Kolmogorov.ai | Morphism | Руководство по установке

Feature Store

ООО Дата Сапиенс

version 0.20.4 | 12.04.2024

Table of contents

1. Release notes

1.1 Release notes for 0.20

1.1.1 Create Loader

Больше нельзя зарегистрировать загрузчик сегмента с типом `segment`, доступно только `keys`. Все загрузчики сегментов становятся загрузчиками ключей (сегментами без даты), поле с датой в них становится переменной датасета.

1.1.2 Transform %fat_compose

В `dates` больше нельзя указать список из нескольких дат (длина списка должна быть = 1).

1.1.3 System Endpoints

Изменена работа и формат ответа эндпоинта `GET /health/readiness`

Теперь в ответе возвращается информация о соответствии версий (`приложения`, `ядра` и `движка`) между backend'ом и всеми Celery-воркерами. В случае если нет воркеров с совпадающими версиями - возвращается ошибка `ServiceUnavailableException`.

1.1.4 Celery Executor

1. В Celery больше не используется result backend
   - `GET /admin/tasks/{task_id}` - эндпоинт теперь возвращает только значение статуса в формате `PlainText` и рассчитывает его на основе информации, которую возвращают воркеры Celery. Если задача выполняется, то статус = `STARTED`, если зарезервирована, то `PENDING`, иначе `UNKNOWN`
   - `POST /jobs/{job_rk}/stop` - эндпоинт теперь останавливает джоб в синхронном режиме через `terminate()`
2. В чарте приложения брокер сообщений по-умолчанию изменен с Redis на RabbitMQ

1.1.5 Изменение формата ответа эндпоинтов

Следующие эндпоинты теперь возвращают `PlainText` вместо строки в json

- `POST /jobs/{job_rk}/code`
- `POST /admin/ref_init`
- `POST /admin/sys_init`
- `POST /admin/migrate_metastore`
- `POST /admin/reset_metastore`

1.1.6 Изменение работы с db_name

Параметр `db_name` перестает храниться в `meta.v100_source` и становится расчетным. В связи с этим:

1. В эндпоинтах, где `db_name` передавался в схеме запроса, параметр становится `readonly`. Это означает, что включение `db_name` в тело запроса не приведет к ошибке, но не окажет никакого влияния.
2. В эндпоинтах, где `db_name` возвращается в теле ответа, параметр теперь вычисляется на основе диалекта, указанного в параметре `url`.
3. Из эндпоинта `GET /sources` убран фильтр по `db_name`.

Эндпоинты с изменениями в теле запроса и ответа:

- `POST /sources`
- `PATCH /sources/{source_rk}`

Эндпоинты с изменениями только в теле ответа:

- `GET /sources/{source_rk}`
- `GET /sources`

1.1.7 Обновление формата URL для `/sources`

Для эндпоинтов группы `/sources`, где передается URL в теле запроса, изменен формат использования переменных окружения: вместо `$VAR` теперь нужно использовать `${VAR}`.

К примеру, если необходимо указать, что URL для source хранится в переменной окружения `DB_URL`, то в запросе `POST /sources` передается `url = ${DB_URL}`.

Кроме того, теперь можно использовать несколько переменных окружения в параметре `url`. Например, конструкция `${DB_DIALECT}://${DB_URL}` будет успешно преобразована в URL, при условии наличия необходимых переменных окружения в системе.

Затронутые эндпоинты:

- `POST /sources`
- `PATCH /sources/{source_rk}`

1.1.8 Из эндпоинтов удаления V2 убрана поддержка параметра async_flg

В следующих эндпоинтах удален параметр `async_flg`:

- `DELETE /dataset`
- `DELETE /loader`
- `DELETE /entity_link`

Теперь удаление всегда происходит асинхронно. Передача параметра `async_flg` в запросе не приведет к ошибке, но не окажет никакого влияния.

1.1.9 Изменение работы эндпоинта предпросмотра датасетов

`GET /dataset/{dataset_rk}/preview` теперь работает только с теми датасетами, схемы которых зарегистрированы в списке `v100_source.schemas` для `source_rk = -1`.

1.1.10 Удаление эндпоинта Reset Datastore

Эндпоинт `POST /v100/admin/reset_datastore` был удален.

1.1.11 ENVIRONMENT VARIABLES

NEW

- `APP__STAT_DISTINCT_LIMIT` - максимальное количество возвращаемых значений в статистике `DISTINCT`
- `APP__LOCK_TIMEOUT` - максимальное время блокировки (в секундах)
- `S3__HTTP_TIMEOUT` - максимальное время ожидания ответа от S3 в секундах

DEPRECATED

- `EXECUTOR__BACKEND_URL`
- `EXECUTOR__ABORT_PROBE_DELAY`

1.2 Release 0.19 notes

1.2.1 Permissions

1. В ответе `PATCH /permissions` (set permissions) теперь всегда возвращается пустой `name`.
2. Остаются только три типа прав доступа к объектам: `no access`, `read`, `full`. Права доступа `view` и `edit` больше недоступны. Если на момент установки релиза в метаданных есть пермиссии `view` / `edit`, то они мигрируют: `view` -> `read`, `edit` -> `full`. Доступ к данным и доступ к просмотру меты по объекту - это теперь одно и то же и появляется при наличии у пользователя прав доступа `read+`. Больше нет отдельного уровня доступа, при котором у пользователя есть права на просмотр метаданных, но отсутствуют права на чтение данных из датастора.
3. Больше не возвращается `no access exception` - теперь отсутствие прав на объект для пользователя выглядит так, как будто объекта нет в системе. Также меняется поведение при попытке удалить базовые права доступа - ошибка не возвращается, но пермиссия не удаляется.

