Программное обеспечение

Система управления данными Polyflow

РУКОВОДСТВО РАЗРАБОТЧИКА

Аннотация

Настоящий документ является руководством разработчика системы управления данными Polyflow.

Документ разработан в соответствии с требованиями ГОСТ Р 59795-2021 «Требования к содержанию документов».

Содержание

Введение

Модуль Polyflow представляет собой сервис оркестровки, сбора и обработки разнородных данных в хранилищах произвольной архитектуры.

Уровень подготовки персонала, необходимого для работы с ИАС, предполагает наличие следующих групп пользователей:

- Служба эксплуатации ИАС;
- Разработчики;
- Операторы.
- Служба эксплуатации ИАС

В службу эксплуатации ИАС входят специалисты следующих категорий: «Администратор защиты (безопасности) информации», «Администратор операционных систем», «Администратор баз данных».

- Администратор защиты (безопасности) информации обеспечивает:
    - Формирование списка пользователей, допущенных к работе с Системой;
    - Настройку учетных записей пользователей, управление ролями доступа, а также интеграция пользователей с помощью LDAP;
    - Формирование матрицы доступа к ресурсам Системы и данным, а также изменение прав доступа.
- Администратор операционных систем отвечает за:
    - Установку компонентов платформы, активацию и первоначальную настройку.
    - Сопровождение ИАС (тестирование работоспособности, восстановление и т.п.), обновление версий (анализ необходимости перехода на новые версии, разработку перечня мероприятий по переводу на новую версию).
- Администратор баз данных отвечает за:

- Генерацию систем управления базами данных;
- Сопровождение и управление информационными ресурсами;
- Сохранение резервных копий, восстановление искаженной информации, архивирование информации и организацию поступления информации из архива;
- Обработку и анализ статистической информации о характере и интенсивности использования данных, о распределении нагрузки на различные компоненты структуры баз данных, внесение изменений в структуру баз данных в процессе эксплуатации Системы с целью повышения производительности, обеспечивает ввод и поддержание в актуальном состоянии общих разделов баз данных (классификаторов).

Служба эксплуатации обеспечивает функционирование в штатном режиме технических и программных средств АИС, отслеживает процессы наполнения АИС данными.

Поддержка функционирования Системы должна осуществляться силами действующей Службы эксплуатации АИС, состоящей из специалистов, обладающих знаниями в области информационных и сетевых платформ, на которых реализована АИС, и опытом администрирования баз данных.

Разработчики

Разработчиками являются специалисты, которые участвуют в процессах разработки процессов загрузки данных в Систему, описания сущностей-получателей и источников, проведения анализа работы процессов, выявления аномалий и их причин.

Разработчики должны иметь опыт разработки в своей отрасли, обладать навыками работы с DWH решениями и базовыми знаниями SQL и Python.

Операторы

Операторами являются специалисты, которые участвуют в процессах сбора данных, базовой настройки, запуска и мониторинга таких процессов.

Для работы с системой необходимо ознакомиться со следующим набором эксплуатационной документации:

- Руководство администратора;
- Руководство разработчика;
- Руководство пользователя.

1 Назначение и условия применения

1.1 Назначение системы

Модуль Polyflow предназначен для решения следующих задач:

- сбор данных из разных источников:
  - файлы
  - внешние системы
  - базы данных
- описание сущностей-получателей и источников;
- мониторинг и управление выполняемыми процессами;
- контроль качества данных;
- трансформация данных.

1.2 Условия применения

Для функционирования АИС необходимо следующее программно-аппаратное обеспечение:

1.2.1 Серверная часть

Минимальные требования к серверному оборудованию следующие:

- CPU 2 vCPU (2.8 ГГц и выше);
- RAM 8 ГБ,
- HDD 10 ГБ

Ориентировочная формула для подсчета конфигурации в зависимости от количества процессов при использовании Local Executor: дополнительно к минимальным системным требованиям необходимо RAM 256-512МБ CPU 0.1 vCPU в среднем на каждый процесс.

Операционная система: Astra Linux Special Edition 1.6 (Смоленск) или аналог.

Права пользователя, разворачивающего приложение: user - non-root with sudo privileges.

Дополнительные требования к установленным приложениям: Docker версии 20.10.0 и до 25, Docker -compose версия 1.29 и выше.

1.2.2 Локальная сеть

Все компоненты платформы должны находиться в одной подсети или должна обеспечиваться прозрачная маршрутизация. Не рекомендуется использовать NAT. В рамках ознакомления рекомендуется отключить брандмауэры. Внутри локальной сети между всеми компонентами не должно быть ограничений по передаче данных. Для доступа из внешней сети достаточно открыть порт, используемый Polyflow (порт задается при установке). При использовании системы с установленными антивирусами или комплексными системами защиты необходимо обеспечить свободную работу, сетевую активность и взаимодействие компонентов.

2 Подготовка к работе

2.1 Состав программного обеспечения

Модуль Polyflow поставляется в виде нескольких файлов:

- образы Docker, содержащие в себе все компоненты с уже настроенным окружением и всеми внутренними зависимостями;
- python-файл manage.py для быстрого развёртывания и настройки модуля;
- дополнительные файлы, используемые при развертывании: __init__.py, docker-compose.yml.tmpl, airflow.cfg.tmpl.

Примечание: Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы. Суть и одно из предназначений Docker такое же, как и у виртуальных машин — это изоляция работы различных конфликтующих программ внутри одного сервера. Наглядно увидеть отличия между виртуальной машиной и контейнером можно, пройдя по ссылке: https://www.docker.com/what-docker.

Таким образом, установка сводится к двум шагам:

1. Установка docker-engine внутри операционной системы;

2. Запуск python-файла manage.py для разворачивания готовых контейнеров с компонентами системы.

Установка и настройка системы описаны в документе «Руководство администратора». Основные операции с интерфейсом описаны в документе «Руководство пользователя».

2.2 Запуск системы

2.2.1 Начало работы

Для того, чтобы начать работу в Polyflow необходимо произвести аутентификацию в системе. Для этого введите в браузере адрес машины, на котором установлено решение, например https://127.0.0.1:8080 (порт

указывается тот, который был задан при установке). На появившейся форме укажите свой логин и пароль (пароль администратора задаётся при установке системы, пароль пользователя – получается у администратора), и нажмите кнопку «Вход» в соответствии с Рисунок 1.

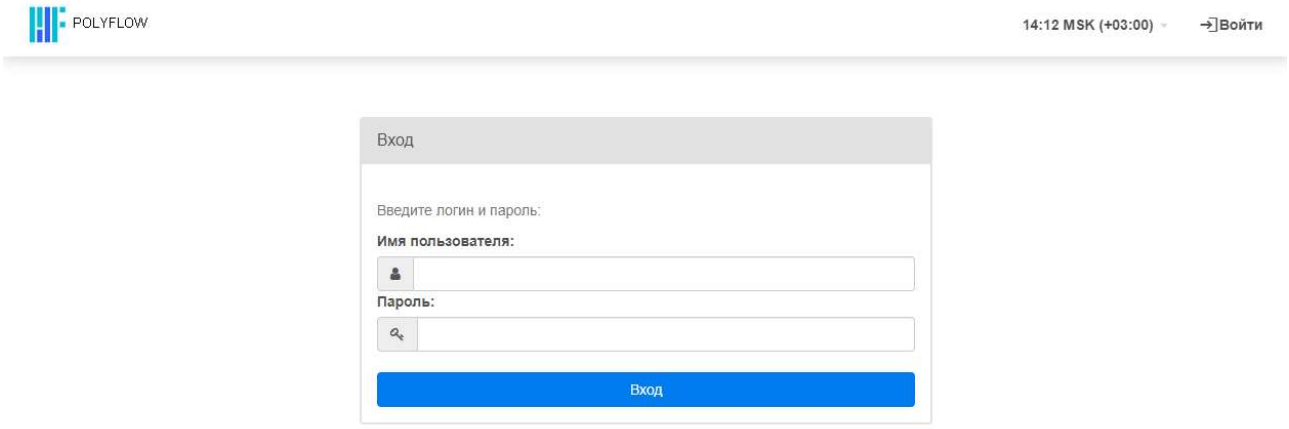

Рисунок 1. Начальная страница

2.3 Порядок проверки работоспособности

Для проверки работы Polyflow необходимо произвести аутентификацию в системе. После успешной аутентификации откроется главная страница приложения, со списком доступных процессов (Рисунок 2). Необходимо выбрать один из них и перейти на форму просмотра информации о выбранном процессе, кликнув на его название (Рисунок 3).

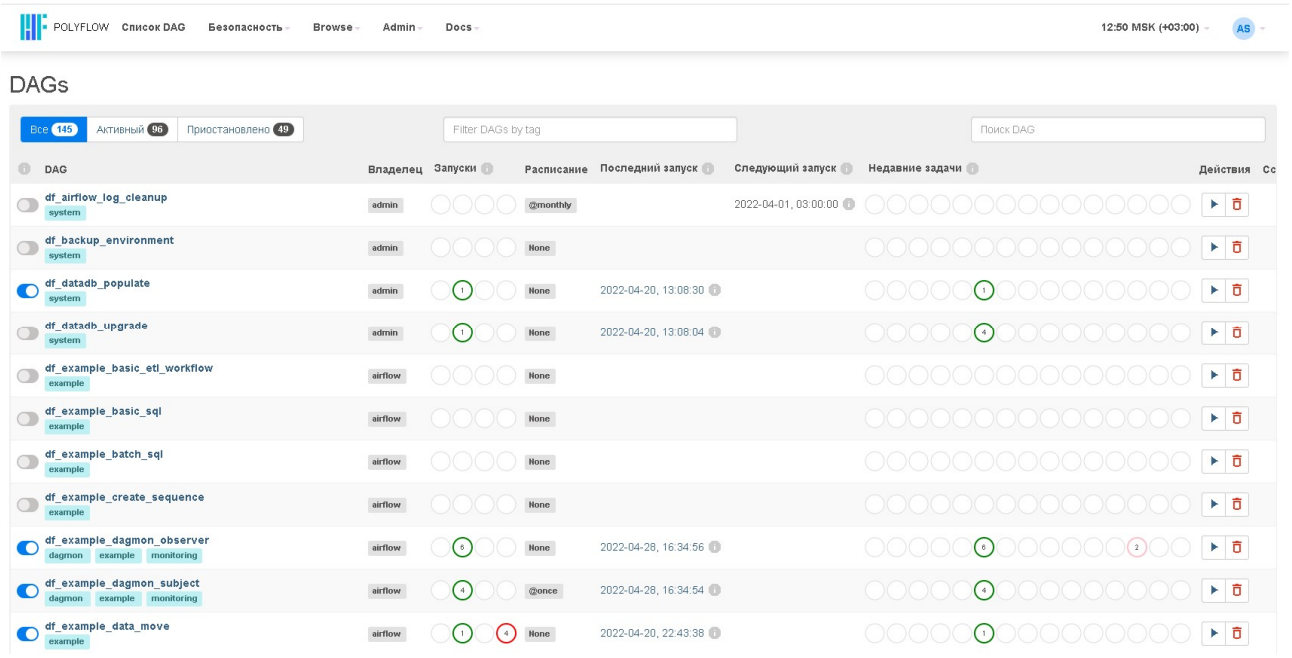

Рисунок 2. Список доступных процессов

Будет открыта форма информации и управления выбранным DAG. По умолчанию открыта вкладка График (Рисунок 3).

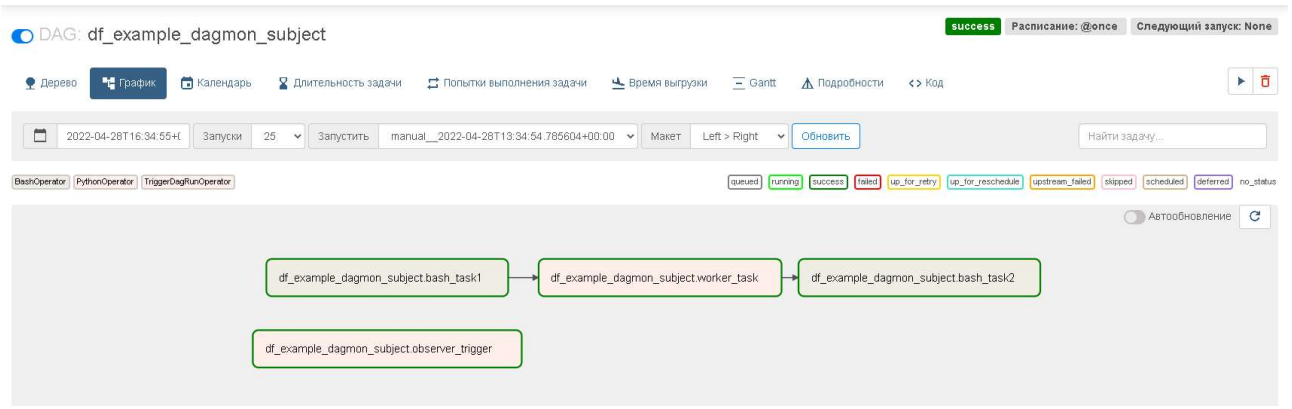

Рисунок 3. Форма просмотра информации о процессе

3 Описание операций

3.1 Определения и сокращения Polyflow

Определения и сокращения Polyflow представлены в таблице 1.

Таблица 1. Определения Polyflow

Термин/Сокращение	Определение
Аутентификация	Проверка принадлежности пользователю указанного им пароля.
Пользователь	Авторизованный пользователь, учетная запись которого позволяет просматривать данные на портале.
Веб-интерфейс	Сайт в компьютерной сети, который предоставляет пользователю интерактивный интернет-сервис, который работает в рамках этого сайта.
Планировщик (Airflow Scheduler)	Компонент, который отслеживает состояние DAG и запускает задачи, зависимости которых были удовлетворены. После запуска системы планировщик работает непрерывно, чтобы отслеживать и синхронизировать папку, содержащую объекты DAG.
Хранилище данных (англ. Content Repository, Data Warehouse, DWH)	Предметно-ориентированная информационная база данных, сочетающая в себе функции системы управления версиями, поисковой машины и СУБД.
Система управления данными Polyflow	Сервис оркестровки сбора и обработки разнородных данных хранилища произвольной архитектуры.
Polyflow	Краткое наименование программного обеспечения «Система управления данными Polyflow»
DAG (Directed Acyclic Graph)	Смысловое объединение задач, которые необходимо выполнить в строго определенной последовательности согласно указанному расписанию.
Task	Операции, применяемые к данным, например: загрузка данных из различных источников, их

	агрегирование, индексирование, очистка от дубликатов, сохранение полученных результатов и прочие ETL-процессы.
OpenID	Открытый стандарт децентрализованной системы аутентификации, предоставляющей пользователю возможность создать единую учётную запись для аутентификации на множестве не связанных друг с другом интернет-ресурсов, используя услуги третьих лиц.
Operator	Сущность, на основе которой создаются экземпляры заданий, где описывается, что будет происходить во время исполнения экземпляра задания.
Sensor	Тип оператора, позволяющий описывать реакцию на определенное событие.
ETL	(от англ. Extract, Transform, Load – дословно «извлечение, преобразование, загрузка») – один из основных процессов в управлении хранилищами данных.
SLA	(от англ. Service Level Agreement) – соглашение об уровне сервиса.

4 Принципы реализации процессов на Polyflow

4.1 Общие

Все проектные артефакты должны храниться в репозитории, локальном либо централизованном. Комментарии в коде и к коммитам должны быть на русском языке. В README к проекту должны быть указана основная ветка репозитория, если это не master (main). Секреты стендов в промышленной эксплуатации в репозиторий коммититься не должны. Для подключения к внешним системам должны использоваться Airflow Connection, а не переменные окружения, константы или иные, небезопасные plain-text источники. При наличии системных модулей всегда использовать их через импорт, а не копированием. Доработки, необходимые более чем для одного проекта рекомендуется реализовывать как системные. Типовые процессы, которые можно использовать как шаблоны при реализации собственных, представлены в `volumes/workspace/dags/df/examples` и `volumes/workspace/dags/df/tests`.

4.1.1 ETL

Все задачи по обработке данных должны выполняться в `DockerOperator` (образ - `df_operator`). Остальные операторы и сенсоры допустимо использовать для задач следующих типов: проверка существования ресурсов (`LocalFileSensor`), перемещение/удаление файловых объектов (`BashOperator`), контроль задач (`TriggerDagRunOperator`, `ExternalTaskSensor`, `LatestOnlyOperator`), операции управления планами загрузки Visiology (`VisiologyAPIOperator`), нетребовательные вспомогательные операции (`PythonOperator` и производные от него операторы управления workflow).
Поддержка прочих операторов не осуществляется.
Не допускается установка дополнительных модулей в образы `df_airflow`, ни в базовый, ни в локальный, как действие, приводящее к необходимости внеплановых обновлений и перезапусков сервиса, конфликтам модулей,

падениям, зависаниям, внутрисистемной конкуренции процессов и общему снижению стабильности работы системы.

4.1.2 DWH

Все сущности базы данных должны быть описаны метаданными и созданы на их основе средствами Polyflow. Создание сущностей скриптами допускается только в случае невозможности их описания в метаданных. На отсутствующий функционал должна быть запрошена доработка, создана соответствующая задача в Polyflow. Изменения в структуре сущностей должны быть отражены в метаданных, сами изменения при этом могут, а в некоторых случаях и должны, выполняться руками.

4.2 Описание файлов и структуры проекта

Вся работа происходит в папке projects, содержимое которой линкуется с папкой workspace.

