Путь к ДАГам | ../docker/airflow/dags/ |
| Путь к исполняемому скрипту | ../docker/airflow/docker-compose.yml |

Подключение к Apache Superset:

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| URL вэб-интерфейса | http://localhost:9000/superset/welcome/ |
| Логин | interns\_7 |
| Путь к исполняемому скрипту | ../docker/superset/docker-compose-non-dev.yml |

2.4. Используемые сущности.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Слой** | **База данных** | **Схема** | **Сущность** | **Тип** |
| Слой временного хранения данных | internship\_sources | sources | brand | таблица |
| category | таблица |
| product | таблица |
| stock | таблица |
| transaction | таблица |
| internship\_7\_db | sources | product\_quantity | таблица |
| stores | таблица |
| stores\_emails | таблица |
| transaction\_stores | таблица |
| Слой подготовленных данных | internship\_7\_db | dds | brand | таблица |
| category | таблица |
| product | таблица |
| stock | таблица |
| transaction | таблица |
| product\_quantity | таблица |
| stores | таблица |
| stores\_emails | таблица |
| transaction\_stores | таблица |
| data\_quality | brand | таблица |
| category | таблица |
| product | таблица |
| stock | таблица |
| transaction | таблица |
| product\_quantity | таблица |
| stores | таблица |
| stores\_emails | таблица |
| transaction\_stores | таблица |
| upload\_tables\_tech | таблица |
| Слой витрин данных | internship\_7\_db | datamarts | orders\_data | таблица |
| stock\_data | таблица |
| stores\_data | таблица |
| orders\_data\_mart | представление |
| stock\_data\_mart | представление |
| stores\_data\_mart | представление |

# 3. Описание слоев.

Архитектурное решение представлено четырьмя слоями: слой управления данных, слой временного хранения данных, слой подготовленных данных, слой витрин данных.

В слое управления данным указаны технологии, осуществляющие обработку данных. Docker осуществляет контейнеризацию Airflow с Python. При этом, Airflow оркестрирует исполнение двух Python-скриптов, содержащих логику обработки данных, и двух python-скриптов, предназначенных для запуска DAG’ов. Первый скрипт предназначен для трансформации и очистки исходных данных и их последующую загрузку в схему подготовленных данных операционной базы данных. Данные, непрошедшие проверку качества попадают в таблицы логирования, размещенные на схеме некорректных данных.

Второй python-скрипт на основе полученных данных формирует и загружает витрины данных. Формирование витрин происходит в два этапа: сперва создаются таблицы соединения, а затем, на их основе, создаются представления. Образованные витрины данных визуализируются с помощью инструмента создания чартов и дашбордов Superset, который, в свою очередь, контейнеризирован с помощью Docker.

Слой временного хранения данных представлен операционной базой (internship\_sources), в которой хранятся исходные «сырые» данные и схемой «sources», размещенной на базе данных хранилища.

Слой подготовленных данных содержит базу данных хранилища (internship\_7\_db) с тремя схемами – схема подготовленных данных (dds), схема некорректных данных (data\_quality) и схема витрин данных (datamarts).

# ETL-процесс

## Содержание проекта

\* Директории:

1. AIRFLOW - содержит информацию об используемых переменных, конфигурационные файлы запуска Apache Airflow с помощью Docker, директории с DAG'ами и скриптами обработки данных.

2. POSTGRES - содержит DDL-скрипты, предназначенный для формирования всех сущностей DDS и datamarts-слоев.

4. DOCUMENTATION - содержит документацию по проекту.

\* Файлы:

1. AIRFLOW/.env - информация о переменных среды AIRFLOW\_UID и PYTHONPATH.

2. AIRFLOW/docker-compose.yml - измененный скрипт запуска Apache Airflow с помощью Docker с примонтированной директорией, хранящей python-скрипты.

3. AIRFLOW/variables.txt - используемые переменные в Apache Airflow.

4. AIRFLOW/dags/sources\_to\_dds.py - python-скрипт с DAG-ом, запускающим процедуру проверки качества данных.

5. AIRFLOW/dags/dds\_to\_dm.py - python-скрипт с DAG-ом, запускающим процедуру создания витрин данных.

6. AIRFLOW/dags/select\_data\_dag.py - python-скрипт с DAG-ом, запускающим процедуру возвращения заданного количества строк из целевой таблицы.

8. AIRFLOW/scripts/data\_quality.py - python-скрипт с классом для проверки качества данных и загрузки данных из одной базы данных в другую.