1.2.2 Catalogue import/export

1. `POST /catalogue/import` теперь работает только с актуальной версией конфига. Актуализировать конфиг нужно перед импортом через отдельный эндпоинт `POST /catalogue/migrate_config`.
2. Изменяется структура конфига - удаляются поля `loaders` и `links`, добавляются поля `datasets` и `jobs`.
3. Импорт становится асинхронным, выполняется в celery. Статус импорта необходимо узнавать через отдельные эндпоинты.
4. Во время работы импорта недоступно создание новых сущностей через api v2 и запуск еще одного импорта. При этом по-прежнему доступно создание сущностей из api v100, т.к. v100 ничего не знает про v2. При создании сущности через api v100 во время работы импорта, корректный результат работы импорта не гарантируется.

1.2.3 Loaders

1. Поле `FROM_DTTM_VALUE` удаляется из всех схем ответов по загрузчикам, остается только в схемах запросов.
2. Поле `SRC_DAG_URL` удаляется из схем запроса/ответа по загрузчикам.
3. В эндпоинте `GET /loader/{loader_rk}` атрибут `DATASET.CSV_PATH` всегда заполнено `null` о

1.2.4 Datasets

1. В эндпоинтах вывода датасетов списком `GET /dataset` и `GET /dataset/page` атрибут `CSV_STATUS` всегда теперь заполнен `null`
2. После установки релиза ссылки на уже сформированные csv потеряются
3. Информация о статусе и ссылке на csv файл остается только в `GET /dataset/{dataset_rk}`

1.2.5 Логирование

В логах больше нет ключа `"traceparent"`, вместо него появляются новые ключи `"trace_id"` и `"span_id"`

1.2.6 v1.0

1. В эндпоинте `POST /api/v100/datasets` параметр `source_rk` перемещается из запроса в схему создания %dataset
2. Из схемы %dataset и %dataset_summary удаляются job_rk и job_state

1.2.7 Files

Для методов `POST v100/files/s3/link`, `POST v100/files/s3/text`, `GET v100/files/s3/text` изменен тип ответа с JSON на PlainText. Например, если раньше возвращалось `"https://example.com"`, то теперь возвращается `https://example.com`.

`POST` `v100/files/s3`

Изменения:

1. Метод теперь не загружает файл, а вместо этого возвращает URL, который можно использовать для загрузки файла напрямую в S3 (через запрос PUT, и контентом файла в body запроса).
2. Удаляется параметр file (тело файла)
3. Параметр path становится обязательным - желаемое имя файла

`GET` `v100/files/s3`

Изменения:

1. `path_mask` переименован в `path_preffix` и не принимает символы типа `*`
2. В ответ не включены файлы из поддиректорий

1.2.8 ENVIRONMENT VARIABLES

NEW

- `SWAGGER` - настройки для swagger
- `SWAGGER__ENABLED` = `true` - доступен ли интерфейс Swagger UI по адресу `/api/doc`, `/tech/doc` и `/api/v100/doc`

NEW (WEB)

- `DOCS_ENABLED` = `false` - включение / отключение документации в интерфейсе

1.3 Release 0.18 notes

1.3.1 Блокировка ИК на время импорта (!)

Во время работы импорта в целях предотвращения неконсистентности метаданных блокируются практически все операции изменения инфокарты - со стороны API это выглядит как "зависение" соответствующих эндпоинтов. Стабильно продожает работать только создание загрузчиков сегментов и датасетов.

NB! Для операций по изменению ИК, запущенных во время работы импорта, не гарантируется корректное выполнение. После завершения работы импорта такие операции могут как примениться к результату импорта, так и упасть с ошибкой. Рекомендуется не проводить такие операции или прерывать их.

1.3.2 Миграция датасетов

- Изменяются суррогатные ключи датасетов
- Суррогатные ключи и имена датасетов-сегментов после миграции будут совпадать с суррогатными ключами и именами связанных загрузчиков
- Удаляются задания расчета датасетов (за исключением сегменов)
- В `GET /entity/{entity_rk}` в поле `DATASET` теперь всегда возвращается пустой список

1.3.3 Миграция загрузчиков

- `FEATURE_COLUMN.KIND` теперь всегда = `feature` (нельзя изменить). Если при создании или редактировании передать `domain`, значение все равно будет `feature`.
- `LOADER.SRC_DAG_URL` теперь всегда = `null` (нельзя изменить)
- История загруженных дат для версий переменных больше не ведется. В результате работы загрузчика переменных для каждой версии в метасторе создается запись с версионностью `-/+ infinity`. Эти же записи возвращаются в `GET /feature/history/{feature_rk}`
- Поля `STG_SCHEMA_NAME` и `STG_COLUMN_NAME` по загрузчикам теперь заполняются также, как поля `SRC_SCHEMA_NAME` и `SRC_TABLE_NAME`. Для file- и sql-загрузчиков эти поля перестают заполняться.
- Для загрузчиков переменных и связей в создаваемой таблице-приемнике поля с датами теперь не всегда называются `from_dttm` и `to_dttm`. Корректные имена полей с датами возвращаются в полях `FROM_DTTM_DB_COL_NM` и `TO_DTTM_DB_COL_NM` в схеме ответа по конкретному загрузчику