Для инициализации структуры проекта необходимо выполнить команду python3 manage.py --init-project <Имя проекта>. Если имя проекта не задано, то будет использовано значение default. После выполнения команды будут созданы следующие папки:

- `dataflow/volumes/projects/<Имя проекта>/cache` - директория для загруженных/закэшированных файлов
- `dataflow/volumes/projects/<Имя проекта>/dags` - директория для дагов
- `dataflow/volumes/projects/<Имя проекта>/metadata` - директория для метаданных сущностей и прочих объектов
- `dataflow/volumes/projects/<Имя проекта>/plugins` - директория для плагинов
- `dataflow/volumes/projects/<Имя проекта>/share` - директория для данных для обработки

- `dataflow/volumes/projects/<Имя проекта>/source` - директория для исходников

4.2.1 Структура исходников

- `source` - основная директория
    - `df` - директория с системным кодом проекта (библиотеками), внутри которой располагаются следующие:
        - `common` - директория с общим кодом проекта (библиотеками)
        - `excel` - экстракт из датасетов в табличном формате (excel, netdb)
        - `python` – различные раннеры
        - `qc` - директория объектов контроля качества
        - `sql` - директория sql объектов:
            - `scripts` – выполняемые скрипты
            - `ddl` – скрипты миграции
            - `callables` – процедуры

- `metadata` - основная директория метаданных, может представлять собой иерархию
- `dags` - основная директория, может представлять собой иерархию, но в веб-интерфейсе всегда имеет плоский вид, где видимость DAG'ов регулируется правами
    - `examples` - директория с примерами настройки DAG'ов, можно скрыть из веб-интерфейса с помощью файла `.airflowignore` в директории `volumes`
    - `tests` - директория с тестовыми DAG'ами, можно скрыть из веб-интерфейса с помощью файла `.airflowignore` в директории `volumes/workspace/dags`

Для сохранения своей версии `.airflowignore` можно воспользоваться следующей командой: `git update-index --assume-unchanged ./volumes/workspace/dags /.airflowignore`

4.3 Правила именования DAG'ов и task'ов

- Системные DAG'и и task'и должны иметь префикс `df_`
- SubDAG'и должны иметь префикс `<id_родителя>`.
- DAG'и task'и в папке examples должны иметь префикс `df_example_`
- DAG'и task'и в папке tests должны иметь префикс `df_test_`
- DAG'и task'и в папке tests, относящиеся к проектам, должны иметь префикс `test_<имя_проекта>_`
- Task'и должны иметь уникальные id, что достигается, например, добавлением префикса с id DAG'a: `<dag_id>.<task_id>`

4.4 Операторы

Как правило, DAG'и передают в докер оператор следующие параметры, в качестве переменных окружения:

- `AF_DAG_ID`: dag.dag_id

- `AF_RUN_ID`: run_id

- `AF_EXECUTION_DATE`: ds

- `AF_EXECUTION_TS`: ts

- `AF_DF_NOW`: df_now - текущая дата со временем в UTC (например, `2019-04-14T07:46:24.850413`)

- `AF_TASK_ID`: task.task_id

- `AF_JOB_ID`: ti.job_id

Дополнительные, часто используемые, параметры:

- `AF_TASK_OWNER`: task.owner

- `AF_PDAG_ID`: идентификатор родительского DAG (значение поля id из таблицы df.op для родительской операции); передается, если необходимо заполнить поле parent_id в таблице df.op для текущего DAG; [пример DAG'a с заполнением AF_PDAG_ID](operator/ws/dags/df/examples/pdagid/triggerer.py)

- `AF_PARAMS`: параметры запуска DAG'a

- все параметры со значениями по умолчанию должны быть заданы в `params`

- если `AF_PARAMS` содержит валидный JSON, то он также доступен в поле `params` из таблицы `df.op_ctx` по `AF_OP_ID`

[пример DAG'a с пользовательскими параметрами запуска](operator/ws/dags/df/examples/sql/parameterized_dagrun.py)

- `AF_PRODUCER`: путь к описанию источника/ов данных, например, файла, относительно папки метаданных

- `AF_CONSUMER`: путь к описанию получателя/ей данных, например, таблицы, относительно папки метаданных

- `AF_SKIPER`: строка, задающая путь до пользовательской функции для управления выполнением таска. Может быть задана как:

- путь до jinja-шаблона с python-кодом и имя функции, указанное через `::`, например - `/app/ws/dags/df/tests/xlsx/extract_excel_const_skiper.py::get_exit_code`.
Переданное значение валидируется по паттерну `^(?:dags|source)\.((?:\w)+\.)+\w+$`. В шаблоне доступны локальные переменные и переменные окружения. Заданная функция будет использована для получения кода завершения работы таска: должна возвращать целое число `n`. На вход функция не принимает параметры. Совместно с параметром `skip_on_exit_code` позволяет пропустить выполнение таска, завершить выполнение таска с ошибкой или выполнить его в обычном режиме:

- `skip_on_exit_code` `not None`:

  - `n > 0`:

    - `n` указан в `skip_on_exit_code` - пропуск выполнения таска

    - `n` не указан в `skip_on_exit_code` - ошибка выполнения таска

  - `n = 0` - успешное завершение работы таска

- `skip_on_exit_code` `is None`

  - `n != 0` - ошибка выполнения таска

  - `n = 0` - успешное завершение работы таска

Не рекомендуется в `skip_on_exit_code` указывать значение 1, т.к. в случае ошибки таск будет помечен как пропущенный.

- путь импорта модуля из папки `dags` или `source`, например - `dags.df.tests.xlsx.extract_excel_const_skiper.get_exit_code`. Переданное значение валидируется по паттерну `^(?:dags|source)\.((?:\w)+\.)+\w+$`

- `AF_SCRIPT_PATH`: путь к директории с SQL скриптами или к непосредственно к файлу

- `AF_NOCOUNT`: флаг, определяющий необходимость выполнять SET NOCOUNT ON перед выполнением sql-скриптов в SQL Server.

True - значение по умолчанию, т.е. SET NOCOUNT ON будет выполнено

- `AF_SCRIPT_RE`: регулярное выражения отбора файлов, если в `AF_SCRIPT_PATH` указана директория

- `AF_METADATA_PATH`: путь к директории с метаданными CDM или к непосредственно к файлу

- `AF_METADATA_RE`: регулярное выражения отбора файлов, если в `AF_METADATA_PATH` указана директория

- `AF_METADATA_DEPTH`: параметр, определяющий максимальную глубину поиска файлов метаданных:

- если его значение равно 1, то смотрим только в `AF_METADATA_PATH` без подкаталогов.

- если 2, то `AF_METADATA_PATH` и его вложенные каталоги.

- если 3, то смотрим до 3 уровня, и так далее.

- 0 - значение по умолчанию, поиск в `AF_METADATA_PATH` и всех уровнях его подкаталогах

- `AF_JSON_PATH`: путь к директории с JSON наполнением таблиц

- `AF_JSON_RE`: регулярное выражения отбора файлов, если в `AF_JSON_PATH` указана директория

- `AF_PROCEDURE_PATH`: путь к SQL процедуре для запуска

- `AF_SINGLETRAN`: default = False, позволяет выполнять sql стейтменты одного файла в одной транзакции

- `AF_ISOLATION_LEVEL`: default = None, позволяет задать уровень изоляции транзакций при использовании `source/df/sql/run_statement.py`. Доступные значения: `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ` и `SERIALIZABLE`. Например, для выполнения запроса `VACUUM <имя таблицы>;` необходимо передать `AF_SINGLETRAN` = True и `AF_ISOLATION_LEVEL` = 'AUTOCOMMIT'

- `AF_IS_SELECT`: default - False, позволяет явно запросить возврат результата sql-запроса типа select при использовании `source/df/sql/run_statement.py`, `source/df/sql/run.py`

- `AF_RULES_PATH`: путь к директории с правилами для загрузки в базу DWH

- `AF_RULES`: список правил для загрузки в базу DWH

- `AF_DF_CONNECTION`: подключение к базе данных с системной схемой `df` (доступно только в DockerOperator)

- `AF_DWH_DB_DIALECT`: диалект SQL, используемый СУБД с данными

- `AF_DWH_DB_NAME`: наименование БД с данными

- `AF_DWH_DB_SCHEMA`: схема с данными

- `AF_DWH_DB_USER`: пользователь для подключения к БД с данными

- `AF_DWH_DB_PASSWORD`: пароль пользователя для подключения к БД с данными

- `AF_DWH_DB_SERVER`: сервер СУБД

- `AF_DWH_DB_PORT`: порт сервера СУБД

- `AF_LOGLEVEL`: уровень логирования в операторе, возможные значения: 'error', 'warning', 'info', 'debug', значение по умолчанию - 'info'

- `AF_S_`: автоматически генерируемый параметр на основе `AF_DWH_DB_SCHEMA`, равен `AF_DWH_DB_SCHEMA`+`.`, либо '', если `AF_DWH_DB_SCHEMA` не задана

- `AF_SC_`: автоматически генерируемый параметр на основе `AF_DWH_DB_SCHEMA`, равен `AF_DWH_DB_SCHEMA`+`.`-`<имя_базы>`, либо `' '`, если `AF_DWH_DB_SCHEMA` не задана

- `AF_DWH_DB_SCHEMA*`: схема с данными, где `*` постфикс, позволяющий задавать более одной схемы, для которой автоматически генерируются соответствующие `AF_S*_` и `AF_SC*_`

- `AF_OP_ID`: идентификатор записи из таблицы `df.op` с информацией о выполняемом процессе, дополнительные данные записываются в `df.op_ins` и `df.op_ctx`, включая параметры запуска процесса из `params` (требует наличия `AF_PARAMS`)

- `AF_EXTRACT_FORMULAS`: позволяет загружать формулы вместо сохраненных результатов предыдущего расчета из Excel

- `AF_VERBOSE`: используется для вывода дополнительной информации о версии основных установленных модулей, загружаемом файле, типе и параметрах используемого ридера.

Каждый исполнитель должен получать тот набор параметров, который ему необходим.

Неизвестные параметры должны просто игнорироваться.

Опциональные параметры, в случае их отсутствия, должны принимать значения по умолчанию.

В докер-операторах можно использовать язык движка шаблонов Jinja2..

Рекомендуется ограничивать потребление ресурсов хоста docker операторами, например, используя параметры cpus, mem_limit, execution_timeout

4.4.1 Обработка результатов SQL-запросов

Для управления форматом возвращаемых данных при использовании раннеров `sql/run.py` и `sql/run_statement.py` с `AF_IS_SELECT = True` доступны следующие переменные:

- **`AF_SELECT_TYPE`** (по умолчанию `'object'`)

Определяет тип возвращаемого набора данных:

- `'single'` — возвращает значение из первой строки первого столбца.

Пример: `select 1` → `1`

- `'single_quoted'` — возвращает значение из первой строки первого столбца в одинарных кавычках.

Пример: `select 'текст'` → `'текст'`

- `'list'` — возвращает список значений из всех столбцов первой строки, разделённых запятыми.

Пример: `select 1, 2, 3` → `1,2,3`

- `'list_quoted'` — возвращает список значений в одинарных кавычках из всех столбцов первой строки, разделённых запятыми.

Пример: `select 'строка1', 'строка2', 'строка3'` → `'строка1','строка2','строка3'`

- `'object'` — текущее поведение (значение по умолчанию). Возвращает результат в виде списка кортежей.

Пример: `select 1` → `[(1,)]`, `select 1, 2, 3` → `[(1, 2, 3)]`

- **`AF_RETURN_HANDLER`** (опционально)

Путь к пользовательской функции-обработчику результата. Функция получает на вход результат SQL-запроса и должна вернуть обработанные данные.

Пример значения: `'source.df.common.helpers.my_handler'`

- **`AF_RETURN_KWARGS`** (опционально)

Именованные аргументы в формате JSON, которые будут переданы в пользовательский обработчик `AF_RETURN_HANDLER`.

Пример значения: `'{"key": "value"}'`

4.4.1.1.1 Приоритет обработки:

1. Если задан `AF_RETURN_HANDLER`, результат передаётся в указанную функцию.

2. Если `AF_RETURN_HANDLER` не задан или недоступен, используется встроенная обработка в соответствии с `AF_SELECT_TYPE`.

3. Если `AF_SELECT_TYPE` не задан или имеет значение `'object'`, возвращается результат в виде списка кортежей.

4.4.1.2 Заполнение системных таблиц df.*

При использовании [системных раннеров](#основные-исполняемые-файлы) доступно автоматическое сохранение информации о выполняемых процессах и задачах в [системных таблицах `df.*`](#описание-системных-таблиц). Для этого в контейнере доступных следующие переменные окружения:

- `AF_DAG_ID`: идентификатор вызываемого DAG'a (значение `'{{ dag }}'`),

- `AF_RUN_ID`: идентификатор запуска (значение `'{{ run_id }}'`),

- `AF_TASK_ID`: идентификатор выполнения задачи DAG (значение `'{{ task.task_id }}'`),

- `AF_MAP_INDEX`: индекс выполняемой задачи (значение `{{ ti.map_index }}`),

- `AF_TRY_NUMBER`: номер попытки выполнения задачи в рамках данного запуска (значение `{{ ti.try_number }}`),

- `AF_JOB_ID`: идентификатор задания (значение `'{{ ti.job_id }}'`)

4.5 Extract

4.5.1 Общие положения

Управляет передачей данных из источника в получатель экстрактор (extractor). В качестве источников и получателей данных используются сущности модели CDM.

Все сущности имеют общий постфикс - Entity.
Для передачи данных в получатели используются аплоадеры (uploaders). Типы сущностей-получателей данных содержат в своем наименовании инфикс Local, при этом не обязательно принадлежат одному семейству.

Для получения данных из источников используются адаптеры (adapters).

Типы сущностей-источников, в зависимости от природы источника, в своем наименовании могут содержать инфиксы Web, File, но никогда Local. Таким образом, аплоадеры всегда взаимодействуют только с Local типами сущностей, а адаптеры - со всеми остальными, т.е. с внешними системами.

4.5.1.1 Аннотации сущности типа LocalEntity

Поддержка зависит от конкретного обработчика.

- `precreateEntity`, `default = False`, автоматически создавать сущность (таблицу)
- `ignoreExistingTable`, `default = True`, если включено, пропустить создание уже существующей таблицы
- `purgeExistingData`, `default = False`, очищать существующие данные перед добавлением новых, используется каскадный

`truncate` для postgresql и `delete` в mssql для таблицы дополнительных свойств

- `enforceStrictMapping`, `default = True`, требовать, чтобы в сущности (таблице) присутствовали все атрибуты модели
- `schema`, `default = None`, схема, к которой принадлежит таблица в базе данных, по умолчанию используется схема из `AF_DWH_DB_SCHEMA`. Если указана схема локальной сущности, то она переопределяет схему по умолчанию, а схема ссылочной сущности переопределять схему локальной сущности
- `useBulkInsert`, `default = True`, использовать пакетную вставку при загрузке данных в таблицу БД
- `recreateIfOutOfSync`, `default = False`, пересоздавать таблицу БД в случае несоответствия ее структуры в части полей и их типов описанию, обрабатывается только при `enforceStrictMapping = True`

4.5.1.2 Поддерживаемые аннотации атрибутов сущности типа LocalEntity

- `role`, `default = None`, роль (поведение) атрибута
- `primaryKey`, `default = False`, если включено, атрибут включается в первичный ключ, если такой атрибут один и его тип integer или int64, то он создается как identity(1,1)
- `logicalKey`, `default = False`, если включено, атрибут входит в логический ключ
- `sequence`, `default = None`, наименование сиквенса, если задано и атрибут с `primaryKey = True`, то дополнительно к таблице создается объект sequence с указанным наименованием
- `sequenceCache`, `default = None`, размер кэша сиквенса (`integer`)
- `default`, `default = None`, значение по умолчанию, в текущей версии поддерживаются только строковые значения

- `nullable`, `default = True`, может ли атрибут содержать пустые (`null`) значения
- `unique`, `default = False`, если включено, значения атрибута должны быть уникальны
- `softNullable`, `default = False`, переопределяет поведение `nullable = False`, т.е. если данная аннотация включена - не создается `Not Null Constraint`
- `softUnique`, `default = False`, переопределяет поведение `unique = True`, т.е. если данная аннотация включена - не создается `Unique Constraint`
- `index`, `default = False`, если включено, по атрибуту строится индекс, есть ограничения, например, для строкового типа должна быть указана размерность
- `length`, `default = None`, размерность для типов `string`, `varchar` и `unicode`
- `format`, `default = None`, формат поля, частичная поддержка регулярных выражений
- `precision`, `default = 28`, общее максимальное количество знаков, используется для типа `decimal`
- `scale`, `default = 10`, количество знаков после запятой используется для типа `decimal`

4.5.1.3 Поддерживаемые типы данных полей

- `string`, соответствует `nvarchar(x)`
- `unicode`, соответствует `nvarchar(x)`
- `varchar`, соответствует `varchar(x)`
- `text`, соответствует `clob`
- `guid`, соответствует `uuid`
- `int64`, соответствует `integer`
- `integer`, соответствует `integer`

- `date`, соответствует дате без времени
- `dateTime`, соответствует дате со временем
- `timestamp`, соответствует дате со временем с миллисекундами
- `decimal`, соответствует числу с десятичными знаками, точность по умолчанию: `28,10`
- `boolean`, соответствует `True|False` либо `1|0`
- `float`, соответствует `float`

4.5.1.4 Свойства по умолчанию атрибутов по ролям (тип атрибута может быть переопределен заданием dataType)