9. AIRFLOW/scripts/data\_marts.py - python-скрипт с классом для создания таблиц и витрин данных.

10. AIRFLOW/scripts/data\_quality\_params.json - набор параметров, используемый при наполнении данными dds-слоя.

11. AIRFLOW/scripts/data\_marts\_params.json - набор параметров, используемый при наполнении данными datamarts-слоя.

12. POSTGRES/DDL.sql - SQL-скрипт, с процедурами создания сущностей на слое DDS

13. POSTGRES/DDL.sql - SQL-скрипт, с процедурами создания сущностей на слое datamarts

14. DOCUMENTATION/АРХИТЕКТУРА РЕШЕНИЯ.docx - информация об архитектуре решения.

## Хранилище данных.

Хранилище данных представлено четырьмя слоями: слой управления данных, слой временного хранения данных,

слой подготовленных данных, слой витрин данных.

В слое управления данным указаны технологии, осуществляющие обработку данных. Docker осуществляет контейнеризацию Airflow с Python. При этом, Airflow оркестрирует исполнение двух Python-скриптов, содержащих логику обработки данных, и двух python-скриптов, предназначенных для запуска DAG’ов.

Первый скрипт предназначен для трансформации и очистки исходных данных и их последующую загрузку в схему подготовленных данных операционной базы данных. Данные, непрошедшие проверку качества попадают в таблицы логирования, размещенные на схеме некорректных данных.

Второй python-скрипт на основе полученных данных формирует и загружает витрины данных. Формирование витрин происходит в два этапа: сперва создаются таблицы соединения, а затем, на их основе, создаются представления. Образованные витрины данных визуализируются с помощью инструмента создания чартов и дашбордов Superset, который, в свою очередь, контейнеризирован с помощью Docker.

![Схема архитектурного решения](https://github.com/asetimankulov/internship/assets/98170451/52a84b9c-4a53-4255-94b0-24d0a8ed59cf)

## Трансформация, очистка и загрузка данных.

Первый набор python-скриптов, предназначенный для трансформации, очистки и загрузки исходных "сырых" данных, содержит DAG, запускающий процедуры

проверки качества загружаемых данных и их преобразования, и файл с классом проверки качества данных и процедурами загрузки и обработки данных.

DAG запускает исполнение 11 тасок. При этом, параллельное исполнение всех тасок нереализуемо в виду наличия ограничений на ссылочную целостность

связанных таблиц. Поэтому реализована следующая последовательность исполнения тасок:

![image](https://github.com/asetimankulov/internship/assets/98170451/631dce83-f961-49c1-99c1-703f14c922ce)

где:

\* end\_step - оператор, исполняющийся в случае успешного завершения всех тасок.

\* remove\_all\_data - оператор Postgres, очищающий все таблицы на слое DDS.

\* brand\_upload, category\_upload, stores\_upload, product\_upload, transaction\_stores\_upload, product\_quantity\_upload, stock\_upload, stores\_emails\_uload, transaction\_upload - операторы Python, исполняющие процедуры проверки качества соответствующих таблиц.

Модуль data\_quality содержит класс проверки данных таблицы по различным критериям качества данных.

Класс содержит следующие методы проверки качества данных:

\* noise\_restricts\_check - логирование и устранение "шумов" в данных. С помощью регулярного выражения выявляется наличие/отсутствие "шумов"

и происходит их устранение.

\* data\_types\_check - типы обрабатываемых данных проверяются на их соответствие полям таблицы загрузки. Некорректные данные заменяются

на пропуск и логируются.

\* missing\_values\_check - проверка наличия/отсутствия пропусков и их обработка. Отсутствие данных по полям первичного ключа приводит к устранению данных, в остальных случаях - данные остаются без изменений или заменяются на указанное значение.

\* duplicates\_check - устранение дубликатов по всем полям или первичному ключу. Некорректные данные логируются.

\* value\_restrict\_check - проверяет соответствие значения указанного поля таблицы некоторым ограничениям. Результат несоответствия логируется.

\* len\_restricts\_check - проверяет длину указанного поля и логирует те строки, которые нарушают ограничение.

\* ref\_integrity - проверка, устраняющая из проверяемой таблицы те строки по указанному полю, которые отсутствуют в связанном наборе данных.

Для каждой таблицы сформированы свои требования к качеству данных. Хранение набора параметров для каждой таблицы реализовано следующим образом:

"category": {

"missing": {

"drop": ["category\_id"],

"fill": {"category\_name": "Категория не определена"}

},

"duplicate": {

"drop": ["category\_id"],

"log": ["category\_id"]

},

"noise": {"category\_name": {"regex": "\_", "match\_replace": {"\_": " "}}},

"len\_restrict": {"category\_name": {"min": 2, "max": null}},

"data\_types": {

"category\_id": "text",

"category\_name": "text"

},

"ref\_integrity": null,

"val\_restrict": null

}

В случае с таблицей "category", проверка качества реализована следующим образом:

\* Обработка пропусков ("missing"): пропуск по "category\_id" приводит к удалению строки. Пропуски по полю "category\_name" заполняются значением "Категория не определена".

\* Обработка дубликатов: дублем считается строка, совпадающая по полю "category\_id". Они же логируются.