1.3.4 Перенос параметра dispose

Параметр `dispose` перенесен из `job_options` на уровень схем %job и %transform

Как следствие - параметр `dispose` теперь нельзя менять в методах:

- `POST /jobs/{job_rk}`
- `PATCH /jobs/{job_rk}`

1.3.5 Отключается поддержка `git` в приложении

- Удалены все переменные `GIT__...`
- Эндпоинты `/api/v100/files` больше не работают с системой git

1.3.6 Асинхронное удаление датасетов, лоадеров, связей

`DELETE /api/v100/datasets/`

- код ответа "200 No Content" меняется на "202 Accepted"
- удаление происходит в фоновом процессе
- новая схема ответа

```
{
  :"runner": str,           # среда исполнения задания
  :"id": str                # идентификатор запуска в среде исполнения
}
```

`DELETE /api/dataset`, **`DELETE /api/loader`**, **`DELETE /api/entity_link`**

Добавляется новый параметр: `async_flg`: bool (default = `APP__DEFAULT_ASYNC_DELETE`)

- Если async_flg=false, то поведение не меняется
- Если async_flg=true, то меняется код и схема ответа:
- код ответа 200 No Content -> 202 Accepted
- удаление происходит в фоновом процессе
- схема ответа:

```
{
  :"runner": str,           # среда исполнения задания
  :"id": str                # идентификатор запуска в среде исполнения
}
```

1.3.7 Изменение набора типов объектов в GET / PATCH /permissions

Все загрузчики (`object_type='loader'`) теперь имеют значение `object_type='dataset'`. Набор доступных значений для `object_type` - (`dataset`, `source`)

1.3.8 Настройка иконок и логотипов в web-интерфейсе

Предустановлены логотипы/иконки для следующих продуктов: Morphism, Audience, Alphyn. Значения переменной `IMAGES_FOLDER_KEY` соответственно `morphism`, `audience`, `alphyn`.

Для настройки собственных иконок и логотипов следующий порядок действий:

- Создать новую папку в `public/images/<folder name>`
- Загрузить в нее изображения со следующими именами:
- `logo_favicon.svg` - иконка для вкладки браузера
- `logo_main.svg` - иконка на главной странице входа
- `logo_title.svg` - логотип в верхнем левом углу интерфейса
- Указать `<folder name>` в переменной `IMAGES_FOLDER_KEY`

1.3.9 Переименование параметра в `v100 GET /datasets`

Параметр запроса `source_rks` переименован в `source_rk`

1.3.10 ENVIRONMENT VARIABLES

DEPRECATED

- `META__WORK_SCHEMA`
- `USE_NEW_V100_REPO`

1.4 Release 0.17 notes

1.4.1 ENVIRONMENT VARIABLES

Deprecated

`USE_NEW_ENTITY_LINK_REPO` `DB__URL` `DB__CONN_OPTIONS` `DB__STAGE_SCHEMA` `DB__STORE_SCHEMA` `DB__WORK_SCHEMA`

NEW

- `USE_NEW_V100_REPO` - выполнять работу с объектами v2, используя целевую архитектуру метаданных v100 (default = `True`). В 0.17 применимо только к загрузчикам. Если `True`, то данные v2 сохраняются как в целевой, так и в легаси архитектуре, при этом чтение происходит только из целевых таблиц metastore. Если `False`, то используются метаданные только из легаси архитектуры.
- `APP__MAX_ITEMS_LIMIT` - максимальное количество элементов в ответе пагинированного эндпоинта.
- `S3` - Добавлена возможность авторизации в S3 хранилище с помощью openid токена (keycloak).
- `S3__AUTH_TYPE` = `openid`: включить авторизацию через openid токен (default = `access_key`). В режиме авторизации через `openid` параметры `S3__ACCESS_KEY` и `S3__SECRET_ACCESS_KEY` не используются.
- `S3__SESSION_TOKEN_LIFETIME` = `3600`: время жизни S3 токена в секундах. Диапазон допустимых значений 900 - 604800 (default = `3600`)
- `SWAGGER` - настройки для swagger
- `SWAGGER__USE_BUNDLED` = `true` - Использовать встроенную версию `.js` и `.css` ресурсов для сваггера
- `SWAGGER__TOKEN_URL` = `null` - кастомный адрес, по которому получать токен
- `APP__DEFAULT_SOURCE_RK` = `-1` - суррогатный ключ источника по умолчанию для запуска заданий

1.4.2 Изменение формата ответа /health

Изменяется схема отчета о состоянии системы.

Вместо `datastore` добавляется секция `sources` с отчетом по подключению к каждому источнику данных из справочника `meta.v100_sources`.

1.4.3 Изменение подключения к датасторам

Настройка подключений к датасторам теперь осуществляется через таблицу `meta.v100_source`.

!NB Источник `source_rk` = -1 (дефолтный датастор) удалять нельзя, т.к. API V2 и узлы `constructor` продолжают работать только с ним.

1.4.4 Загрузчики из внешних датасторов

Загрузчики из SQL/таблиц теперь работают только с источником `source_rk` = -1. Другие источники не могут использоваться для загрузки данных, репликация данных между источниками больше не производится.