- `id`: `integer, primary key`, суррогатный ключ
- `code`: `nvarchar(32), primary key`, читаемый идентификатор записи
- `rid`: `guid, primary key`, идентификатор строки
- `sid`: `varchar(128), not null`, идентификатор источника
- `runid`: `varchar(128), not null, index`, идентификатор запуска
- `jobid`: `integer, not null, index`, идентификатор задания
- `datebegin`: `datetime, not null`, дата начала периода действия
- `dateend`: `datetime, not null`, дата окончания периода действия
- `unitid`: `nvarchar(128), not null`, идентификатор структурного подразделения
- `cd`: `date, default date`, дата записи (created date)
- `cdt`: `date, default date`, дата записи (created datetime)
- `cts`: `timestamp, default utcnow`, дата и время записи (created timestamp) (подразумевается дата и время создания объекта, а не записи в техническом плане)
- `tag`: `integer`, битовая маска в десятичной системе, используется для тегирования данных

- `fk`: `guid`, внешний ключ, роль добавляется автоматически для связанных сущностей, поддерживается загрузкой в несколько сущностей за экстракт

4.5.1.5 Поддерживаемые аннотации связей типа SingleKeyRelationship

- `generateKey`, `default = False`, генерация внешних ключей в БД при создании таблиц по метаданным
- `onUpdate`, поведение при изменении внешнего ключа, допустимые значения, поддержка и имплементация зависят от СУБД
- `onDelete`, поведение при удалении внешнего ключа, допустимые значения, поддержка и имплементация зависят от СУБД

4.5.1.6 Аннотации сущности типа ViTableEntity (поддержка зависит от конкретного обработчика)

- `precreateEntity`, `default = False`, автоматически создавать сущность (таблицу)
- `ignoreExistingTable`, `default = True`, если включено, пропустить создание уже существующей таблицы
- `purgeExistingData`, `default = False`, очищать существующие данные перед добавлением новых, используется каскадный `truncate` для postgresql и `delete` в mssql для таблицы дополнительных свойств
- `enforceStrictMapping`, `default = True`, требовать, чтобы в сущности (таблице) присутствовали все атрибуты модели
- `schema`, `default = None`, база, к которой принадлежит таблица, по умолчанию используется схема из `AF_DWH_DB_SCHEMA`. Если указана схема локальной сущности, то она переопределяет схему по умолчанию

- `recreateIfOutOfSync`, `default = False`, пересоздавать таблицу БД в случае несоответствия ее структуры в части полей и их типов описанию, обрабатывается только при `enforceStrictMapping = True`

4.5.1.7 Поддерживаемые аннотации атрибутов сущности типа LocalEntity

- `role`, `default = None`, роль (поведение) атрибута
- `primaryKey`, `default = False`, если включено, атрибут включается в первичный ключ, если такой атрибут имеет целочисленный тип, то он создается как `identity(1,1)`
- `sequence`, `default = None`, наименование сиквенса, если задано и атрибут с `primaryKey = True`, то дополнительно к таблице создается объект sequence с указанным наименованием

4.5.1.8 Поддерживаемые типы данных полей

- `string`, соответствует `nvarchar(x)`
- `integer`, соответствует `integer` (дополнительно поддерживаются `short`, `long`)
- `time`, соответствует времени
- `date`, соответствует дате без времени
- `dateTime`, соответствует дате со временем, при взаимодействии с API используется `'datetime'`
- `real`, соответствует `real`
- `boolean`, соответствует `true|false`
- `float`, соответствует `float`

4.5.1.9 Свойства по умолчанию атрибутов по ролям (тип атрибута может быть переопределен заданием dataType)

- `id`: long, primary key, суррогатный ключ, для которого автоматически создается sequence с наименованием в формате `<table_name>_<field_name>_sq`, если sequence не задан явно.

4.5.1.10 Сущность типа `XComEntity`

Сущность предназначена для вывода данных, получаемых от аплоадера (uploader), в XCom, например, для последующего использования в тасках.

Для получения данных из XCom (Context) можно воспользоваться методом `context_pull(default: Optional[Any] = None) -> Any` или стандартным методом `ti.xcom_pull`.

Аннотации сущности:

- `schema`, default = None, схема, к которой принадлежат выводимые данные. Например для табличных данных, это может быть имя схемы базы данных.

Разбор сложных структур реализуется в рамках конкретных задач индивидуально.

[Пример DAG'a с использование сущности типа `XComEntity`](operator/ws/dags/df/tests/xcom/extract_onec_web_xcom.py)

4.5.2 Поддерживаемые переменные

- `AF_TRANSPOSE`: default - False, транспонировать таблицу-источник перед экстракцией, что может потребоваться при загрузке данных по столбцам (группам столбцов), а не по строкам.

4.5.3 Использование пользовательских обработчиков данных (переменная `AF_ARRAY_HANDLER`)

При экстракте сущности `ContentFileEntity`, `JSONContentFileEntity`, `OneCWebEntity` поддерживают переменные для задания пользовательской функции для обработки данных источника (содержимого файла или ответа API):

- `AF_ARRAY_HANDLER`, default = None, путь (указатель) на функцию, в которую будут переданы данные источника, возвращенный результат будет отправлен получателю. Функции на вход передаются следующие параметры:

- `data` - данные источника (тип зависит от типа источника);

- `op_ins_id` - идентификатор записи из таблицы `df.op_ins` с информацией о выполняемой задаче процесса;

- `expiration` - время, до которого данные считаются актуальными (используется в зависимости от типа источника, по умолчанию не используется);

- `meta` - информация о данных источника в виде json (зависит от типа источника).

Результат функции должен представлять собой список списков, первый элемент которого содержит наименования полей для получателя.

- `AF_ARRAY_KWARGS`, default = None, список именованных аргументов, значения которых дополнительно передаются в `AF_ARRAY_HANDLER`.

Результат работы обработчика будет передан в экстрактор, для дальнейшей загрузки.

Дополнительно к пользовательским обработчикам существуют системные.

4.5.3.1 Системные обработчики данных

Системные обработчики доступны в модуле `source.df.common.helpers.adapter`:

- `source.df.common.helpers.adapter.content_to_array` - используется для загрузки данных источника целиком в таблицу.

Поддерживаются следующие параметры:

- `path` - default = None, строка, представляющая путь к значению в данных источника, если необходимо выбрать только часть данных;

- ожидается JSON pointer, например, `/flow/steps/2/title`, где строки - ключи, а числа - индексы

- `serialize_value` - default = False, признак необходимости сериализации данных источника с помощью `json.dumps`;

- `expand_value` - default = False, признак необходимости разворачивания элементов списка данных источника и добавления каждого элемента отдельно в результирующий список;

При использовании данного обработчика в получателе будут доступны следующие поля:

- `op_ins_id` - идентификатор записи из таблицы `df.op_ins` с информацией о выполняемой задаче процесса;

- `expiration` - время, до которого данные считаются актуальными;

- `meta` - информация о данных источника в виде json (зависит от типа источника).

- `value` - данные источника (тип зависит от типа источника);

- `source.df.common.helpers.adapter.dict_list_to_array` - разворачивает список словарей в список списков.

При использовании данного обработчика в получателе будут доступны поля, имена которых совпадают с ключами словарей исходного списка словарей.

4.5.4 Extract из Excel

Поддерживаются 3 диалекта СУБД: mssql, postgresql, частично qhb (без функционала переноса данных между базами) и частично sqlite (для быстрого тестирования).

Переменные окружения будут использованы в файлах метаданных как переменные подстановки только если они имеют префикс `AF_`. Соответствие сущностей источника и получателя определяется по name. Сущности, не найденные и в источнике и получателе - игнорируются. Соответствие атрибутов сущностей источника и получателя определяется по name.

В модели Producer'a может быть указан порядковый номер/буквенное обозначение колонки и/или ее наименование для сопоставления с именем атрибута модели Consumer.

Обязательные свойства атрибутов сущности типа `ExcelFileEntity`: `name`.

Опциональные свойства атрибутов сущности типа `ExcelFileEntity`: `ordinal/columnLetter`, `columnName`. У каждого атрибута сущности типа `ExcelFileEntity` должен присутствовать `ordinal/columnLetter` либо `columnName`. Если у сущности задана опция `columnNamesRow`, но у атрибута не задан `columnName`, значение `columnName` берется из `name`.

Значения опций в аннотациях могут быть строковыми, числовыми, а также `null`, `false`, `true`. Распознаваемые специальные значения опций, задаются в кавычках: `"null"`, `"false"`, `"true"`.
Одномоментно один экстрактор обрабатывает один файл.
В описаниях аннотаций термины "маска" и "регулярное выражение" взаимозаменяемы.
Объекты `Connection` для подключения к СУБД `QHB` должен иметь тип `Postgres` и `Extra` опцию `dialect = qhb`, если диалект не задан через переменную окружения `AF_DWH_DB_DIALECT`.

4.5.4.1 Поддерживаемые аннотации сущности типа ExcelFileEntity:

- `rowsNumberToSkip`, default = 0, пропускать первые N строк, нумерация с 1

- `rowsLoadLimit`, default = 0, обработать не более N строк начиная с 1, 0 - обработать все

- `rowsBatchSize`, default = 10000, количество строк (записей) данных, обрабатываемых как один целостный набор

- `batchSizeBreadth`, default = 15, максимальное количество столбцов в пачке, если оно меньше числа обрабатываемых столбцов, то пропорционально уменьшается `rowsBatchSize`

- `skipBlankRows`, default = True, пропускать пустые строки

- `rowMaskToSkip`, маска пропуска строк

- `rowMaskToSkipL` - `rowMaskToSkip` по склеенным значениям ячеек

- `headLastRowMask`, маска последней строки шапки для пропуска, не может идти позже строки с наименованиями `columnNamesRow`

- `skipHeadLastRow`, default = True, пропускать последнюю строку шапки

- `headLastRowMaskStrict`, default = False, искать маску последней строки шапки только в обрабатываемых столбцах

- `headLastRowMaskL` - `headLastRowMask` по склеенным значениям ячеек

- `minHeadRow`, default = 1, определяет номер первой строки шапки

- `minHeadRowMask`, default = None, маска первой строки шапки, переопределяет `minHeadRow`, если срабатывает

- `maxHeadRowOffset`, default = 20, определяет максимально возможное количество строк в шапке

- `minHeadRowMaskL` - `minHeadRowMask` по склеенным значениям ячеек

- `tailFirstRowMask`, маска первой строки подвала для пропуска

- `skipTailFirstRow`, default = True, пропускать первую строку подвала

- `tailFirstRowMaskL` - `tailFirstRowMask` по склеенным значениям ячеек

- `detectMergedCells`, default = False, определять объединенные ячейки и заполнять их, включение данной настройки замедляет обработку

- `enforceStrictMapping`, default = True, требовать, чтобы в источнике были определены все атрибуты получателя

- `lowMemoryMode`, default = False, не загружать файл целиком в память (время загрузки значительно возрастет, использовать только для очень больших файлов), поддерживается только для `xlsx`

- `ignoreAbsentFiles`, default = False, игнорировать отсутствие файла для загрузки, завершением работы без ошибки

- `sheetOrdinal`, default = 1, номер загружаемого листа Excel, нумерация с 1, параметр игнорируется, если задана опция `sheetName`

- `sheetName`, default = None, имя загружаемого листа Excel

- `reInSheetNames`, default = False, идентифицировать листы Excel по регулярному выражению, если sheetName не задан - игнорируется

- `ignoreAbsentSheets`, default = False, пропускать файл и продолжать работу, даже если искомый лист не найден

- `columnNamesRow`, default = 0, номер строки с наименованиями, идентифицирующими атрибут получателя, нумерация с 1, должна идти позже шапки `headLastRowMask` (при значении, отличным от 0, дополнительно требуется указать аннотацию `rowsNumberToSkip` с тем же значением)

- `columnNamesRowMask`, default = None, маска для поиска строки с наименованиями, идентифицирующими атрибут получателя

- `columnNamesRowMaskL` - `columnNamesRowMask` по склеенным значениям ячеек

- `reInColumnNames`, default = False, идентифицировать колонки по регулярному выражению

- `dateBeginVariable`, default = None, переменная (Variable), содержащая дату начала периода загрузки в формате `YYYYMMDD`, загрузка выполняется в строку

- `dateEndVariable`, default = None, переменная (Variable), содержащая дату окончания периода загрузки в формате `YYYYMMDD`, загрузка выполняется в строку

- `unitIdVariable`, default = None, переменная (Variable), содержащая идентификатор предприятия, по которому загружаются данные

- `reInPathDelimiter`, default = None, составляющие полного пути к файлу, обернутые в разделитель, если он задан, обрабатываются как регулярное выражение

- `dateBeginGroup`, default = None, номер регулярного выражения и номер группы в нем в формате 'нрв:нг', как дата начала периода загрузки в пути к файлу, если `reInPathDelimiter` is not None и не определена `dateBeginVariable`, формат даты в пути - `YYYYMMDD`

- `dateEndGroup`, default = None, номер регулярного выражения и номер группы в нем в формате 'нрв:нг', как дата окончания периода загрузки в пути к файлу, если `reInPathDelimiter` is not None и не определена `dateEndVariable`, формат даты в пути - `YYYYMMDD`

- `unitIdGroup`, default = None, номер регулярного выражения и номер группы в нем в формате 'нрв:нг', как идентификатор предприятия в пути к файлу, если `reInPathDelimiter` is not None и не определена `unitIdVariable`

- `archiveFolder`, default = None, директория для переноса обработанных файлов с сохранением структуры с первой переменной папки, если такая есть

- `failedFolder`, default = None, директория для переноса обработанных с ошибками файлов, структура каталогов с файлами сохраняется, если путь к файлам содержит переменные; для переноса файлов аннотация `ignoreBogusFiles` должна иметь значение True, а `doNotPipeSkipped` - False

- `ignoreArchiveErrors`, default = False, игнорировать ошибки при архивации

- `removeArchivedFolders`, default = False, удалять пустые переменные папки, обрабатывается только если задана `archiveFolder`

- `doNotPipeSkipped`, default = False, не выполнять постобработку файлов, пропущенных по каким-либо причинам

- `skipHiddenSheets`, default = True, не обрабатывать скрытые листы, отключение данной настройки замедляет обработку, поддерживается только `xlsx` и `xls`

- `skipHiddenParts`, default = True, не обрабатывать скрытые строки и столбцы, поддерживается только `xlsx` и `xls`

- `skipHiddenRows`, default = False, не обрабатывать скрытые строки (учитывается только в режиме расширенной загрузки), поддерживается только `xlsx`

- `skipHiddenColumns`, default = False, не обрабатывать скрытые столбцы (учитывается только в режиме расширенной загрузки), поддерживается только `xlsx`

- `loadExtra`, default = False, расширенная загрузка, включение данной настройки позволяет загружать форматирование ячеек и переключает загрузчик на обработку только `xlsx` (влияет на загружаемые значения, например, значение объединенных ячеек не дублируется)

- `readonlyExtra`, default = True, при расширенной загрузке открывать файл только для чтения (влияет на загружаемые свойства, например, отсутствует col_idx, форматирование объединенных ячеек не дублируется, а их индивидуальные значения считываются)

- `maxBlanksToProcess`, default = None, обрабатывать максимум пустых строк подряд, для случаев, когда библиотека не может определить конец файла, 0 - завершать обработку на первой пустой строке

- `ignoreBogusFiles`, default = False, пропускать файлы, содержащие структурные ошибки

- `multiAccessMode`, default = False, позволяет одновременный доступ на запись к файлу нескольких процессов

- `skipMappingErrors`, default = False, позволяет пропускать файлы с ненайденными столбцами без падения

- `loadMultipleColumns`, default = False, позволяет загружать несколько столбцов в один атрибут

- `loadMultipleSheets`, default = False, позволяет загружать все листы подходящие под `sheetName` при `reInSheetNames` = True в одну сущность

- `convertToXLSX`, default = False, конвертирует файл в xlsx перед экстрактом, с целью повышения совместимости с расширенными функциями обработки данных, а также типов извлекаемых данных, поддерживаемые форматы: 'xls', 'xlsb', 'ods'

- `loadBreaker`, default = None, путь (указатель) на функцию, в которую будут передана загружаемая строка. В зависимости от результата, загрузка может быть прервана

- `loadBreakerKwargs`, default = None, список именованных аргументов, значения которых передаются в `loadBreaker`

- `loadBreakerCounter`, default = 1, количество срабатываний функции прерывания загрузки `loadBreaker`, после которого загрузка прерывается.