\* Обработка шумов ("noise"): значения поля "category\_name" проверяются регулярным выражением "\\_". Некорректный символ "\\_" заменяется на " ".

\* Обработка длины значений ("len\_restrict"): значение по полю "category\_name" будет залоггировано в случае, если его длина < 2.

\* Обработка типов данных ("data\_types"): значения полей "category\_id" и "category\_name" проверяются на соответствие типу данных text.

\* Обеспечение ссылочной целостности ("ref\_integrity"): нет наборов данных (таблиц), на которые бы ссылался какой-либо внешний ключ.

## Формирование и загрузка витрин.

Второй набор python-скриптов, предназначенный для формирования и загрузки витрин, содержит DAG, запускающий процедуры создания витрин, и файл с классом формирования таблиц, на основе которых будут созданы представления.

DAG запускает исполнение 7 тасок. При этом, осуществлено параллельное исполнение тасок, наполняющих таблицы, сформированных на основе данных из dds-слоя. Реализована следующая последовательность исполнения тасок:

![image](https://github.com/asetimankulov/internship/assets/98170451/2dc0d8ad-8cc8-48ec-8e6e-eef1c57fa855)

где:

\* remove\_all\_dm\_data - оператор Postgres, очищающий все таблицы на слое datamarts.

\* orders\_data\_upload, stock\_data\_upload, stores\_data\_upload - операторы Python, исполняющие процедуры наполнения соответствующих таблиц. При этом, формирование таблицы stores\_data осуществляется на основе выборки набора полей из наполняемой таблицы stock\_data\_upload.

\* create\_orders\_view, create\_stock\_view, create\_stores\_view - создание представления на основе наполняемых таблиц.

Модуль data\_marts содержит класс формирования таблиц данных, на основе которых формируются витрины данных.

Класс содержит следующие методы формирования таблиц:

\* create\_mart - формирование набора данных и его загрузка в целевую таблицу

\* orders\_data - формирование таблицы orders\_data

\* stock\_data - формирование таблицы stock\_data

\* stores\_data - формирование таблицы stores\_data

Для каждой таблицы сформированы свои параметры формирования. Хранение набора параметров для каждой таблицы реализовано следующим образом:

"stock\_data": {

"conn\_info": {

"from": {"conn\_id": "dds\_id", "schema": "dds"},

"to": {"conn\_id": "dm\_id", "schema": "datamarts"}

},

"join\_info": {

"source": {"table": "stock", "rename": true},

"joined\_tables": [

{"table": "stores",

"how": "inner",

"left\_on": ["stock\_pos"],

"right\_on": ["stores\_pos"],

"cast": null,

"rename": true},

{"table": "product",

"how": "inner",

"left\_on": ["stock\_product\_id"],

"right\_on": ["product\_product\_id"],

"cast": null,

"rename": true},

{"table": "category",

"how": "inner",

"left\_on": ["product\_category\_id"],

"right\_on": ["category\_category\_id"],

"cast": null,

"rename": true},

{"table": "brand",

"how": "inner",

"left\_on": ["product\_brand\_id"],

"right\_on": ["brand\_brand\_id"],

"cast": null,

"rename": true},

{"table": "product\_quantity",

"how": "inner",

"left\_on": ["stock\_product\_id"],

"right\_on": ["product\_quantity\_product\_id"],

"cast": null,

"rename": true},

{"table": "stores\_emails",

"how": "inner",

"left\_on": ["stock\_pos"],

"right\_on": ["stores\_emails\_pos"],

"cast": null,

"rename": true}

]

},

"column\_names": {

"stock\_available\_on": "Дата наличия товара",

"stores\_pos": "ID магазина",

"stores\_pos\_name": "Магазин",

"stock\_product\_id": "ID товара",

"product\_name\_short": "Товар",

"stock\_available\_quantity": "Доступное количество товара, шт.",

"stock\_cost\_per\_item": "Закупочная цена товара, руб.",

"available\_amount": "Сумма доступного остатка, руб.",

"category\_category\_id": "ID категории",

"category\_category\_name": "Категория",

"brand\_brand": "Бренд",

"update\_date": "Дата последнего обновления",

"load\_id": "ID процесса загрузки"

}

}

В случае с таблицей "stock\_data", процедура формирования реализована следующим образом:

\* Объединение исходных таблиц ("join\_info"): таблицей-источником служит "stock"("source"["table"]), наименование полей которой будет изменено ("rename": true) добавлением названия таблицы в качестве префикса. Далее происходит присоединение нескольких таблиц через операцию join. Присоединение таблицы "stores" происходит через inner join, где ключ объединения слева - stock\_pos, а справа - stores\_pos. Для осуществления присоединения изменения типов какого-либо поля не требуется ("cast": null). Происходит переименование полей ("rename": true).

\* Создание представления ("column\_names"): из объединения таблиц выбирается набор полей с последующим переименованием.