1.4.5 Миграция загрузчиков

Загрузчики мигрируют в целевую архитектуру метаданных v100. Важные изменения после миграции:

- Статус загрузчика берется только из задания загрузки (джоба). При удалении связанного джоба, статус загрузчика будет пустым.
- В новой архитектуре снова начинают работать sql-загрузчики, но теперь скрипт должен состоять только из одной инструкции `select` без использования CTE.

Для обеспечения возможности "отката" к легаси архитектуре, в релизе предусмотрена переменная `USE_NEW_V100_REPO` (бывш. `USE_NEW_ENTITY_LINK_REPO`).

1.4.6 Переименование загрузчиков

После миграции загрузчиков, имена загрузчиков связей всегда совпадают с именами связей и изменяются синхронно. Имена всех загрузчиков связей принудительно приводятся к именам связей, которые они грузят. При этом, если среди загрузчиков переменных и загрузчиков сегментов обнаруживается загрузчик с таким же именем, как у связи, то такому загрузчику присваивается случайное сгенерированное имя.

Проверить наличие таких загрузчиков можно с помощью запроса:

```
select * from meta.loader l
where l.type != 'entity_link' and l.name in (
    select name from meta.entity_link
)
```

Такие загрузчики лучше переименовать до установки релиза.

1.4.7 Активация / деактивация загрузчиков

Из API v2 удаляется функциональность активации/деактивации вместе с эндпоинтами:

- POST /loader/activate
- POST /loader/deactivate

Все ранее деактивированные загрузчики принудительно активируются при установке релиза. Если при принудительной активации обнаруживается версия переменной с неуникальным именем, то такой версии переменной присваивается случайное сгенерированное имя.

Проверить наличие таких версий переменных можно с помощью запроса:

```
select distinct fc.*
from meta.loader l
inner join meta.loader_feature_map lfm on lfm.loader_rk = l.loader_rk
inner join meta.feature_column fc on fc.column_rk = lfm.column_rk
inner join meta.feature_column fc_dupl on (
    fc_dupl.feature_rk = fc.feature_rk
    and fc_dupl.name = fc.name
    and fc_dupl.column_rk != fc.column_rk
)
where l.active_flg = false
```

Отобрать деактивированные загрузчики можно с помощью запроса:

```
select * from meta.loader
where active_flg = false
```

Такие загрузчики лучше удалить или активировать до установки релиза.

1.4.8 Переход на SQLAlchemy 2.0

Строка подключения к БД `meta` не должна содержать явного указания драйвера: `postresql://...`

После перехода на второе поколение библиотеки SQLAlchemy для коннекта к БД `meta` используется асинхронный драйвер postgresql - `asyncpg`. Если в строке подключения к БД будет указан другой драйвер (например `postgresql+psycopg2://`), то приложение сообщит об этом и упадет. При этом для миграций `alembic` по прежнему используется драйвер `psycopg2` и приложения самостоятельно принимает решение, какой драйвер надо использовать в зависимости от контекста.

1.4.9 Изменение формата расписания в API V2

Для методов `POST /loader/create`, `PATCH /loader/{loader_rk}`, `GET /loader/{loader_rk}` меняется формат поля `SCHEDULE` для request/response body: вместо схемы расписание теперь задается в формате пятизначной крон-строки.

1.4.10 Удаление набора эндпоинтов датасета API V2

Следующие эндпоинты были удалены:

- `POST /dataset`
- `PUT /dataset/{dataset_rk}`
- `POST /dataset/{dataset_rk}/split`
- `GET /dataset/history/{dataset_rk}`
- `GET /dataset/{dataset_rk}/execution`
- `POST /dataset/impute`
- `POST /dataset/validate`

Функционал метода `POST /dataset/impute` перенесен в API V100 и реализован как узел `fat_impute`.

1.4.11 Изменение эндпоинта DELETE DATASET API V2

Из схемы `reqeust body` удален аргумент `DELETE_FLG`. Теперь эндпоинт удаляет датасет как в метаданных, так и физическую таблицу в datastore (с учетом `force_flg`).

1.4.12 Интеграция с сервисом генерации Excel каталога

В WEB-интерфейсе реализована поддержка интеграции с сервисом генерации каталога в формате Excel. Для этого необходимо указать адрес сервиса в переменную `INFOMAP_GENERATOR_API` в values.yaml веб-интерфейса.

1.5 Release 0.16 notes

> **Важно**
>
> Airflow больше не нужен для работы приложения. Перед установкой релиза **важно** ознакомиться с обновленной инструкцией по установке

1.5.1 Изменение правил создания таблицы датасета

1. Для всех новых датасетов имя таблицы теперь всегда совпадает с именем датасета при регистрации (исключение - датасеты, полученные через `POST /transform` с явным указанием параметра `to_name`).
2. Создание таблицы датасета в dag_executor теперь происходит без предварительного выполнения `drop table if exists` (исключение - датасет-сегмент). Проверка уникальности имени таблицы теперь производится только по каталогу датасетов в метаданных. Если датасет с такой таблицей отсутствует в metastore в момент регистрации, но при этом в момент его расчета такая таблица уже есть в datastore, датасет зарегистрируется успешно, но загрузка данных завершится с ошибкой.
3. Типы полей таблицы датасета теперь берутся только из типов данных используемых переменных. **NB!** Важно перепроверить, что типы данных указаны корректно для всех переменных каталога.