4.5.4.2 Поддерживаемые аннотации атрибутов сущности типа ExcelFileEntity

- `extraToLoad`, default = None, перечень дополнительных свойств ячеек, подлежащих загрузке, значения загружаются как строки, формат перечня: `свойство.один:свойство.два:...:свойство.эн`, включается при `loadExtra` = True

- `isRow`, default = False, является ли источником значений атрибута строка, строки не могут располагаться ниже строки, заданной `columnNamesRow` или `columnNamesRowMask`

- `rowColumn`, default = 0, ограничивает чтение строки заданным номером колонки, используется для загрузки значения определенной ячейки из строки при `isRow` = true

- `rowColumnMask`, default = None, определяет номер колонки `rowColumn` по маске, имеет приоритет над `rowColumn`

- `rowChainMask`, default = None, определяет номер колонки `rowColumn` по маске, используя значения всех ячеек начиная со строки по `firstRowMask`, имеет приоритет над `rowColumnMask`

- `rowColumnGroupMask`, default = None, определяет номера колонок (группу) `rowColumn` по маске, используется для загрузки значений определенных ячеек из строки при `isRow` = true, несколько колонок разных атрибутов могут образовывать комбинации, если в любой из групп более одной колонки

- `firstRowMask`, default = None, регулярное выражение, определяющее строку, с которой начинается отсчет строк, найденная строка имеет ordinal = 1

- `targetAttribute`, default = None, имя атрибута при `isRow` = true и `rowColumn` = 0, для которого определяется значение

- `const`, default = None, загрузка константны в атрибут при `isRow` = false, поддерживает специальные значения

- `valueMask`, default = None, регулярное выражение, определяющее значение из `rowColumn` или `const` записывается значение группы 0 или 1, по умолчанию используется `^(.*)$`, если значение не соответствует шаблону, заданному регулярным выражением, то возвращается значение целиком

- `setColumnOrdinal`, default = False, переопределяет номер колонки связанного атрибута `targetAttribute` на колонку, определенную в `rowColumn`, `rowColumnMask` или `rowChainMask`

- `doNotLoad`, default = False, при `isRow` = true позволяет не загружать атрибут в таблицу, метаданные получателя при этом не проверяются

- `isPivot`, default = False, определяет номер колонки с данными для кросс-таблицы

- `maxBlanksToProcess`, default = None, обрабатывать максимум пустых столбцов подряд, для случаев, когда библиотека не может определить конец строки, 0 - завершать обработку на первой

- `targetEntity`, default = None, наименование сущности, в атрибут которой должно сохраниться значение

- `handler`, default = None, путь (указатель) на функцию, в которую будут передано значение атрибута-константы, возвращенный результат будет отправлен получателю

- `kwargs`, default = None, список именованных аргументов, значения которых передаются в `handler`

4.5.4.3 Специальные значения аннотации const атрибутов сущности типа ExcelFileEntity:

- `sheetName` - значение аннотации `const` приравнивается к наименованию загружаемого листа, если загрузка выполняется по наименованию листа, иначе - к номеру листа
- `fileName` - имя файла
- `filePath` - полный путь файла

Если `ordinal` для атрибута типа `const = 0`, то при отсутствии какого-либо значения для этого атрибута, загрузка завершится с ошибкой

Если `ordinal` для атрибута типа `const != 1`, то при отсутствии какого-либо значения для этого атрибута будет загружено значение из соответствующего атрибута, если он существует, иначе загрузка завершится с ошибкой

4.5.4.4 Примеры дополнительных свойств ячеек для аннотации extraToLoad:

- `row` - номер строки, например, '1'
- `column` - номер колонки, например, '1'
- `coordinate` - координаты строки, например, 'E7'
- `font.name` - имя шрифта, например, 'Arial'
- `font.size` - размер шрифта, например, '11.0'
- `font.b`, `font.bold` - признак жирного шрифта ('True'|'False')

- `font.i`, `font.italic` - признак курсива ('True'|'False')
- `font.underline` - признак подчеркнутого шрифта ('True'|'False')

- `font.strike` - признак зачеркнутого шрифта ('True'|'False')
- `font.color.rgb` - цвет шрифта, например, 'FF0070C0'

- `font.color.theme` - стиль, например, '4'
- `alignment.indent` - размер отступа, например, '1.0'
- `alignment.horizontal` - горизонтальное выравнивание, например, 'right
- `alignment.vertical` - вертикальное выравнивание, например, 'center
- `alignment.wrap_text` - перенос текста ('True'|'False')
- `fill.bgColor.rgb` - цвет фона, например, 'FFCCFFCC'
- `fill.patternType` - тип узора заливки, например, 'solid'

- `fill.start_color.rgb` - начальный цвет заливки, например, 'FFFFFF00'
- `fill.end_color.rgb` - конечный цвет заливки, например, '00000000'
- `border.left.style` - граница слева, например, 'thin'
- `border.right.style` - граница справа, например, 'thin'
- `border.top.style` - граница сверху, например, 'thin'
- `border.bottom.style` - граница снизу, например, 'thin'

4.5.4.5 Специальные дополнительные свойства аннотации extraToLoad:

- `range` - диапазон объединения ячейки (при наличии), пример - `'A1:C3'`
- `sheet.hidden` - признак скрытия листа (`'True'|'False'`)
- `row.hidden` - признак скрытия строки (`'True'|'False'`)
- `column.hidden` - признак скрытия колонки (`'True'|'False'`)
- `row.outline` - уровень группировки строки например, `'0'`
- `column.outline` - уровень группировки строки например, `'0'`

Пример значения аннотации `extraToLoad: "font.b:font.i:alignment.indent:fill.bgColor.rgb:range"`

Значения дополнительных свойств аннотации `extraToLoad` хранятся в автоматически создаваемой таблице с наименованием `<имяТаблицыСДанными>_f`.

Структура таблицы: `id [int identity not null], rowId [int fk not null], column [varchar(128) not null], key [varchar(128) not null], value [varchar(256) not null]`

Для включения загрузки дополнительных свойств необходимо выполнить следующие шаги:

- в аннотации источника включить loadExtra
- в аннотации атрибутов источника задать значение extraToLoad
- в таблицу данных добавить атрибут типа uuid [not null, unique], в метаданных получателя задать этому атрибуту роль rid
  Поле типа uuid в таблице данных необходимо для привязки дополнительных свойств к строкам данных до непосредственной вставки строк данных в таблицу, т.к. id на этот момент еще неизвестен.
  В результате загрузки, помимо таблицы данных будет заполнена таблица с дополнительными свойствами <имя_таблицы_данных>_f, с привязкой по rowid к строкам

таблицы данных и свойствами в виде пар ключ-значение, в разрезе полей.

Пример скрипта добавления поля типа uuid в таблицу данных (postgres):

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

alter table "public"."test_formatting" add "gid2" uuid not null default uuid_generate_v1();

alter table "public"."test_formatting" add CONSTRAINT "test_formatting_gid2_key" UNIQUE ("gid2");

alter table "public"."test_formatting" drop CONSTRAINT "test_formatting_gid2_key";

alter table "public"."test_formatting" drop column "gid2";
```

4.5.4.6 Использование пользовательских обработчиков данных (аннотация `handler`)

Полученные данные (значение атрибута) могут быть обработаны системными или пользовательскими функциями (в общем случае - `callable`).

Результат обработки будет передан в получатель.

Таким образом, например, можно загрузить метаданные файла-источника.

Системные функции, предназначенные для обработки данных:

- `get_file_meta`
  - поддерживает именованный аргумент `attribute`, который принимает такие ключи как, например:
    - `st_size` - размер файла
    - `st_mtime` - дата и время модификации файла
  - возвращает соответствующие атрибуты `path.stat`
- `ts_to_dt`
  - поддерживает именованный аргумент `format`
  - возвращает дату и время, соответствующую переданному Unix time либо текущие

Обработчик можно использовать только с атрибутами-константами (`const`)

4.5.4.7 Использование пользовательских обработчиков прерывания загрузки (аннотация `loadBreaker`)

Полученные данные могут быть обработаны системными или пользовательскими функциями (в общем случае - `callable`). По умолчанию в функции доступны следующие переменные:

- `values_dict` - значения текущей строки в виде словаря

- `i` - номер текущей строки

- `all_values` - все значения текущей строки в виде списка.

Дополнительно к указанным переменным, в функции доступен список пользовательских именованных переменных, передаваемых через `loadBreakerKwargs`.

Результат функции будет передан в экстрактор и в зависимости от него загрузка может быть прервана или продолжена в обычном режиме:

- `True` и количество срабатываний функции увеличивается на 1, если общее количество срабатываний превышает `loadBreakerCounter`, загрузка прерывается

- `False` - текущая строка загружается

Таким образом, например, можно прервать загрузку при появлении в обязательных полях пустых значений.

Системные функции, предназначенные для обработки данных:

- `attributes_empty` (путь ``'source.df.common.helpers.general.attributes_not_empty'`)

- поддерживает именованный аргумент `attributes` - список атрибутов, значение которых необходимо проверить на пустую строку и None, например: ``'salary, add_salary, taxes, total, count'`

- возвращает `True`, если все переданные атрибуты содержат пустую строку или None

4.5.4.8 Дополнительные разъяснения по загрузке

Сущность обрабатывается без ошибки загрузки, даже если набор атрибутов источника и получателя отличается, если в метаданных и источника и получателя опция `enforceStrictMapping` явно отключена.

При определении номера и группы регулярного выражения в формате `'нрв:нг'`, регулярные выражения нумеруются с 1, а группы с 0, где группа 0 - значение выражения целиком.

Если при загрузке сложной шапки возникает ошибка `Ошибка экстрактора Excel: Столбца для <атрибут> нет в <файл>, метаданные не соответствуют файлу?`, возможно, что загрузчик не может определить границу шапки и данных, в этом случае следует задать `columnNamesRow` или ее аналог более точно, чтобы опция срабатывала только точно на нужной строке.

Если при загрузке возникает ошибка `Не задан номер или буквенное обозначение колонки для атрибута <имя>`, следует убедиться, что у соответствующего атрибута указан `ordinal/columnLetter`.

Если при загрузке возникает ошибка `Коллизия колонки <имя или маска> (номер) и атрибута <имя>.`, вероятно произошло совпадение наименований колонок данных, либо маска наименования подошла более чем одному значению.

Экстракт из `xls`, `xlsb`, `ods`, `xlsm` является эксперементальным и обладает рядом ограничений, так, в частности:

- ни для одного из указанных типов файлов не поддерживаются параллельный экстракт и экстракт форматирования, для типов `xlsb`, `ods` не поддерживаются идентификация объединенных ячеек, для типов `xlsm`, `xlsb` не поддерживается экстракт из скрытых ячеек, а для `ods` экстракт из скрытых ячеек выполняется всегда
- при экстракции из `xls`, `xlsm`, `xlsb`, `ods` файлов и включенной аннотации `convertToXLSX`, либо при включенной параллельной

экстракции (`multiAccessMode = True`), в контейнере оператора должно быть достаточно сводного места (hd) для полной копии обрабатываемого файла

- типы данных в результате экстракции из `xls`, `xlsm`, `xlsb`, `ods` могут различаться при различных состояниях аннотации `convertToXLSX`
- типы данных при экстракте из `xlsb` без конвертации идентифицируются только как строковые и числовые
- при экстракции из `xlsb`, `ods` пустые строки до начала данных игнорируются

4.5.5 Extract из XML API NetDB

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул, буквенных идентификаторов и листов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `NetDBFileEntity`

4.5.5.1 Уникальные аннотации сущности типа NetDBFileEntity:

- `spreadMergedRows`, `default = False`, дублировать значения в объединенных строках `rowspan`
- `spreadMergedColumns`, `default = False`, дублировать значения в объединенных колонках colspan

4.5.5.2 Уникальные аннотации атрибутов сущности типа NetDBFileEntity:

- `param`, `default = None`, загрузка параметров из элемента `<params />` в атрибут при `isRow = false`, формат переменной параметров - `@ndbparam<индекс_параметра_с_1>`

4.5.6 Extract из CSV

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул и буквенных идентификаторов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `CSVFileEntity`

Мультилистовые CSV файлы в текущей версии не поддерживаются

4.5.6.1 Уникальные аннотации сущности типа CSVFileEntity:

- `encoding`, `default = utf-8`, кодировка файла
- `delimiter`, `default = ,`, разделитель атрибутов
- `detectFloat`, `default = False`, пытаться распознать дробные числа, слегка замедляет обработку
- `detectInt`, `default = False`, пытаться распознать целые числа, слегка замедляет обработку, действует только при включенной detectFloat
- `detectDatetime`, `default = False`, пытаться распознать даты, слегка замедляет обработку

4.5.7 Extract из HTML

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул и буквенных идентификаторов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `HTMLFileEntity`

4.5.8 Extract содержимого текстового файла

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул и буквенных идентификаторов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `ContentFileEntity`. Доступны следующие поля:

- `op_ins_id` - идентификатор записи из таблицы `df.op_ins` с информацией о выполняемой задаче процесса;

- `expiration` - время, до которого данные считаются актуальными (в данном адаптере не используется);

- `meta` - расширенный объект типа os.path в виде json, содержащего информацию о файле: имя файла, путь до файла внутри контейнера, размер, дата изменения и т.п. (например, `{"st_atime": 1671805678.032348, "st_atime_ns": 1671805678032348000, "st_blksize": 4096, "st_blocks": 8, "st_ctime": 1671549361.061063, "st_ctime_ns": 1671549361061063000, "st_dev": 2096, "st_gid": 1000, "st_ino": 199910, "st_mode": 33204, "st_mtime": 1671549361.061063, "st_mtime_ns": 1671549361061063000, "st_nlink": 1, "st_rdev": 0, "st_size": 3, "st_uid": 1000, "path": "/app/ws/share/df/examples/content/text_a.txt", "name": "text_a.txt"}`);

- `value` - содержимое файла в виде текста

4.5.8.1.1 Дополнительные возможности сущности типа `ContentFileEntity`:

Адаптер предоставляет следующий функционал:

- загрузка содержимого файла в таблицу,

- обработка содержимого загружаемых файлов пользовательским скриптом.

Поддерживаемые переменные окружения таска:

- `AF_PRODUCER` - путь к описанию источника/ов данных, например, файла, относительно папки метаданных;

- `AF_CONSUMER` - путь к описанию получателя/ей данных, например, таблицы, относительно папки метаданных;

- `AF_SCRIPT_PATH` - опциональный параметр, задающий путь к SQL-скрипту, описывающему обработку содержимого загружаемых файлов. Например, выборка данных из xml. Содержимое файлов загружается в системную таблицу `df.op_data`, с которой может работать заданный запрос. Список полей, возвращаемых запросом, определяет список возможных

атрибутов получателя. В скрипте, дополнительно к переменным окружения, доступны следующие переменные подстановки:

- `AF_OP_ID` - идентификатор записи из таблицы `df.op` с информацией о выполняемом процессе (требует наличия `AF_DAG_ID`);

- `AF_OP_INS_ID` - идентификатор записи из таблицы `df.op_ins` с информацией о выполняемой задаче процесса;

- `AF_OP_DATA_ID` - идентификатор записи `df.op_data.id`, загруженной текущим таском;

- `AF_PURGE_LOADED_DATA` - default - True, указывает необходимость удаления записи, загруженной текущим таском, из таблицы `df.op_data` после завершения обработки;

- `AF_DWH_DB_CONNECTION` - подключение к базе данных, в которую необходимо загрузить данные

Очистка содержимого таблицы `df.op_data` происходит:

- отдельным шагом системного процесса `df_airflow_log_cleanup`

- системным процессом `df_op_data_cleanup` (дополнительно запускается команда `VACUUM {{AF_DWH_DB_SCHEMA}}.op_data`).

Из таблицы удаляется:

- содержимое, которое было загружено ранее, чем N дней (задается переменной `df_airflow_log_cleanup_params.tables.op_data.retention_days`, по умолчанию 15 дней),

- или данные которые стали неактуальными (значение поля `expiration` стало меньше `CURRENT_TIMESTAMP`)

Пример загрузки обработанного xml приведен в процессе [`df_example_extract_processed_content_xml`](operator/ws/dags/df/examples/content/extract_xml.py)

4.5.8.2 Extract содержимого JSON-файла

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул и буквенных идентификаторов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `JSONContentFileEntity`.

Идентификация атрибутов выполняется по наименованиям (`columnName`), либо в порядке их идентификаторов (`ordinal`)

4.5.8.2.1 Уникальные аннотации сущности типа `JSONContentFileEntity`:

- `resource` - путь до элемента с данными внутри файла, которые нужно загрузить. Каждый уровень вложенности разделяется точками. Например: `Elements` или `Elements.item.Attributes`;

4.5.8.2.2 Дополнительные возможности сущности типа `JSONContentFileEntity`:

Адаптер предоставляет следующий функционал:

- загрузка части файла в таблицу,

- загрузка содержимого файла в таблицу,

- обработка содержимого загружаемых файлов пользовательским обработчиком, заданным через переменные `AF_ARRAY_HANDLER` и `AF_ARRAY_KWARGS`.

Если обработчик не передан, то для преобразования данных источника к табличному виду будет использован системный обработчик `source.df.common.helpers.adapter.dict_list_to_array`. у

4.5.9 Extract из PIPware API

Используется Excel экстрактор без возможности разбора шапки таблицы, ввиду отсутствия ее поддержки форматом.

Для описания источника используется тип сущности `PIPwareEntity`

4.5.9.1 Уникальные аннотации сущности типа PIPwareEntity:

`rowsNumberToSkip`, `default = 1` (наименования колонок), пропускать первые N строк, нумерация с 1.

4.5.10 Extract из KZStat API

Используется Excel экстрактор,в части не касающейся обработки форматирования, формул и буквенных идентификаторов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `KZStatWebEntity`

4.5.11 Extract из KZTask API (API Поручений)

Используется Excel экстрактор,в части не касающейся обработки форматирования, формул и буквенных идентификаторов, ввиду отсутствия поддержки форматом таковых. Для описания источника используется тип сущности `KZTaskWebEntity`. Идентификация атрибутов выполняется по наименованиям (`columnName`). Адаптер поддерживает следующие `extra` опции объекта `Connection`:

- `auth_endpoint` - адрес авторизации относительно API entry point (`Connection.host`). По умолчанию задан `login`. Обязательно наличие логина и пароля в объекте `Connection` для авторизации.

4.5.12 Extract из Visiology DC API

Используется Excel экстрактор в качестве обработчика измерения как тела таблицы.

Для описания источника используется тип сущности `DCDimensionEntity`

Идентификация атрибутов выполняется по наименованиям (`columnName`), либо в порядке их идентификаторов (`ordinal`).

Порядок следования колонок строк: `id`, `name`, `path`, `attr1 value, .., attrN value` Количество записей, получаемое через API в одном запросе

можно ограничить как значением ключа `'batch'` в `'extra'` подключения, так и аннотацией `dcBatchSize` метаданных источника

4.5.12.1 Уникальные аннотации сущности типа DCDimensionEntity:

- `dcBatchSize`, `default = 1000`, максимальное количество элементов измерения в одном ответе API
- `pathDelimiter`, `default = /`, задает разделитель элементов `path` атрибутов
- `rowsNumberToSkip`, `default = 1` (наименования колонок), пропускать первые N строк, нумерация с 1.

4.5.13 Extract из БД Postgres, MS SQL Server, Firebird, Oracle (совместимость драйвера - 12.1)

Выполняется отдельным раннером `/app/source/python/run_sql_move.py`.

Для получения загружаемого набора данных из таблицы в базе источнике должен быть определен SQL скрипт по пути `AF_PRODUCER_SCRIPT_PATH`.

Для записи данных в таблицу базы получателя должен быть определен SQL скрипт по пути `AF_CONSUMER_SCRIPT_PATH`.

Дополнительно использует переменные окружения `AF_DWH_DB_*_PRODUCER` и `AF_DWH_DB_*_CONSUMER` для подключения к источнику и получателю, соответственно.

Количество записей, отправляемых в таблицу-получатель за раз, ограничивается переменной окружения `AF_BATCH_SIZE`, по умолчанию - 1000, актуально только для получателя Postgres.

Для обработки больших объемов данных, например - превышающих доступную контейнеру оперативную память, следует ограничивать предельное количество записей, передаваемых единовременно, через переменную `AF_PRODUCER_BATCH_SIZE`.

`Schema` объекта Connection типа Oracle определеяет Oracle Instance.

Объект Connection для подключения к СУБД Firebird должен иметь тип `File (path)` и `Extra` опцию `dialect = firebird`. Connection `Schema` определеяет `Database path`.

4.5.13.1 Поддержка ODBC

Polyflow обеспечивает подключение к различным СУБД через ODBC-драйверы на уровне программного кода. Поддержка реализована с использованием популярных Python-библиотек, таких как pyodbc.

Ключевые особенности:

- Универсальный доступ: ODBC используется по умолчанию для подключения к таким СУБД, как Microsoft SQL Server и Oracle, что обеспечивает совместимость со множеством версий и конфигураций баз данных.
- Интеграция с диалектами: в рамках единого интерфейса connection_scope ODBC-подключения органично интегрированы в систему поддержки диалектов СУБД (MSSQL, Oracle, PostgreSQL, Firebird).
- Гибкая настройка: для MSSQL формируется стандартизированная ODBC-строка подключения с использованием актуального драйвера («ODBC Driver 17 for SQL Server»), с возможностью передачи дополнительных параметров (extra).

Пример реализации ODBC-подключения в коде Polyflow:

```
#python
# Пример для Microsoft SQL Server
if dialect == 'mssql':
    conn = pyodbc.connect(get_ms_conn_string(server, db, user, password,
port=port, extra=extra), autocommit=False)


# Пример для Oracle
```

```
elif dialect == 'oracle':
    driver = get_oracle_driver() # Возвращает корректный ODBC-драйвер
    conn = pyodbc.connect(**{'DBQ': server, 'uid': user, 'pwd': password,
'driver': driver})
```

4.5.13.2 Интеграция с Greenplum

В дополнение к стандартным базам данных Polyflow поддерживает подключение к кластерным СУБД, построенным на основе PostgreSQL, в частности — к Greenplum Database.

Обоснование совместимости: Greenplum представляет собой MPP-архитектуру (Massively Parallel Processing), построенную на ядре PostgreSQL. Благодаря этому, для подключения к Greenplum можно использовать тот же тип соединения (Conn Type), что и для PostgreSQL, и тот же драйвер.

Рекомендации по настройке подключения:

- Conn Id: Произвольное уникальное имя (например, greenplum_dwh)
- Conn Type: PostgreSQL
- Host: Адрес координатора (мастер-узла) кластера Greenplum
- Login / Password: Учетные данные пользователя
- Schema: Схема данных (при необходимости)
- Port: Порт для подключения (по умолчанию для Greenplum — 5432, как и у PostgreSQL)

Особенности работы: Все SQL-скрипты и операции, выполняемые через подключение к Greenplum, должны учитывать распределенную природу этой СУБД, включая правильное определение ключей распределения данных.

4.5.13.3 Интеграция с Postgres Pro

Polyflow обеспечивает полную поддержку подключения к Postgres Pro — российской промышленной СУБД, созданной на базе PostgreSQL.

Обоснование совместимости: Postgres Pro сохраняет полную бинарную и функциональную совместимость с PostgreSQL. Это означает использование того же SQL-синтаксиса, API и протоколов взаимодействия. Поэтому для подключения к Postgres Pro используется тот же тип соединения (Conn Type: PostgreSQL) и те же драйверы, что и для стандартного PostgreSQL.

Рекомендации по настройке подключения:

- Conn Id: Произвольное уникальное имя (например, postgrespro_warehouse)
- Conn Type: PostgreSQL
- Host: Адрес сервера СУБД Postgres Pro
- Login / Password: Учетные данные пользователя
- Schema: Схема данных (при необходимости)
- Port: Порт для подключения (по умолчанию — 5432)

Особенности и преимущества: Использование Postgres Pro в качестве целевой СУБД позволяет воспользоваться расширенными возможностями этой платформы, такими как механизм инкрементального резервного копирования PTRACK, улучшенная диагностика и оптимизации для высоконагруженных систем, что особенно актуально для корпоративных хранилищ данных (DWH).

4.5.13.4 Интеграция с экосистемой Arenadata

Polyflow обеспечивает совместимость и поддерживает интеграцию с решениями компании Arenadata — одного из ведущих российских разработчиков платформ для хранения и анализа больших данных.

Поддерживаемые продукты Arenadata:

- Arenadata DB (ADB): MPP-СУБД (Massively Parallel Processing), основанная на открытом коде Greenplum и PostgreSQL. Polyflow может подключаться к Arenadata DB для извлечения и загрузки данных, используя те же механизмы и коннекторы (Conn Type), что и для работы с PostgreSQL и Greenplum.
- Arenadata QuickMarts (ADQM): Высокопроизводительная колоночная СУБД, совместимая с ClickHouse. Интеграция реализуется через коннекторы, предназначенные для работы с ClickHouse.

Обоснование совместимости: Архитектура ключевых продуктов Arenadata (ADB и ADQM) построена на технологиях, которые изначально поддерживаются Polyflow. Arenadata DB сохраняет полную совместимость с PostgreSQL и Greenplum, а Arenadata QuickMarts — с ClickHouse. Благодаря этому, для подключения к ним используются стандартные, уже описанные в документации Polyflow типы соединений и драйверы.

Преимущества интеграции: Совместное использование Polyflow и платформенных решений Arenadata позволяет создавать мощные и отказоустойчивые ETL-конвейи в полностью отечественном технологическом стеке, что особенно актуально для задач построения корпоративных хранилищ данных (DWH) в условиях импортозамещения.

4.5.13.5 Интеграция с ClickHouse

Polyflow поддерживает подключение к ClickHouse через использование PostgreSQL-совместимого протокола, который поддерживается самой СУБД ClickHouse. Данный метод позволяет использовать стандартный коннектор Polyflow для PostgreSQL для выполнения операций с ClickHouse.

Принцип работы: ClickHouse может работать с клиентами, использующими PostgreSQL-протокол. Это позволяет настроить в Polyflow подключение к ClickHouse так, как если бы это был сервер PostgreSQL.

Рекомендации по настройке подключения:

- Conn Id: Произвольное уникальное имя (например, clickhouse_analytics)
- Conn Type: PostgreSQL
- Host: Адрес сервера ClickHouse
- Login / Password: Учетные данные пользователя ClickHouse
- Database: Имя базы данных в ClickHouse
- Port: Порт, на котором ClickHouse слушает входящие подключения по PostgreSQL-протоколу (должен быть задан в настройках ClickHouse, по умолчанию не активирован).

Важные замечания:

- Для работы данного метода необходимо, чтобы в конфигурации ClickHouse был явно задан параметр postgresql_port.
- Убедитесь, что используемый пользователь ClickHouse имеет пароль, так как подключение с пустым паролем через psql (и, следовательно, через данный протокол) невозможно.

4.5.13.6 Интеграция с MongoDB

Polyflow поддерживает интеграцию с документоориентированной NoSQL СУБД MongoDB через специализированные адаптеры и операторы.

Способы интеграции:

1) Через HTTP API адаптер

- Использование типа сущности JSONMapWebEntity или JSONListWebEntity

- Подключение через HTTP Connection с указанием endpoint MongoDB REST API
- Поддержка аутентификации Basic Auth

2) Прямое подключение к БД

- Использование стандартного MongoDB Connection
- Поддержка операций чтения/записи через Python-скрипты
- Возможность использования агрегационных pipeline

Пример настройки Connection:

```
#python
# Параметры подключения к MongoDB
Conn Id: mongodb_analytics
Conn Type: MongoDB
Host: mongodb://localhost:27017
Extra: {
    "database": "analytics",
    "collection": "events"
}
```

Пример использования в DAG:

```
#python
mongo_extract_task = DockerOperator(
    task_id='extract_from_mongodb',
    image='df_operator:latest',
    command='python /app/ws/source/df/python/run.py',
    environment_variables={
        'AF_SCRIPT_PATH': '/app/ws/source/df/python/scripts/
run_mongo_extract.pyj', # пользовательский скрипт реализации экстракта
        'AF_MONGO_CONNECTION': 'mongodb_analytics',
        'AF_QUERY': '{"timestamp": {"$gte": "2024-01-01"}}',
        'AF_OUTPUT_PATH': '/app/share/mongodb_output.json'
    }
)
```

Поддерживаемые операции:

- Извлечение данных по произвольным запросам

- Загрузка данных в коллекции MongoDB
- Агрегация данных с использованием MongoDB pipeline
- Инкрементальное обновление документов

Особенности работы:

- Поддержка работы с большими объемами документов через batch-обработку
- Возможность преобразования JSON-структур в реляционную модель
- Интеграция с системой контроля качества данных Polyflow

Требования:

- MongoDB версии 4.0 и выше
- Наличие соответствующих прав доступа у пользователя
- Установка необходимых Python-библиотек в контейнере оператора

4.5.14 Extract из БД Postgres, MS SQL Server в Visiology Smart Forms и ViQube по HTTP API

Выполняется отдельным раннером `/app/ws/source/df/python/run_sql_to_viapi.py`.

Поддерживаются создание, удаление, обновление для элементов измерений Smart Forms.

Поддерживаются создание, удаление для элементов таблиц ViQube.

Поддерживаются следующие переменные окружения:

- `AF_DWH_DB_CONNECTION_CONSUMER` код или сериализованное подключение к получателю

- `AF_BATCH_SIZE` - количество элементов, единовременно отправляемых на эндпойнт, по умолчанию - 1000

- `AF_SCRIPT_PATH` - путь к шаблону SQL селекта источника

- `AF_DELIMITER`- разделитель для элементов `path`, по умолчанию - `/` (только для Smart Forms)

- `AF_OPERATION` - тип операции

  - `create` - добавление записей (по умолчанию)

  - `delete` - удаление записей (по опциональному условию)

  - `update` - обновление записей (по условию, только для Smart Forms)

- `AF_ENDPOINT` - API endpoint (ресурс)

- `AF_FILTER_TYPE` - правило объединения условия отбора, по умолчанию `and`, поддерживаются: `and`, `or` (только для Smart Forms)

- `AF_CONTAINER` - целевая таблица (только для ViQube delete)

4.5.14.1 Структура SQL запросов транслируемых в запросы к измерениям Smart Forms:

```sql
select element_id, element_name, element/path, attr1_id, attr1_value, ..., attrN_id, attrN_value
```

`element_id`, `element_name`, `element/path` - обязательны, если `element/path` пуст, то следует передавать `''`

Структура запроса для удаления элементов измерения по `Id`:

```sql
select id as value, 'Id' as type, 'условие' as operator, '' as name
```

Структура запроса для обновления элементов измерения по `name`:

```sql
select 'наименование элемента' as value, 'name' as type, 'условие' as condition, '' as name, 'новое значение' as n_value, 'attribute' as n_type, 'наименование атрибута' as n_name
```

4.5.14.2 Структура SQL запросов транслируемых в запросы к таблицам ViQube:

```sql
```

```
select column1_value as column1_name, ..., columnN_value as columnN_name
```

Структура запроса для удаления записей таблицы:

```sql
select column_name as "column", 'условие' as "condition", 'somevalue' as
"value"
```

Поддерживаются следущие условия (объединяются через `and`):

- EQ - равно

- NE - не равно

- GT - больше

- LT - меньше

- GE - больше или равно

- LE - меньше или равнос

4.5.15 Extract из XML файлов Электронного бюджета

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул, буквенных идентификаторов и листов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `EBFileEntity`. Идентификация атрибутов выполняется по наименованиям (`columnName`), либо в порядке их идентификаторов (`ordinal`). Отчетный период данных содержится в имени XML файла, для извлечения его оттуда необходимо использовать аннотации `reInPathDelimiter` и одну из `dateBeginGroup` или `dateEndGroup`.

4.5.16 Extract из XML файлов Электронного учета МинСельХоза

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул, буквенных идентификаторов и листов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `MSHEUFileEntity`. Идентификация атрибутов выполняется по наименованиям (`columnName`).

4.5.17 Extract из Excel 2003 XML Spreadsheet

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул, буквенных идентификаторов и листов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `Excel2003XMLFileEntity`

4.5.18 Extract из API Электронного бюджета

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул, буквенных идентификаторов и листов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `EBWebEntity`. Идентификация атрибутов выполняется по наименованиям (`columnName`).

4.5.18.1 Уникальные аннотации сущности типа EBWebEntity:

- `nullMappingErrors`, `default = False`, позволяет в случае отсутствия в источнике колонки записывать `NULL` в таблицу.

4.5.19 Extract из API, возвращающего данные в виде списка словарей в формате JSON

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул, буквенных идентификаторов и листов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `JSONMapWebEntity`.

Идентификация атрибутов выполняется по наименованиям (`columnName`).

Адаптер поддерживает следующие `extra` опции объекта `Connection`:

- `auth - basic` или `ntlm` как метод аутентификации API
- `domain` - имя домена при `auth = ntlm`, если требуется

4.5.20 Extract из API, возвращающего данные в виде словаря массивов в формате JSON

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул, буквенных идентификаторов и листов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `JSONListWebEntity`.

Идентификация атрибутов выполняется по наименованиям (`columnName`).

Адаптер поддерживает следующие `extra` опции объекта `Connection`:

- `auth - basic` или `ntlm` как метод аутентификации API
- `domain` - имя домена при `auth = ntlm`, если требуется
- `columns_name` - параметр ответа API содержащий массив заголовков. По умолчанию задан `columns`.
- `data_name` - параметр ответа API содержащий список массивов данных соответствующих заголовкам указанным в заголовках. По умолчанию задан `data`.

4.5.21 Extract из сервиса [Nasdaq](http://www.nasdaqomx.com/commodities/market-prices/history), возвращающего данные в виде html таблицы

Для описания источника используется тип сущности `NasdaqHTMLWebEntity`.

Идентификация атрибутов выполняется по наименованиям (`columnName`).

Для указания даты необходимо прописать options `fromDate` и `toDate` в метаданных формата `yyyy-mm-dd`. Кроме того, можно указать значения опции `XPath`, позволяющего выбрать Market, Types, Product и другие возможные изменения в сервисе.

По умолчанию значение `//inst` - Маркет `Eletricity Nordic` и снята галка с `Traded`.

В качестве примера: `//inst[@tp='F'][ph/hi/@rv!='' or ph/hi/@tv!='']` - Маркет `Electricity UK`, Продукт `Futures`, и стоит галочка `Traded`.

4.5.22 Extract из 1C OData API

Используется Excel экстрактор, в части не касающейся обработки форматирования, формул и буквенных идентификаторов, ввиду отсутствия поддержки форматом таковых.

Для описания источника используется тип сущности `OneCWebEntity`.

4.5.22.1 Уникальные аннотации сущности типа `OneCWebEntity`:

- `resource` - наименование сущности или ресурса 1С, например: `Catalog_Сотрудники`;

4.5.22.2 Дополнительные опции сущности типа `OneCWebEntity`

Для указания дополнительных параметров API необходимо прописать options:

- `format`, default = 'json' - формат получаемых от ресурса данных, доступны значения `json`, `atom` для загрузки `json` и `xml` соответственно;

- `filter`, default = None - параметр, позволяющий фильтровать возвращаемые данные ресурса, например: `Должность_Key ne guid'00000000-0000-0000-0000-000000000000'`, подробнее можно найти [тут](https://infostart.ru/1c/articles/1570140/);

- `orderby`, default = None - параметр, позволяющий получить отсортированные данные, например: `Название asc, Производитель desc`;

- `top`, default = None - параметр, для ограничения количества записей, возвращаемых при обращении к ресурсу, например: `10`;

- `skip`, default = None - параметр, позволяющий исключить из результата запроса первые несколько записей, например: `2`. Если параметры $top и $skip указываются одновременно, то параметр $skip будет применен

раньше, чем параметр $top. Приоритет применения параметров не зависит от порядка их указания в метаданных;

- `count`, default = None - параметр позволяет получить в качестве результата запроса не выборку, а ее (выборки) размер, например: `true`. При успешном выполнении, тело ответа будет содержать только число элементов коллекции, отформатированное как обычное число;

- `expand`, default = None - параметр для указания связанных ресурсов, которые необходимо включить в число полученных. Например, запрос `Catalog_Сотрудники?$expand=ФизическоеЛицо` возвращает данные из ресурса `Сотрудники` и связанную с ней информацию из ресурса `ФизическоеЛицо`. Для задания такого запроса в параметр `expand` необходимо передать значение `ФизическоеЛицо`;

- `select`, default = None - описывает перечень свойств сущности, которые получаются при обращении к стандартному интерфейсу, например: `Period, ФизическоеЛицо_Key, ВидСтажа_Key, ДатаОтсчета`;

- `search`, default = None - параметр позволяющий ограничить результат, включив в него только те объекты, которые соответствуют указанному выражению поиска. Определение того, что означает соответствие, зависит от реализации. Запрос `$search=Геров` позволяет получить всех людей, в содержании которых есть слово `Геров`;

Работа опций зависит от реализации API конкретного стенда с 1С.

4.5.22.3 Возможности сущности типа `OneCWebEntity`:

Адаптер предоставляет функционал загрузки из 1С OData API:

- табличных данных (при использовании формата `json` (по умолчанию));

- ответа API обработанного пользовательской функцией;

- ответа API обработанного с помощью пользовательского SQL-скрипта.

4.5.22.4 Поддерживаемые переменные окружения таска:

- `AF_PRODUCER` - путь к описанию источника/ов данных, например, файла, относительно папки метаданных;

- `AF_CONSUMER` - путь к описанию получателя/ей данных, например, таблицы, относительно папки метаданных;

- Поддерживаются стандартные переменные `AF_ARRAY_HANDLER` и `AF_ARRAY_KWARGS` для задания пользовательской функции для обработки ответа API. Дополнение к стандартным параметрам и возвращаемому результат для данного типа адаптера:

- `data` - содержимое ответа API (для формата `atom` ответ в виде текста, для `json` - ответ в виде словаря);

- `meta` - информация о ресурсе в виде json, содержащего следующую информацию: `path` - путь ресурса, `params` - параметры вызова, `format` - формат ответа `json` или `atom`.

При загрузке данных в формате `atom`, указание обработчика обязательно.

Если формат не задан, то по умолчанию будет использоваться json.

Если обработчик не передан, то для преобразования данных источника к табличному виду по умолчанию будет использован системный обработчик `source.df.common.helpers.adapter.dict_list_to_array`. Для получения ответа API в виде строки или числа (например, `count=true`), можно воспользоваться системным обработчиком `source.df.common.helpers.adapter.content_to_array`.

- `AF_SCRIPT_PATH` - опциональный параметр, задающий путь к SQL-скрипту, описывающему обработку ответа OData API 1С. Например, выборка данных из xml. Ответ API загружается в системную таблицу `df.op_data`, с которой может работать заданный запрос. Список полей,

возвращаемых запросом, определяет список возможных атрибутов получателя. В скрипте, дополнительно к переменным окружения, доступны следующие переменные подстановки:

- `AF_OP_ID` - идентификатор записи из таблицы `df.op` с информацией о выполняемом процессе (требует наличия `AF_DAG_ID`);

- `AF_OP_INS_ID` - идентификатор записи из таблицы `df.op_ins` с информацией о выполняемой задаче процесса;

- `AF_OP_DATA_ID` - идентификатор записи `df.op_data.id`, загруженной текущим таском;

- `AF_PURGE_LOADED_DATA` - default - True, указывает необходимость удаления записи, загруженной текущим таском, из таблицы `df.op_data` после завершения обработки;

4.5.23 Встроенные функции трансформации данных

Polyflow включает набор готовых SQL-процедур (процедуры из source/df/sql/callables/common) для типовых операций трансформации данных, которые доступны "из коробки" и могут использоваться в ETL-процессах.

Категории доступных процедур:

1) Валидация и проверка данных:

- `check_format.sql` - валидация форматов данных

2) Преобразование типов данных (безопасное):

- `try_cast_int.sql`, `try_cast_float.sql`, `try_cast_dec.sql` - безопасное приведение к числовым типам
- `try_cast_dt.sql`, `try_cast_dtm.sql` - безопасное приведение к дате и времени

- `try_convert_int.sql`, `try_convert_float.sql`, `try_convert_dec.sql` - альтернативные функции конвертации
- `try_convert_dt.sql`, `try_convert_dtm.sql`, `try_convert_dt_ymd.sql` - конвертация дат с различными форматами

3) Работа со строками:

- `split_string.sql`, `split_string_ord.sql` - разделение строк с сохранением порядка
- `split_part.sql`, `split_part_tab.sql` - разделение строк на части
- `trim.sql`, `trim_left.sql`, `trim_right.sql` - обработка пробельных символов

4) Вспомогательные функции:

- `to_log.sql` - логирование операций трансформации в системные таблицы

Пример использования в ETL-процессе:

```sql
-- Безопасное преобразование данных при загрузке
INSERT INTO target_table
SELECT
    try_cast_int(raw_id) as id,
    try_convert_dt(raw_date) as date,
    trim(raw_name) as name,
    try_cast_dec(amount) as amount
FROM source_table;
```

Преимущества использования:

- Отказоустойчивость: Функции с префиксом try_ обрабатывают ошибки преобразования, возвращая NULL вместо прерывания процесса

- Стандартизация: Единообразная обработка данных across всеми проектами
- Производительность: Оптимизированные реализации часто используемых операций
- Совместимость: Поддержка различных диалектов SQL (PostgreSQL, MS SQL Server)

Эти процедуры интегрируются с основными экстракторами Polyflow и могут использоваться как на этапе извлечения, так и на этапе трансформации данных.

4.6 Loaders (Загрузчики)

Инструменты загрузки/кэширования (loaders) данных из различных источников.

Как правило, данные загружаются в виде файлов, которые в дальнейшем обрабатываются соответствующими экстракторами. Корневая системная директория для загруженных/закэшированных файлов: `/app/cache` (том с обязательным правом на запись, но без права на исполнение)

Системные раннер: `/app/source/python/run_loader.py` . Для подключения к провайдеру данных необходимо создать объект `Connection`, с типом подключения, определяемым конкретным загрузчиком, и указать его атрибуты. Поддерживаемые `Extra` опции подключения, задаются в виде JSON словаря:

- `run` - режим запуска, `default = `wait'`; поддерживается:
- `once` - однократная проверка с последующей загрузкой, если файлы найдены,
- `wait` - ждать пока файлы не появятся и завершить работу после их загрузки
- `repeat` - возобновить поиск после загрузки

- `pattern` - маска поиска файла (regular expression match), `default = ^(.*)$`
- `interval` - интервал в секундах между проверками наличия файлов по маске, `default = 300`
- `timeout` - таймаут ожидания появления файлов, соответствующих маске, в секундах, `default = 86400`
- `mode` - режим загрузки, `default = copy`:
- `empty` - очистить локальную папку перед загрузкой
- `copy` - загружать файл
- `cut` - загружать и удалять файл с удаленного контейнера (директории)
- `unpack` - распаковать загруженные архивы
- `delete` - удалить загруженные файлы Данные опции являются общими для всех загрузчиков.

4.6.1 Loader из облака POLYHUB

Взаимодействует с API `PolyHub`, предоставляя следующий функционал:

- поиск файлов в бакете облака в соответствии с переданной маской

- загрузка найденных файлов в локальный кэш

- опциональное удаление найденных файлов из облака после их успешной загрузки

- опциональное перемещение найденных файлов в папку облака после их успешной загрузки

- возможность как беспрерывного мониторинга облака, так и завершения работы после загрузки найденных в рамках текущего запуска файлов

Для подключения к провайдеру данных необходимо создать объект `Connection`.

Атрибуты подключения:

- `Conn Id` - уникальный в пределах инстанса идентификатор подключения, например - `polyhub_local`

- `Conn Type` - тип подключения, как правило - `HTTP`

- `Host` - базовый url API облака, например - https://example.com

- `Port` - порт подключения, например - `44390`, для `https` в `Host` по умолчанияю - `443`

- `Extra` - дополнительные опции подключения к провайдеру

Поддерживаемые `Extra` опции подключения, задаются в виде JSON словаря:

- `provider` - код провайдера (известного лоадера), например - `polyhub`

- `mode` - режим загрузки, поддерживаются все общие режимы, указанные для загрузчиков; дополнительно к общим для всех загрузчиков режимам, поддерживается:

- `move` - перемещать скачанные файлы в папку корзины, идентификатор которой указан в `AF_LOADER_PARENT`. В данном режиме дополнительно необходимо задавать `AF_LOADER_CONTAINER` - идентификатор папки поиска файлов в корзине облака, т.е. ограничить поиск файлов определенной папкой

- `upload` - загружать локальные файлы из папки `AF_LOADER_STORAGE` в папку корзины, идентификатор которой указан в `AF_LOADER_CONTAINER`. Если заданы `AF_LOADER_PARENT`/`AF_LOADER_FAILURE_PARENT`, то успешно/не успешно обработанные файлы будут перемещены в указанные в переменных папки

- `token` - токен подключения к облаку, переопределяет `Connection.Password`, который ограничен 255 символами, выдается провайдером

- `options` - дополнительные параметры конкретного загрузчика, в частности:

- `ssl_verify` - true/false проверка SSL сертификата, по умолчанию true

- `container` - идентификатор контейнера (директории) поиска файлов в корзине облака, заданной токеном, default = '' (корень бакета)

- `parent` - идентификатор контейнера (директории) в корзине облака, в которую необходимо перемещать файлы после их успешного скачивания в режиме `move`

- `descent` - загружать структуру хранения файла, по умолчанию 1 (только файл), 0 - все уровни (вышестоящие директории)

Все опции могут быть переопределены переменными окружения таска с добавлением префикса `AF_LOADER_`

Поддерживаемые переменные окружения таска (помимо переопределяющих опции подключения):

- `AF_PROVIDER_CONNECTION` - `Conn Id` подключения к облаку, например - `polyhub_local`

- `AF_TARGET_STORAGE` - директория сохранения/кэширования загруженных из облака файлов, по умолчанию - `<task_id>/<run_id>` относительно `/app/cache`.

4.6.2 Loader из FTP

Предоставляет следующий функционал:

- поиск файлов на FTP в соответствии с переданной маской

- загрузка найденных файлов в локальный кэш

- опциональное удаление найденных файлов с FTP после их успешной загрузки

- возможность как беспрерывного мониторинга FTP, так и завершения работы после загрузки найденных в рамках текущего запуска файлов

Для подключения к провайдеру данных необходимо создать объект `Connection`.

Атрибуты подключения:

- `Conn Id` - уникальный в пределах инстанса идентификатор подключения, например - `ftp_local`

- `Conn Type` - тип подключения, как правило - `FTP`

- `Host` - доменное имя / ip FTP, например - 0.0.0.0

- `Port` - порт подключения, например - `11021`, по умолчанияю - `21`

- `login` - логин пользователя

- `Password` - пароль пользователя

- `Extra` - дополнительные опции подключения к провайдеру

Поддерживаемые `Extra` опции подключения, задаются в виде JSON словаря:

- `provider` - код провайдера (известного лоадера), например - `ftp`

- `type` - тип подключения: FTP или FTPS: , default = `FTP`

- `level` - максимальный уровень дерева каталогов, в которых будет производиться поиск (None, 0 - нет ограничений, 1 - корневой каталог), default = `None`

- `min_level` - минимальный уровень дерева каталогов для скачивания файлов (None, 0, 1 - корневой каталог), default = `1`

- `time_shift` - значение сдвига времени, измеряемое в секундах. Сдвиг времени - различие между местным временем сервера (FTP) и местным временем клиента (сервер polyflow) в данный момент, т.е. по определению: `time_shift = server_time - client_time`. Например, для случая, когда на FTP-сервере `UTC+05:00`, а на сервере polyflow - `UTC+00:00`, значение будет равным `18000`

- `filter` - опциональный параметр. Строка, содержащая путь до jinja-шаблона с python-кодом и имя функции, указанное через `::`, например - `/app/ws/source/df/python/scripts/custom/examples/filter_files.pyj::filter_by_date`. В шаблоне доступны локальные переменные и переменные окружения. Заданная функция будет использована для фильтрации загружаемых файлов: должна возвращать True для файлов, которые должны быть скачаны. На вход функция принимает объект, поля которого являются свойствами файла:

- `name` - имя файла (например, `name: 'file_cccc'`)

- `path` - путь до файла, относительно рабочей директории (например, `path: 'dir_2/dir_22/dir_221/file_cccc'`)

- `stat` - стандартный объект типа os.path, содержащий информацию о файле: размер, дата изменения и т.п. (например, `stat: StatResult(st_mode=33188, st_ino=None, st_dev=None, st_nlink=1, st_uid='root', st_gid='root', st_size=0, st_atime=None, st_mtime=1635435480.0, st_ctime=None)`)

- `mode` - режим загрузки; дополнительно к общим для всех загрузчиков режимам, поддерживаются:

- `multipattern` - передача списка наименований (масок) файлов, поиск которых осуществляется относительно `container`

- `multicontainer` - передача списка контейнеров (рабочих директорий), относительно которых выполняется поиск по `pattern`

- `options` - дополнительные параметры конкретного загрузчика, в частности:

- `container` - контейнер (директория) поиска файлов на FTP, default = '' (домашняя директория, указанная в настройках FTP)

Одновременная передача списка файлов (масок) и списка контейнеров (директорий) недопустима.

Все опции могут быть переопределены переменными окружения таска с добавлением префикса `AF_LOADER_`

Поддерживаемые переменные окружения таска (помимо переопределяющих опции подключения):

- `AF_PROVIDER_CONNECTION` - `Conn Id` подключения к облаку, например - `ftp_local`

- `AF_TARGET_STORAGE` - директория сохранения/кэширования загруженных с FTP файлов, по умолчанию - `<task_id>/<run_id>` относительно `/app/cache`

4.7 Вызовы API внешних систем

Выполняются отдельными специализированными раннерами.

Поддерживаются следующие общие переменные окружения:

- `AF_DWH_DB_CONNECTION_PRODUCER` - сериализованное подключение к внешней системе,

- `AF_METHOD` - метод вызова (`get`, `post`), по умолчанию - `get`

- `AF_ENDPOINT` - API endpoint (ресурс), например, `vqadmin/api/databases/db/loadplans` или `backbone/set/176131`

- `AF_RESPONSE_HANDLER` - путь (указатель) на функцию, в которую будут передан результат вызова. Функции на вход передаются следующие параметры:

  - `response` - результат вызова API;

- `AF_RESPONSE_KWARGS` - список именованных аргументов, значения которых дополнительно передаются в `AF_RESPONSE_HANDLER`, по умолчанию - None

4.7.1 API Visiology

Выполняется отдельным раннером `/app/ws/source/df/python/run_visiology.py`.

Поддерживаются выполнение вызовов API Visiology (ViQube, SmarForms, ViQube Admin).

4.7.2 API NetDB

Выполняется отдельным раннером `/app/ws/source/df/python/run_netdb.py`.

Поддерживаются выполнение вызовов API NetDB.

4.7.3 API POLYHUB

Выполняется отдельным раннером `/app/ws/source/df/python/run_polyhub.py`.

Поддерживаются выполнение вызовов API POLYHUB.

Общие переменные окружения:

- `AF_ENDPOINT` - API endpoint (опционально, по умолчанию `api/graphql`).

Дополнительно к общим переменным окружения (кроме `AF_METHOD`) поддерживаются следующие:

- `AF_QUERY` - тело GraphQL-запроса, например:

```
mutation ($input: ExecuteTriggerAlertInput) {
  public {
    alert {
      executeTrigger(input: $input) {
        ok
      }
    }
  }
}
```

- `AF_PARAMS` - параметры запроса, например:

```
{
    "input": {
        "token": "<token>",
        "arguments": []
    }
}
```

- `AF_TOKEN` - токен доступа (уведомления, корзины или другого объекта в POLYHUB),

- `AF_ATTACHMENTS` - список путей вложений (используется для прикрепления вложений к уведомлению), например: `['/app/ws/share/df/examples/csv/simple.csv']`.

Пример использования приведен в процессе [`df_test_polyhub_operator`](operator/ws/dags/df/examples/phapi/operator.py).

4.7.4 Работа с другими внешними сервисами через HTTP API

Polyflow поддерживает интеграцию с различными внешними системами через HTTP API, включая популярные CRM-системы, такие как amoCRM. Интеграция осуществляется с использованием стандартных операторов и подключений HTTP.

4.7.4.1 Общий принцип интеграции

Для работы с внешними API в Polyflow используются следующие компоненты:

- HTTP Connection - настройка подключения к внешнему сервису
- PythonOperator/DockerOperator - выполнение пользовательского кода для взаимодействия с API
- HTTP API адаптеры - специализированные обработчики для различных форматов данных

4.7.4.2 Пример интеграции с amoCRM

Настройка подключения к amoCRM:

```
#python
# Создание Connection для amoCRM в интерфейсе Polyflow
Conn Id: amocrm_connection
Conn Type: HTTP
Host: https://your_domain.amocrm.ru
Extra: {
    "auth_type": "bearer_token",
    "api_version": "v4"
}
```

Пример реализации DAG для работы с amoCRM:

```
#python
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime
import requests
```