1.5.2 Миграция связей

Связи сущностей мигрируют в целевую архитектуру метаданных v100. Миграция связей является первым шагом процесса миграции загрузчиков на целевую архитектуру.

Для обеспечения возможности "отката" к легаси архитектуре, в релизе предусмотрена переменная `USE_NEW_ENTITY_LINK_REPO`.

1.5.3 Удаление job-ов загрузчиков

Задания загрузки данных (job-ы) удаляются для всех ранее зарегистрированных загрузчиков с загрузкой данных. Такие загрузчики перестают запускаться через эндпоинт `POST /loader/{loader_rk}/execute`.

Проверить наличие таких загрузчиков можно с помощью запроса:

```
select * from meta.loader
where kind != 'external'
```

Если есть необходимость запускать такие загрузчики, то после установки релиза их нужно будет перерегистрировать.

1.5.4 init-контейнеры

Их больше нет. Вся необходимая логика, которая была в инит контейнерах, теперь выполняется в момент старта `API`

1.5.5 Установка из whl-пакета

1. Приложение больше не предоставляет CLI команды `axiom db init` и `axiom execution init-dags`
2. Приложение больше не предоставляет эндпоинты `/api/v100/admin/migrate_metastore`, `/api/v100/admin/migrate_dags`, запускать их руками больше нет необходимости, при старте своей работы приложение самостоятельно выполнит все необходимые миграции.
3. Эндпоинт `/api/v100/admin/ref_init` также нет необходимости вызывать руками, если справочники пустые на момент старта приложение, оно само их заполнит.

1.5.6 ENVIRONMENT VARIABLES

DEPRECATED

- Все переменные, относящиеся к Airflow `AF__...`
- Все переменные, относящиеся к росбанку `RB__...`
- Все переменные, относящиеся к sentry `SENTRY__`
- Все переменные, относящиеся к git `GIT__`. Если после апгрейда приложения на версию 0.16 переменные будут еще определены, то миграции подчистят `dag_executor` и прочие файлы, которые относятся к приложению и находятся в git. В релизе 0.17 планируется окончательно убрать поддержку git.
- `API_HOST` - больше нет необходимости знать где развернуто приложение
- `API_ENV_PASSWORD` - нет необходимости
- `INIT_CONTAINERS_ENABLED` - больше нет init контейнеров
- `ROLLBACK_STAGE_DDL_PATH`
- `ROLLBACK_STAGE_CSV_PATH`
- `META__ROLLBACK_PASS`
- `ENGINE__PLUS_INFINITY`
- `ENGINE__MINUS_INFINITY`
- `ENGINE__GREENPLUM_SESSION_OPTS` - необходимо пользоваться ENGINE__CONN_OPTIONS или профилем исполнения
- `ENGINE__POSTGRES_SESSION_OPTS` - необходимо пользоваться ENGINE__CONN_OPTIONS или профилем исполнения

MOVED

- `API_TIMEOUT_KEEPALIVE` - вместо нее надо использовать `GUNICORN_CMD_ARGS`
- `UVICORN_RELOAD` - вместо нее надо использовать `GUNICORN_CMD_ARGS`
- `API_ROOT_PATH_PREFIX` -> `API__ROOT_PATH_PREFIX`
- `FILE_ENCODING` -> `S3__FILE_ENCODING`
- `MAX_CHAIN_LENGTH` -> `APP__MAX_CHAIN_LENGTH`
- `MAX_CHAINS_COUNT` -> `APP__MAX_CHAINS_COUNT`
- `MAX_DELETE_BATCH_SIZE` -> `APP__MAX_DELETE_BATCH_SIZE`
- `DEFAULT_ITEMS_LIMIT` -> `APP__DEFAULT_ITEMS_LIMIT`
- `DEFAULT_PREVIEW_LIMIT` -> `APP__DEFAULT_PREVIEW_LIMIT`
- `MAX_PREVIEW_LIMIT` -> `APP__MAX_PREVIEW_LIMIT`

NEW

- `USE_NEW_ENTITY_LINK_REPO` - выполнять сохранение и чтение связей, используя целевую архитектуру метаданных v100 (default = `True`). Если `True`, то связи сохраняются как в целевой, так и в легаси архитектуре, при этом чтение происходит только из целевых таблиц metastore. Если `False`, то используются метаданные только из легаси архитектуры.
- `EXECUTOR` - Настройки Celery
- `EXECUTOR__BROKER_URL` = `null`: адрес брокера сообщений: redis/sentinel или rabbit-mq
- `EXECUTOR__OPTIONS` = `{}`: дополнительные опции приложения Celery в формате JSON
- `EXECUTOR__ABORT_PROBE_DELAY` = `15`: частота проверки что задание отменено (в секундах)
- `APP` - настройки приложения
- `APP__NAME` = `"axiom"`: отображаемое имя приложения
- `APP__AUTO_LINKS` = `true`: создавать ли авто-связи между сущностями

1.6 Release 0.15 notes

1.6.1 Создание индексов

В связи с частичным бэкпортом оптимизации создания датасетов в POST /transform из Axiom 0.16, необходимо вручную создать недостающие индексы в metastore Axiom:

```
create index if not exists ind_dataset_db_table_name on meta.dataset (db_table_name);
create index if not exists ind_dataset_db_schema_name on meta.dataset (db_schema_name);
```

Индексы будут автоматически созданы при установке релиза Axiom 0.16 в случае их отсутствия.

1.6.2 ENVIRONMENT VARIABLES

NEW

- `AF__PYTHON_VERSION` - версия python, установленная в Airflow
- `KEYCLOAK__AUTH_REQUIRED` - вкл/выкл механизма аутентификации
- `KEYCLOAK__STRATEGY` - introspect (по умолчанию) / userinfo - эндпоинт KC для получения информации
- `KEYCLOAK__CLIENT_SECRET` - секрет для клиента KC
- `API_ROOT_PATH_PREFIX` - префикс для корневого URL приложения. Пример: axiom

1.6.3 Настройки KeyCloak

ВАЖНО: Начиная с текущего релиза роли для работы с приложением достаются только на уровне клиента. Наличие реалм-ролей (вместо клиентских) у пользователей будет игнорироваться.

ВАЖНО 2: Стратегия работы через introspect, описанная далее, работает только на Keycloak **версии 23 и выше**.

В приложении появляется возможность ходить в эндпоинт /introspect для валидации токена (включено по умолчанию).

Для корректной работы необходимо создать в KC нового **ПРИВАТНОГО** клиента. На клиенте нужно создать необходимые роли axiom. Эти клиентские роли требуется раздать пользователям в соответствии с ролевой моделью.

Секрет нового созданного клиента необходимо положить в переменную окружения `KEYCLOAK__CLIENT_SECRET`.

Для включения старого подхода с походом в эндпоинт /userinfo требуется задать значение `KEYCLOAK__STRATEGY=userinfo`

1.6.4 Airflow

1. Добавлена инструкция о создании дага, очищающего историю запусков в Airflow.
2. Протетистирована работоспособность приложения с версией airflow 2.9. При переезде на него важно корректно указать переменную среды `AF__PYTHON_VERSION`

1.6.5 Префикс для корневого URL приложения - API_ROOT_PATH_PREFIX

При необходимости использования префикса нужно внести изменения в values.yaml:

1. API_ROOT_PATH_PREFIX в configMaps (например: "axiom")
2. uriPrefix в ingress (например: "/axiom/api")

1.6.6 Префикс для корневого URL WEB приложения - ROOT_PATH_PREFIX

При необходимости использования суффикса к домену WEB интерфейса нужно внести изменения в values.yaml на примере суффикса `axiom-web`:

1. uriPrefix в ingress: /axiom-web
2. value для ROOT_PATH_PREFIX: "/axiom-web"
3. Добавить перед /index.html в livenessProbe и readinessProbe livenessProbe: httpGet: path: /axiom-web/index.html ...

   readinessProbe: httpGet: path: /axiom-web/index.html

1.7 Release 0.14 notes

1.7.1 Breaking change

Удалено поле `CSV_PATH` из response body следующих методов:

- GET api/dataset
- GET api/dataset/page

1.7.2 API

Удалены эндпоинты для работы с файлами в api/v2.

1.7.3 Инициализирующее заполнение реестра сущности

Скрипт ниже позволяет выполнить инициализирующее заполнение реестра сущности по всем зарегистрированным сущностям.

В качестве реестра сущности отбирается самый первый созданный загрузчик готовых переменных:

```
with t as (
select
    ft.entity_rk,
    l.loader_rk,
    row_number() over (
    partition by
        ft.entity_rk
    order by
        l.created_dttm asc nulls last,
        l.loader_rk asc nulls last
    ) as rn
from meta.loader l
inner join meta.loader_feature_map lfm on lfm.loader_rk = l.loader_rk
inner join meta.feature_column fc on fc.column_rk = lfm.column_rk
inner join meta.feature_table ft on ft.table_rk = fc.hist_table_rk and ft.kind = 'final'
)
update meta.v100_entity e
set options = jsonb_set(options, '{fat_registry}', cast(t.loader_rk as text)::jsonb, true)
from t
where
    t.entity_rk = e.entity_rk
    and t.rn = 1
;
```

1.7.4 ENVIRONMENT VARIABLES

Deprecated

- `LOG__FORMAT` - теперь настраивается через /sys/logging_profiles
- `LOG__MAX_LINE_LENGTH` - теперь настраивается через /sys/logging_profiles
- `LOG__KEYS_REDEFINE` - теперь настраивается через /sys/logging_profiles

NEW

- `AF__LOG_CONFIG` - конфигурация логгера для Airflow
- `APP__AUTO_LINKS` - создание автосвязей (по умолчанию = `true`)

1.8 Release 0.13 notes

1.8.1 ENVIRONMENT VARIABLES

Deprecated

- `ENGINE__SKIP_FEATURES_STATS`
- `ENGINE__SKIP_STAT_ENTITY_COUNT`
- `KEYCLOAK__OPENID_POSTFIX`
- `KEYCLOAK__USERINFO_POSTFIX`
- `KEYCLOAK__ACCESS_ROLE` - заменена на `KEYCLOAK__ACCESS_ROLES`