```
def get_amocrm_access_token():
    """
    Получение access token для amoCRM API
    """
    auth_url = "https://your_domain.amocrm.ru/oauth2/access_token"
    auth_data = {
        'client_id': 'your_client_id',
        'client_secret': 'your_client_secret',
        'grant_type': 'authorization_code',
        'code': 'authorization_code',
        'redirect_uri': 'your_redirect_uri'
    }

    response = requests.post(auth_url, data=auth_data)
    return response.json()['access_token']

def extract_amocrm_contacts():
    """
    Извлечение контактов из amoCRM
    """
    access_token = get_amocrm_access_token()
    headers = {'Authorization': f'Bearer {access_token}'}

    contacts_url = "https://your_domain.amocrm.ru/api/v4/contacts"
    response = requests.get(contacts_url, headers=headers)

    # Обработка и сохранение контактов
    contacts_data = response.json()
    return contacts_data

def create_amocrm_lead():
    """
    Создание сделки в amoCRM
    """
    access_token = get_amocrm_access_token()
```

```
    headers = {
        'Authorization': f'Bearer {access_token}',
        'Content-Type': 'application/json'
    }

    lead_data = {
        "name": "Сделка из Polyflow",
        "price": 10000,
        "_embedded": {
            "contacts": [{"id": 123456}]
        }
    }

    leads_url = "https://your_domain.amocrm.ru/api/v4/leads"
    response = requests.post(leads_url, headers=headers, json=lead_data)
    return response.json()

with DAG('amocrm_integration',
         start_date=datetime(2024, 1, 1),
         schedule_interval='@daily') as dag:

    extract_contacts = PythonOperator(
        task_id='extract_amocrm_contacts',
        python_callable=extract_amocrm_contacts
    )

    create_lead = PythonOperator(
        task_id='create_amocrm_lead',
        python_callable=create_amocrm_lead
    )

    extract_contacts >> create_lead
```

4.7.4.3 Интеграция с Яндекс Метрикой

Polyflow поддерживает получение данных из API Яндекс Метрики для формирования отчетов и передачи данных о конверсиях из CRM-систем. Интеграция реализуется через универсальные адаптеры для HTTP API.

Общий принцип интеграции:

1) Аутентификация: для доступа к API необходимо получить OAuth-токен в Яндекс ID.

2) Подключение: в Airflow создается объект Connection типа HTTP.

3) Получение данных: для формирования отчетов используются методы API отчетов с указанием метрик и группировок.

4) Передача данных: для отправки информации о конверсиях (например, изменение статуса сделки) используется Measurement Protocol.

Пример реализации DAG для получения отчета:

```
#python
from airflow import DAG
from airflow.operators.docker_operator import DockerOperator
from datetime import datetime


default_args = {
    'owner': 'data_team',
    'start_date': datetime(2024, 1, 1),
}


with DAG('yandex_metrica_report',
         default_args=default_args,
         schedule_interval='@daily') as dag:


    get_metrica_report = DockerOperator(
        task_id='get_metrica_report',
        image='df_operator:latest',
```

```
        command='python /app/ws/source/df/python/run.py',
        environment={
            'AF_SCRIPT_PATH':
'/app/ws/source/df/python/scripts/ya/run_metrica_extract.py', #
пользовательский скрипт реализации
            'AF_PROVIDER_CONNECTION': 'yandex_metrica_connection',
            'AF_METRICS': 'ym:s:visits,ym:s:users',
            'AF_DIMENSIONS': 'ym:s:date,ym:s:trafficSource',
            'AF_DATE_FROM': '{{ ds }}',
            'AF_DATE_TO': '{{ ds }}'
        }
    )
```

Рекомендации:

- Для хранения токенов доступа используйте Airflow Variables.
- Учитывайте ограничения API Яндекс Метрики на количество запросов.
- Для отладки запросов к API используйте стандартные инструменты логирования в Polyflow.

4.7.5 Поддерживаемые внешние системы

Polyflow поддерживает интеграцию со следующими категориями внешних систем через HTTP API:

- CRM-системы:
  - amoCRM - российская CRM-система для управления продажами
  - Другие CRM с HTTP API поддержкой
- Биллинг и платежи:
  - Платежные шлюзы
  - Финансовые системы
- Аналитические платформы:
  - Системы бизнес-аналитики
  - Платформы метрик и мониторинга
- Прочие системы:

- ERP-системы
- Сервисы нотификаций
- Облачные хранилища

4.7.6 Типовые сценарии использования

1) Выгрузка данных из внешних систем в хранилище данных
2) Загрузка результатов обработки во внешние системы
3) Синхронизация данных между различными платформами
4) Автоматизация бизнес-процессов с внешними сервисами

4.7.7 Рекомендации по настройке

- Используйте Airflow Variables для хранения чувствительных данных
- Настройте повторные попытки при временных сбоях API
- Реализуйте обработку ошибок и нотификации
- Соблюдайте лимиты запросов внешних API

4.8 Безопасность данных в процессах ETL

Шифрование чувствительных данных:

- Хранение паролей и ключей в Airflow Variables (шифруются в БД)
- Использование Secrets Backend для управления секретами
- Шифрование файловых операций через SSL/TLS

Пример безопасного подключения:

```
#python
# Данные подключения хранятся в зашифрованном виде (dag)
environment={
      'AF_DWH_DB_CONNECTION_PRODUCER': serialize_connection('testmssql'),
      'AF_DWH_DB_CONNECTION_CONSUMER': serialize_connection('testpostgresql'),
```

```
      ....
},


# Данные подключения хранятся в зашифрованном виде (user script)
def get_ms_conn():
...
    conn_param = get_conn_parameters(connection=getenv('AF_SRC_CONNECTION'))
    server=conn_param['host']
    port=conn_param['port']
    db=conn_param['schema']
    user=conn_param['login']
    pwd=conn_param['pass']
```

4.9 Описание базовых системных объектов продукта

4.9.1 Хуки

4.9.1.1 VisiologyHook

Подключение и низкоуровневое взаимодействие с API Visiology. Использует Connection к Visiology.

4.9.1.2 NetDBHook

Подключение и низкоуровневое взаимодействие с API NetDB. Использует Connection к NetDB.

4.9.1.3 EmailHook

Отправка писем с использованием системных и пользовательских шаблонов для темы и тела письма. Пользовательские шаблоны должны располагаться относительно директории `source/df/common/email`.

Поддерживаются шаблоны Jinja.

Дополнительно к стандартным переменным Airflow (Templates reference), доступны переменные `df_now` (текущая дата со временем в UTC), `base_url` (значение параметр `base_url` из секции `[webserver]` конфига `airflow.cfg`), `log_url` (ссылка на лог в интерфейсе приложения).

4.9.1.4 PolyhubHook

Подключение и низкоуровневое взаимодействие с API POLYHUB. Использует Connection к POLYHUB.

Пример использования приведен в процессе [`df_test_polyhub_hook`](operator/ws/dags/df/examples/phapi/hook.py).

Доступен только в контейнере df_operator.

4.9.2 Операторы

4.9.2.1 DockerOperator

Внутри оператора используется python 3.9.

При передаче данных из оператора через XCom (при xcom_push=True и xcom_all=False - получение только последнего сообщения) из-за специфики оператора возможно непредсказуемое возвращение сообщений из логов и стандартного потока вывода. Для решения данной проблемы существует возможность добавления возвращаемого значения в контекст. Данный контекст используется только в рамках текущего таска. Все содержимое контекста перед завершением таска печатается в стандартный поток вывода.

Для разделения XCom и других сообщений из стандартного потока вывода в операторе предусмотрены переменные окружения:

- `AF_SAFE_XCOM`: при True (значение по умолчанию), выводимые через Context значения, оборачиваются в теги `<![XCOMDATA[`, `]ATADMOCX]!>`, для разделения

их от сообщений из логов и стандартного потока вывода контейнера;

- `AF_ENCODE_XCOM`: при True (значение по умолчанию), выводимые через Context значения, кодируются в base64.

ВАЖНО. Для вывода результата в XCom с помощью bash-команды echo выводимое значение:

- при AF_SAFE_XCOM = True должно быть обернуто в `<![XCOMDATA[`, `]ATADMOCX]!>`,

- либо должно выводиться при AF_SAFE_XCOM = False

Работа с контекстом:

- для добавления значения используется функция `context_push(value: Any)`, добавляемое значение переписывает предыдущее;

- для добавления значения типа dict используется функция `context_update(value: dict)`, добавляемое значение обновляет предыдущее;

- для добавления значения типа list (или единичного в список) используется функция `context_extend(value: list | Any)`, добавляемое значение расширяет предыдущее(-ие);

- для получения значения внутри текущего таска используется `context_pull(default: Optional[Any] = None) -> Any`;

- для инициализации контекста с записью содержимого в файл доступна функция `init_context(pickler: Optional[str] = None) -> None`:

- `pickler` - имя библиотеки сериализации. Доступны следующие значения: `pickle` (значение по умолчанию), `json`.

Файлы создаются в папке cache/rendered/context/<task_folder> на томе df_workspace, где <task_folder> для каждого таска по шаблону `dag_id={dag_id}/run_id={run_id}/task_id={task_id}[/map_index={map_index}]/attempt={try_number}`. После завершения работы таска файлы удаляются. Для их сохранения необходимо в переменную окружения `AF_KEEP_RENDERED` передать значение `True`.

ВАЖНО:

- При возвращении значения таском (т.е. добавлении его в XCom) следует помнить, что без дополнительной обработки пользовательской функцией можно вернуть только нативно сериализуемые в json типы.

- Не рекомендуется использовать в скриптах, возвращающих значение через XCom, вывод stdout (использование print), т.к. распечатанное таким образом значение может переопределить возвращаемое в XCom значение, если для `AF_SAFE_XCOM` и `AF_ENCODE_XCOM` установлены в False.

[Пример DAG'a с использование контекста для передачи данных из DockerOperator](operator/ws/dags/df/examples/mixed/docker_complex_xcom_context.py)

4.9.2.2 VisiologyAPIOperator

Предназначен для обращения к API Visiology. Использует VisiologyHook.

Позволяет выполнять произвольные запросы к API ViQube, SmarForms, ViQube Admin.

Функционал оператора продублирован в системном раннере `source/df/python/run_visiology.py`, который рекомендуется использовать вместо VisiologyAPIOperator.

4.9.2.3 NetDBAPIOperator

Предназначен для обращения к API NetDB. Использует NetDBHook

Функционал оператора продублирован в системном раннере `source/df/python/run_netdb.py`, который рекомендуется использовать вместо NetDBAPIOperator.

4.9.3 Сенсоры

4.9.3.1 LocalFileSensor

Проверяет наличие объектов по указанному пути, поддерживает фильтрацию регулярными выражениями.

Обязательные аргументы сенсора:

- `path` - путь поиска, как правило, относительно `/app/share`

Опциональные аргументы сенсора:

- `pattern` - маска (регулярное выражение) поиска (`default: ^.+$`)
- `trigger_delay` - отложенное срабатывание (сек., `default: 0`)
- `files_only` - срабатывать только на файлы (`boolean`, `default: True`)
- `match_name` - сверять только наименование объекта, а не пути (`boolean`, `default: True`)
- `lazy` - срабатывать при первом совпадении или искать все (`boolean`, `default: True`)
- `do_xcom_push` - возвращать найденные объекты как xcom (`boolean`, `default: False`), стандартный аргумент с инвертированным умолчанием,
  при `lazy = True` возвращается путь как строка, иначе - список путей
- `depth` - максимальная глубина вложенности (`int`, `default: 1`, при 0 - глубина не ограничена), если не равен 1, автоматически включает `do_xcom_push` и отключает `lazy`, если они не заданы.

4.9.4 Соединения

4.9.4.1 Visiology

Conn Type: HTTP.

Требует наличия Host, Login, Password.

Поддерживает Extra параметры: `unit`, `api_version`, `ssl_verify`.

Параметр `unit` обязателен и может принимать следующие значения: `viqube`, `dc`.

4.9.4.2 NetDB

Conn Type: HTTP.

Требует наличия Host, а также:

- либо Login и Password для подключения с использованием `sessionid`

- либо `{ "api_key": "your-netdb-api-key" }` в Extra для подключения через `Netdb-Api-Key` (приоритетный вариант) Поддерживает Extra параметры: `api_key`, `ssl_verify`, `api_version`.

4.9.4.3 MS SQL

Объекты `Connection` для подключения к СУБД в `Extra` поддерживают дополнительные опции, добавляемые к строке подключения.

Опции задаются в нижнем регистре с символом нижнего подчеркивания между словами (`under_score` или `snake_case`).

Например: для опций `Encrypt=yes;TrustServerCertificate=yes` соответствующая `Extra` будет иметь следующий вид `{"encrypt": "yes", "trust_server_certificate": "yes"}`.

4.9.5 Контроль качества данных

Подсистема контроля качества предназначена для проверки соответствия данных КХД заданным критериям.

Правила контроля описываются метаданными в формате JSON, что позволяет их настройку как в текстовом редакторе, так и в визуальном контструкторе платформы POLYHUB(polyhub.ru)

Отдельные контроли задаются юнитами и выполняются чекерами. Чекеры представляют собой SQL-функции.

В поставке содежится базовый набор наиболее неоходимых чекеров.

Для более специфичных контролей, можно создавать пользовательские чекеры, которые затем можно использовать наравне с системными.

4.9.5.1 Метаданные чекеров

Инфрмация о запусках правила контроля качества и его параметры записываются в таблицу qc системной схемы.

Получить аргументы, переданные в отдельные чекеры можно с помощью функции `get_checker_arg`.

Функция принимает на вход идентификатор правила, наименование юнита, наименование параметра чекера и тип параметра.

Пример вызова: `select df.get_checker_arg(89, 'test_checker', 'key', null::integer)`.

Результаты выполнения контролей сохраняются в таблицу qc_error системной схемы, либо в пользовательскую таблицу, указанную в правиле контроля качества.

Структура пользовательской таблицы должна соответствовать структуре системной.

4.9.5.2 Создание пользовательских чекеров

Для создания пользовательского чекера необходимо добавить 2 файла.

Один - с его описанием в формате JSON.

А вторым должен быть SQL-скрипт, непосредственно выполняющий контроль.

Созданные файлы следует расположить на томе `df_workspace` по следующим путям:

- JSON-описание - `source/suf/common/schemas/static/rules/ServersideCaseQuerier/checkers`
- SQL-скрипт - `source/suf/sql/callables/rules/ServersideCaseQuerier/checkers`

где `ServersideCaseQuerier` указывает на тип используемого контроллера движка ККД

Переменные, доступные в SQL-скрипте:

- context: - контекст ККД

- entity - сущность (таблица), данные которой должны быть проверены ККД

- rid - атрибут результата запроса, идентифицирующий строку с ошибкой в результатах контроля

- errors_entity - сущность, для результатов ККД

- свойства узла контроля (unit)

- значения параметров чекера (unit.checker.parameters)

SQL-запрос чекера может возвращать следующие атрибуты:

- tid - идентификатор таблицы, тип - oid

- ctid - системный атрибут `ctid`, возвращающий номер секции и строки, тип - tid, в таблице ошибок доступен как атрибуты parition и row

- rid - идентификатор строки, тип - varchar

- col - наименование столбца таблицы, содержащего ошибочные данные, тип - varchar(128)

- jrow - номера, либо идентификаторы строк, содержащих ошибочные данные, как правило, список, тип - JSON

- jcol - атрибуты, содержащие ошибочные данные, как правило, список, тип - JSON

- details - любые дополнительные данные, тип - JSON

Значения атрибутов считываются в порядке их следования. Если передано меньше атрибутов, то значения непереданных атрибутов считаются null.

Все атрибуты являются необязательными, если значения нет, то следует возаращать null.

На данный момент псевдонимы атрибутов не проверяются на соответствие вышеперечисленным, но в случае несоответствия обратная совместимость не гарантируется.

[Пример DAG'a, файлов и дополнительную информацию по ККД](https://gitlab.polyanalitika.ru/system/wiki/-/wikis/Контроль-качества-данных)

4.10 Расположение управляющих системных объектов продукта

4.10.1 DAG'и

- `dags/df/backup_environment` - бэкап папок, потенциально содержащих проектные файлы. Подробное описание работы приведено в [DAG Docs](operator/ws/dags/df/backup_environment.py)
- `dags/df/cleaner` - очистка содержимого папок, переданных через параметры в `dag_run.conf`. Предназначен для вызова другими DAG'ами. Подробное описание работы приведено в [DAG Docs](operator/ws/dags/df/cleaner.py)
- `dags/df/folder_cleanup` - очистка содержимого папок, настроенных через переменную `df_folder_cleanup_params`. Предназначен для использования прикладными разработчиками. Подробное описание работы приведено в [Описание дага очистки директорий, заданных пользователем, df_folder_cleanup](#описание-дага-очистки-директорий-заданных-пользователем-df_folder_cleanup) и [DAG Docs](operator/ws/dags/df/folder_cleanup.py)
- `dags/df/log_cleanup` - периодическая очистка логов и содержимого таблицы `df.op_data`. Подробное описание работы приведено в [Описание дага очистки логов df_airflow_log_cleanup](#описание-дага-очистки-логов-df_airflow_log_cleanup) и [DAG Docs](operator/ws/dags/df/log_cleanup.py)
- `dags/df/op_data_cleanup` - периодическая очистка содержимого таблицы `df.op_data` с дополнительным запуском vacuum. Подробное описание работы приведено в [DAG Docs](operator/ws/dags/df/op_data_cleanup.py) и [Описание дага очистки логов df_airflow_log_cleanup](#описание-дага-очистки-логов-df_airflow_log_cleanup)
- `dags/df/populate_datadb` - заполнение системных таблиц данными, а также удаление старых не используемых системных файлов, указанных в `/ws/share/df/unused_sources.txt`. Подробное описание работы приведено в [DAG Docs](operator/ws/dags/df/populate_datadb.py)

- `dags/df/upgrade_datadb` - создание и обновление системных объектов. Подробное описание работы приведено в [DAG Docs](operator/ws/dags/df/upgrade_datadb.py)

4.10.1.1 Описание дага очистки логов `df_airflow_log_cleanup`

Даг предназначен для периодической очистки логов Airflow и содержимого `df.op_data`. Параметры запуска настраиваются в переменной (Airflow Variables) `df_airflow_log_cleanup_params`. Пример переменной:

```json
{
  "enable_delete": true,
  "schedule": "@daily",
  "alert_email_addresses": [],
  "cleanup_default_params": {
    "depth": 0,
    "method": "size",
    "retention_days": 15,
    "size_mb": 1024
  },
  "folders": {
    "base": {
      "method": "size",
      "size_mb": 200
    },
    "scheduler": {
      "method": "size",
      "size_mb": 100,
      "enable_delete": true
    },
    "dpm": {
      "method": "retention",
      "retention_days": 7
    }
  },
```

```
  "tables": {
    "op_data": {
      "retention_days": 16
    }
  }
}
```

Описание параметров:

- `enable_delete` (bool) - глобальный переключатель - удалять логи старше периода хранения (или суммарный объем которых больше заданного значения) или нет, по умолчанию `true` - удалять,

- `schedule` (str) - период запуска, например: @daily, @weekly, @monthly. По умолчанию `@daily`,

- `alert_email_addresses` (list) - список адресатов, которым отправляется уведомление при возникновении ошибки выполнения DAG;

- `cleanup_default_params` (dict) - словарь с параметрами очистки файлов по умолчанию. Доступны следующие параметры:

  - `depth` (int) - глубина поиска и удаления файлов в указанной директории; например: значение `1` ограничивает поиск только директорией `path`, т.е. файлы будут искаться и удаляться только в этой папке, `2` - поиск и удаление файлов в директории `path` и в ее непосредственных подпапках, `0` - поиск и удаление файлов во всех вложенных папках без ограничений по глубине; по умолчанию `0`,

  - `method` (str) - метод очистки. Доступны следующие значения:

    - `retention` - исходя из даты создания файлов (значение по умолчанию),

    - `size` - исходя из выделенного места под файлы;

    по умолчанию `retention`,

  - `retention_days` (int) - количество дней, за которые необходимо хранить файлы, по умолчанию `15`,

- `size_mb` (int) - объем памяти на диске в мегабайтах, доступный для файлов. При превышении места под логи в режиме очистки `size` удаляются самые старые файлы, по умолчанию `1024`;

- `folders` (dict(dict)) - список словарей с директориями и их параметрами очистки. Доступны следующие настройки следующих директорий:

- `base` (str) - параметры очистки для основной директория логов (значение настройки BASE_LOG_FOLDER из logging конфига airflow.cfg),

- `scheduler` (str) - параметры очистки для логов планировщика (значение настройки CHILD_PROCESS_LOG_DIRECTORY
из scheduler конфига airflow.cfg),

- `dpm` (str) - параметры очистки для логов менеджера процессов (значение настройки DAG_PROCESSOR_MANAGER_LOG_LOCATION из logging конфига airflow.cfg),

Для каждой директории доступны следующие параметры:

- `depth` (bool) - глубина поиска и удаления файлов в указанной директории `path`; по умолчанию используется значение `depth` из `cleanup_default_params`,

- `method` (str) - метод очистки. Значения совпадают со значениями параметра `method` из `cleanup_default_params`. По умолчанию используется значение `method` из `cleanup_default_params`,

- `retention_days` (int) - количество дней, за которые необходимо хранить файлы. Передается при `method = 'retention'`, по умолчанию используется значение `retention_days` из `cleanup_default_params`,

- `size_mb` (int) - объем памяти на диске в мегабайтах, доступный для файлов. Передается при `method = 'size'`, по умолчанию используется значение `size_mb` из `cleanup_default_params`,

- `enable_delete` - переключатель - удалять логи или нет (bool), по умолчанию используется значение `enable_delete` из переменной `df_airflow_log_cleanup_params`.

- `op_data_retention_days` - период хранения файлов, загруженных в df.op_data, задается в днях (по умолчанию 15 дней).

Следующие параметры устарели, при настройке рекомендуется использовать новые из переменной `df_airflow_log_cleanup_params`:

- `max_log_age_in_days` - новый параметр `df_airflow_log_cleanup_params.cleanup_default_params.retention_days`
- `max_log_size_in_mb` - новый параметр `df_airflow_log_cleanup_params.cleanup_default_params.size_mb`
- `max_proc_log_age_in_days` - `новый параметр df_airflow_log_cleanup_params.folders.dpm.retention_days`
- `enable_delete_child_log` - новый параметр `df_airflow_log_cleanup_params.folders.scheduler.enable_delete`
- `op_data_retention_in_days` - новый параметр `df_airflow_log_cleanup_params.tables.op_data.retention_days`

4.10.2 Основные исполняемые файлы

- `source/df/excel/run.py` - экстракт данных из Excel файлов
- `source/df/hippo/run.py` - экстракт данных из CSV файлов
- `source/df/python/run.py` - выполнение python-скриптов
- `source/df/python/run_exec.py` - выполнение python-скриптов (устаревший вариант `source/df/python/run.py`)
- `source/df/python/run_loader.py` - выполнение загрузки/кэширования (loaders) данных из различных источников (чаще всего в виде файлов)
- `source/df/python/run_sql_move.py` - выполнение загрузки данных из БД
- `source/df/python/run_visiology.py` - выполнение вызовов API Visiology (ViQube, SmarForms, ViQube Admin)

- `source/df/python/run_netdb.py` - выполнение вызовов API NetDB
- `source/df/python/run_polyhub.py` - выполнение вызовов API POLYHUB
- `source/df/python/run_sql_to_viapi.py` - выполнение загрузки результатов SQL-запросов в Visiology DataCollect через DC API
- `source/df/qc/run.py` - выполнение правил ККД
- `source/df/sql/merge_data_json.py` - merge данных в таблицы БД
- `source/df/sql/run.py` - выполнение SQL-скриптов
- `source/df/sql/run_create_entity.py` - создание таблиц в БД
- `source/df/sql/run_procedure.py` - выполнение процедуры с возможностью предварительной загрузки метаданных в контекст
- `source/df/sql/run_statement.py` - выполнение SQL-скриптов по выражениям
- `source/df/viapi/viqube/run.py` - выполнение Http запросов к ViQube API

4.11 Создание и поддержка объектов БД

DDL операции выполняются с использованием alembic. Штатный запуск alembic выполняется DAG'ами, например, `df_upgrade_datadb`.

Для ручного запуска alembic с хоста необходимо прописать строку подключения sqlalchemy.url в соответствующем alembic.ini.

4.12 Логирование в системе

4.12.1 Логирование в тасках

Настройка уровня логирования задается:

- для системы с помощью параметра `logging_level` раздела `logging` файла `airflow.cfg`

- для оператора через переменную окружения `AF_LOGLEVEL`

Для периодической очистки логов в системе используется даг `df_airflow_log_cleanup`.

На стендах с высокой активностью запуска дагов в случае увеличения количества логов рекомендуется:

- увеличить периодичность запуска дага `df_airflow_log_cleanup`

- использовать в системе уровень логирования `ERROR`

- не оставлять на промышленных решениях в дагах операторы с уровнем логирования `DEBUG`

4.12.2 Логирование в SQL-скриптах

Функция `df.to_log` используется для добавления записей в таблицу логов `df.log`. Она позволяет логировать различные данные, такие как сообщения, описания, уровень логирования и дополнительные данные в формате JSON. Функция гибко настраивается с помощью следующих параметров:

- `p_op_ins_id`: character varying (по умолчанию NULL)

Идентификатор таска, связанный с записью лога. Это внешний ключ, который ссылается на таблицу `df.op_ins`. Значение доступно через подстановку `{{AF_OP_INS_ID}}`.

- `p_caller`: character varying (по умолчанию NULL)

Имя компонента или модуля, который вызывает функцию.

- `p_message`: character varying (по умолчанию NULL)

Основное сообщение для записи в лог.

- `p_description`: character varying (по умолчанию NULL)

Дополнительное описание или детали для записи.

- `p_type`: character varying (по умолчанию NULL)

Тип записи логирования.

- `p_value`: character varying (по умолчанию NULL)

Значение или данные, связанные с записью.

- `p_data`: character varying (по умолчанию NULL)

Дополнительные данные в текстовом формате.

- `p_jdata`: json (по умолчанию NULL)

Дополнительные данные в формате JSON.

- `p_level`: character varying (по умолчанию 'INFO')

Уровень логирования, например, 'INFO', 'ERROR', 'DEBUG'.

- `p_to_log`: boolean (по умолчанию false)

Флаг, указывающий, следует ли выводить добавленную запись в лог таска.

- `p_show_notice`: boolean (по умолчанию false)

Флаг, указывающий, следует ли показывать уведомление (используется для Postgres для raise notice).

Логирование доступно только для скриптов, выполняемых через `source/df/sql/run.py` для Postgres.

Для логирования необходимо наличие плагина `dblink` установленного в системную схему `df`, а также наличие прав на него у пользователя хранилища `dwh`.

Пример вызова

```sql
CALL df.to_log(
    p_op_ins_id := op_ins_id,
    p_message := 'num1: ' || num1,
    p_show_notice := true,
    p_to_log := true
);
```

В этом примере функция `df.to_log` вызывается с заданными параметрами `p_op_ins_id`, `p_message`, `p_show_notice`, и `p_to_log`. Сообщение содержит значение переменной `num1`, уведомление будет показано, и запись будет добавлена в лог таска. По умолчанию в лог выводится только поле `df.log.message`.

4.12.2.1 Использование пользовательских обработчиков SQL-логов (переменная `AF_NOTICES_HANDLER`)

Поддерживается переменная для задания пользовательской функции для обработки SQL-логов:

- `AF_NOTICES_HANDLER`, default = None, путь (указатель) на функцию, в которую будет передан датасет (список словарей), возвращенный результат будет приведен к строке и выведен в лог таска. Функции на вход передаются следующие параметры:

- `dict_list` - записи из таблицы `df.log` для текущего таска (`op_ins_id = {{ AF_OP_INS_ID }}`), помеченные для вывода в лог таска (`to_log = true`), представляет собой список словарей.

- `AF_NOTICES_KWARGS`, default = None, список именованных аргументов, значения которых дополнительно передаются в `AF_NOTICES_HANDLER`.

Дополнительно к пользовательским обработчикам существуют системные.

4.12.2.1.1 Системные обработчики SQL-логов

Системные обработчики доступны в модуле `source.df.common.helpers.adapter`:

- `source.df.common.helpers.adapter.stringify_filtered_dict_list` - используется для фильтрации сообщений лога и преобразования их к строке.

Поддерживаются следующие параметры:

- `dict_list` - записи из таблицы `df.log`;

- `columns` - default = None, список полей из таблицы `df.log`, которые нужно вести. Доступны следующие поля: `op_ins_id`, `caller`, `message`, `description`, `type`, `value`, `data`, `jdata`, `level`, `to_log`;

- `conditions` - default = None, словарь условий фильтрации записей, в котором ключ - это поле из `df.log`, значение - значение записей для заданного поля (единичное или список), которое необходимо вывести.

Пример использования доступен в даге [df_example_raise_notice_sql](operator/ws/dags/df/examples/sql/sql_raise_notice.py)

4.13 Управление файловыми операциями

Polyflow предоставляет комплексные механизмы для операций с файлами, включая "рассылку" по различным путям, мониторинг и обработку ошибок.

4.13.1 Базовые операции копирования и распределения

Использование BashOperator для простого копирования:

```
#python
copy_task = BashOperator(
    task_id='distribute_reports',
    bash_command='cp /app/share/reports/daily_report.csv
/app/share/archive/{{ ds }}/ && '
                 'cp /app/share/reports/daily_report.csv /app/share/backup/'
)
```

Гибкое распределение через PythonOperator

```
#python
def distribute_files(**context):
    import shutil
    execution_date = context['ds']

    source = '/app/share/reports/daily_report.csv'
    destinations = [
        f'/app/share/archive/{execution_date}/',
        '/app/share/backup/',
        '/app/share/departments/finance/'
    ]

    for dest in destinations:
        shutil.copy2(source, dest)

distribute_task = PythonOperator(
    task_id='distribute_files',
    python_callable=distribute_files
)
```

4.13.2 Динамическое управление путями

- Параметризованные пути с использованием шаблонов Jinja2:
- Динамические пути на основе даты выполнения ({{ ds }})
- Пути, зависящие от параметров DAG ({{ params.department }})
- Условное распределение на основе результатов предыдущих задач

Примеры шаблонов распределения:

```
# python
# Распределение по датам
path_template = '/app/share/{{ execution_date.strftime("%Y/%m/%d") }}/'


# Распределение по бизнес-юнитам
department_paths = {
    'finance': '/app/share/finance/reports/',
    'sales': '/app/share/sales/daily/',
    'marketing': '/app/share/marketing/analytics/'
}


# Использование в DockerOperator
file_task = DockerOperator(
    task_id='process_and_distribute',
    image='df_operator:latest',
    command='python /app/process_data.py',
    environment={
        'AF_OUTPUT_PATH': '/app/share/output/{{ ds }}/',
        'AF_BACKUP_PATH': '/app/share/backup/{{
execution_date.strftime("%Y/%m") }}/',
        'AF_DEPARTMENT': '{{ params.department }}'
    },
    params={
        'department': 'finance'
    }
)
```

Передача путей между задачами через XCom:

```
#python
```

```
def generate_file_paths(**context):
    paths = {
        'report_path': f'/app/share/reports/{{{{ ds }}}}/financial_report.csv',
        'archive_path': f'/app/share/archive/{{{{ ds }}}}/',
        'notification_path': '/app/share/notifications/'
    }
    return paths

path_generator = PythonOperator(
    task_id='generate_paths',
    python_callable=generate_file_paths,
    do_xcom_push=True
)

distributor = DockerOperator(
    task_id='distribute_files',
    image='df_operator:latest',
    command='python /app/distribute.py',
    environment={
        'AF_PATHS': '{{ ti.xcom_pull(task_ids="generate_paths") }}'
    }
)
```

4.13.3 Мониторинг файловых операций

Контроль распределения через LocalFileSensor:

```
#python
sensor_tasks = []
destinations = ['/app/share/archive/', '/app/share/backup/', '/app/share/departments/']

for dest in destinations:
    sensor = LocalFileSensor(
        task_id=f'check_file_in_{dest.replace("/", "_")}',
        path=dest,
        pattern='.*daily_report.*\\.csv$',
        timeout=300
```

```
    )
    sensor_tasks.append(sensor)
```

4.13.4 Обработка ошибок и надежность

Повторные попытки и уведомления при сбоях:

```
#python
distribute_task = BashOperator(
    task_id='distribute_with_retry',
    bash_command='cp /app/share/source/file.csv /app/share/destination/ ||
exit 1',
    retries=3,
    retry_delay=timedelta(minutes=5),
    email_on_failure=True
)
```

Ключевые возможности:

- Многоуровневое распределение файлов по динамическим путям
- Мониторинг успешности операций через сенсоры
- Отказоустойчивость с автоматическими повторными попытками
- Интеграция с системой уведомлений о проблемах
- Поддержка сложных сценариев через комбинацию операторов

Такой подход обеспечивает промышленную надежность файловых операций в распределенных средах.

5 Аварийные ситуации

Для Системы определены следующие режимы функционирования:

- штатный;
- аварийный.

Аварийный режим функционирования Системы используется при отказе одного или нескольких компонент программного и (или) технического обеспечения.

При переходе в аварийный режим в Системе предусмотрено формирование соответствующего информационного сообщения.

После выдачи сообщения, администратору необходимо выполнить комплекс мероприятий по устранению причины перехода Системы в аварийный режим.

При работе с АИС могут возникнуть следующие неисправности, приводящие к аварийным ситуациям:

- Превышение нагрузки на АИС. В этом случае необходимо ограничить количество тяжело-нагруженных процессов или общее их количество;
- Недостаток свободной оперативной памяти на сервере. В этом случае необходимо ограничить ресурсы для контейнера.
- Другие неисправности. В случае нарушения технологического процесса или при длительных отказах технических средств администратор системы обязан сообщить о возникшей проблеме в службу технической поддержки, провести диагностику работы Системы, определить вероятную причину неисправности и передать лог-файлы из соответствующего docker-контейнера. Чтобы связаться с службой поддержки необходимо сообщить о возникшей неисправности по электронному адресу: support@polyanalitika.ru.

6 Рекомендации по освоению

Основным источником информации, используемым при освоении Системы, является данное руководство.

Начинать работу с Системой следует со знакомства с разделами руководства «Подготовка к работе», «Описание операций».

Для обеспечения успешной работы пользователям необходимо обладать основными навыками работы с веб-приложениями, опубликованными в сети Интернет.

Лист регистрации изменений

Изм.	*Номера листов (страниц)*				*Всего листов (страниц) в докумен-те*	*Номер докумен-та*	*Входящий номер сопроводи-тельного документа и дата*	*Подпись*	*Дата*
	измененных	*заменен-ных*	*новых*	*аннулиро-ванных*					

Изм.	*Лист*	*№ докум.*	*Подп.*	*Дата*	*.РЭ*	*Лист* 109

Подпись и дата | *Инв. №дубл.* | *Взам. инв. №* | *Подпись и дата* | *Инв. №подл.*

Подпись и дата

Инв. №дубл.

Взам. инв. №

Подпись и дата

Инв. №подл.

					.РЭ	Лист
Изм.	Лист	№ докум.	Подп.	Дата		110