NEW

- `DEFAULT_ITEMS_LIMIT` - значение по-умолчанию для параметра limit в GET методах
- `KEYCLOAK__ACCESS_ROLES` - роли пользователей, задаются в виде словаря:
- ключи - "read", "transform", "load", "admin"
- значения - названия соответствующих ролей в Keycloak Старая роль "axiom" заменяется на новые 4: "read", "transform", "load", "admin".
  Эти роли необходимо добавить в Keycloak.
  По-умолчанию: `'{"read": "axiom_read", "transform": "axiom_transform", "load": "axiom_load", "admin": "axiom_admin"}'`
- `KEYCLOAK__CACHE_LIFETIME` - время жизни кеша токенов в секундах, по-умолчанию 7200
- `DEFAULT_PERMISSION_LEVEL` - значение permission по умолчанию для создаваемых объектов (`"no_access"`, `"view"`, `"read"`, `"edit"`, `"full"`), по-умолчанию: `"full"`
- `UVICORN_RELOAD` - влючение флага --reload в Uvicorn (по умолчанию False)

1.8.2 Настроки Keycloak 20+

Info

Актуально начиная с версии keycloak-20. Оригинальный тред с описанием проблемы: https://github.com/keycloak/keycloak/issues/16168

Для валидации токена приложение ходит в keycloak в эндпоинт `/userinfo`. Эндпоинт требует, чтобы в переданном токене содержался scope='openid', без него поход в эндпоинт завершается возвращает ошибкой. Этого можно добиться двумя способами:

- **Без модификации настроек keycloak**: при получении токена передавать параметр `scope=openid` (и другие, если необходимо)
- **Настройка Keycloak**, чтобы `openid` включался автоматом:

a. Manage -> Client Scopes -> Create client scope -> `openid` -> Include in token scope **[ON]**

b. Clients -> {axiom client} -> Client scopes -> Add client scope -> `openid` -> Default

1.8.3 Обновление справочников

Таблицы удаляются миграцией

- `meta.data_type`
- `meta.data_type_default`

- `meta.data_type_category`
- `meta.data_type_stats_map`

Ручные изменения

Если справочник meta.ref_type_indicator не пустой, то надо выполнить запрос в схеме meta:

```
insert into meta.ref_type_indicator (
    fs_type,
    indicator,
    author,
    created_dttm,
    updated_dttm
)
with indicators as (
    select 'COUNT_DISTINCT' as ind union all
    select 'COUNT_DISTINCT_PER_DTTM' union all
    select 'NULL_COUNT' union all
    select 'NULL_COUNT_PER_DTTM'union all
    select 'NULL_SHARE' union all
    select 'NULL_SHARE_PER_DTTM'
)
select
    tp.fs_type,
    i.ind,
    '_root',
    now(),
    now()
from
    indicators i
inner join meta.ref_fs_type tp on
    1 = 1;

commit;
```

1.8.4 Migration

В миграции 0.12 присутствует ошибка, которая может привести к невозможности удаления некоторых загрузчиков. Ошибка исправляется в миграции релиза 0.14.

При возникновении проблем в процессе эксплуатации версий 0.12 и 0.13 рекомендуется в метасторе приложения выполнить следующий скрипт, исправляющий ошибку:

```
delete from meta.feature_table_x_entity_column
where table_rk not in (select table_rk from meta.feature_table)
```

1.9 Release 0.12 notes

1.9.1 ENVIRONMENT VARIABLES

Removed

- `AF__LOADER_ID`
- `AF__DATASET_DELETE_ID`
- `LEGACY__DATA_TYPE_DB_NAME`

1.9.2 DATASOURCES

- Эндпоинты на создание и управление источниками данных переехали в /api/v100/sources
- Существующие эндпоинты /api/datasource после релиза работают только на чтение
- После установки релиза необходимо через /api/v100 обновить параметр "schemas" по всем источникам, указав список схем, доступных через этот источник

1.9.3 LOADERS

- Загрузчики, которые были на расписании, необходимо вручную повторно поставить на расписание

1.9.4 DAGs

Удалены даги `loader`, `dataset_delete`

1.9.5 Migration

В миграции присутствует ошибка, которая может привести к невозможности удаления некоторых загрузчиков. Ошибка исправляется в миграции релиза 0.14.

При возникновении проблем в процессе эксплуатации версий 0.12 и 0.13 рекомендуется в метасторе приложения выполнить следующий скрипт, исправляющий ошибку:

```
delete from meta.feature_table_x_entity_column
where table_rk not in (select table_rk from meta.feature_table)
```

1.10 Release 0.11 notes

1.10.1 Изменения data_type_rk

Изменяются значения data_type_rk для типов данных. Необходимо учесть, если используется хардкод:

Data type name	Old data_type_rk	New data_type_rk
AMOUNT	1	1796623978
CURRENCY	2	419959694
SHARE	3	1681445414
STRING	4	1601148794
BIG_INT	5	1523509284
FLAG	6	1387904535
FLOAT	7	671674020
INT	8	890642190
SMALL_INT	9	315822524
BIG_STRING	10	1430130245
DATETIME	11	48934079
DATE	12	1157021114

Для всех остальных типов данных (в т.ч. новых) значение data_type_rk должно рассчитываться по следующему алгоритму:

```
import hashlib

data_type_name = 'new_data_type'
INT_MAX = 2**31 - 1
int(hashlib.sha256(data_type_name.encode()).hexdigest(), 16) % INT_MAX
```

Справочники типов данных v100 и v2 должны изменяться синхронно и полностью соответствовать друг другу.

1.10.2 Перенос таблицы alembic version

Если таблица alembic_version хранится в схеме public, то перед установкой релиза 0.11 необходимо выполнить следующие запросы в metastore:

```
CREATE TABLE meta.alembic_version (
    version_num varchar(32) NOT NULL,
    CONSTRAINT alembic_version_pkc PRIMARY KEY (version_num)
);

INSERT INTO meta.alembic_version SELECT version_num from public.alembic_version;

DROP TABLE public.alembic_version;
```

1.10.3 Обновление справочников

Если таблица meta.ref_type_default пустая, необходимо выполнить инициализацию справочников, вызвав POST v100/admin/ref_init. Иначе необходимо учесть новые типы данных в справочнике дефолтов:

```
insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'postgresql', 'timestamp with time zone', 'DATETIME', '_root', current_timestamp, current_timestamp;

insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'greenplum', 'timestamp with time zone', 'DATETIME', '_root', current_timestamp, current_timestamp;
```

```
insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'postgresql', 'timestamp without time zone', 'DATETIME', '_root', current_timestamp, current_timestamp;

insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'greenplum', 'timestamp without time zone', 'DATETIME', '_root', current_timestamp, current_timestamp;

insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'postgresql', 'character varying', 'BIG_STRING', '_root', current_timestamp, current_timestamp;

insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'greenplum', 'character varying', 'BIG_STRING', '_root', current_timestamp, current_timestamp;

insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'postgresql', 'character', 'BIG_STRING', '_root', current_timestamp, current_timestamp;

insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'greenplum', 'character', 'BIG_STRING', '_root', current_timestamp, current_timestamp;

insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'postgresql', 'decimal', 'FLOAT', '_root', current_timestamp, current_timestamp;

insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'greenplum', 'decimal', 'FLOAT', '_root', current_timestamp, current_timestamp;

insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'postgresql', 'unknown', 'BIG_STRING', '_root', current_timestamp, current_timestamp;

insert into meta.ref_type_default (db_name, db_type, default_fs_type, author, created_dttm, updated_dttm)
select 'greenplum', 'unknown', 'BIG_STRING', '_root', current_timestamp, current_timestamp;

update meta.ref_type_default
set db_type = 'double precision' -- изменилось название
where db_type = 'double_precision' and db_name in ('postgresql', 'postgres');
```

1.10.4 Использование PVC в качестве хранилища кода DAG-ов

Настройка приложения axiom

1. Создать pvc с типом RWM

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: axiom-dags
  namespace: axiom-dev
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: nfs
  volumeMode: Filesystem
```

2. Примонтировать PVC в values.yaml приложения

```
...
  persistentVolumes:
    ...
    - name: axiom-dags
      mountPath: /var/dags
      existingVolumeClaim: true
    ...
```

3. Указать значения переменных окружения в axiom:

```
AF__DAGS_FILE_SYSTEM=pvc
(если отличается от значения по умолчанию) AF__DAGS_PVC_MOUNT_PATH=/var/dags
# PYTHONPATH в Airflow
AF__VENV__PYTHONPATH=/opt/bitnami/airflow/dags/axiom/axiom_custom:/opt/bitnami/airflow/dags/axiom/axiom_custom/axiom
# Путь до кода DAG-ов в Airflow
AF__VENV__DAGS_PATH=/opt/bitnami/airflow/dags/axiom
# Путь до кода tasks в Airflow
AF__VENV__TASKS_PATH=/opt/bitnami/airflow/dags/axiom/axiom_custom/axiom/custom_tasks
```

Настройка Airflow

1. Удалить секцию service.git
2. Примонтировать PVC в values.yaml Airflow

```
...
  extraVolumes:
    ...
    - name: dags
      persistentVolumeClaim:
        claimName: axiom-dags
    ...
    extraVolumeMounts:
    ...
    - name: dags
      mountPath: /opt/bitnami/airflow/dags
      readOnly: false
    ...
```

3. Указать значение переменной окружения

```
PYTHONPATH=/opt/bitnami/airflow/dags**
AIRFLOW_CONFIG_LOCATION=/opt/bitnami/airflow/dags/axiom/airflow_config.ini
```

1.10.5 Настройка пула для дагов в Airflow

Для настройки пула в Airflow под даги приложения необходимо:

1. Создать пул с именем в Airflow (Airflow UI: Admin -> Pools -> +)
2. Указать имя созданного пула в переменную окружения в чарте Airflow: AXIOM_DAGS_POOL

1.10.6 Airflow Environment Variables (**NEW**)

```
SSL_VERIFICATION = false    # Требуется ли ssl верификация для приложения. По умолчанию False.
                            # Возможные значения: True/False/cert_path
AXIOM_DAGS_POOL = "default_pool"    # Имя пула для тасок axiom в Airflow
```

1.10.7 API Environment Variables (**CHANGES**)

Унифицированы имена всех переменных окружения, в частности переменные сгруппированы по секциям: Metastore, Datastore, Engine, S3, Airflow, Log, Sentry, Keycloak, Git, Rosbank (см. таблицу ниже) Все переменные типа bool и int в values файле больше нет необходимости обрамлять в кавычки. Переменные типа json должны обрамляться одинарными кавычками, например: LOG_FIELDS_REDEFINE='{"traceback": "aboba"}'.

Важно! Старые переменные, отвечающие за размер/таймаут пула - сгруппированы в DB__CONN_OPTIONS. Расширенная инструкция по конфигурации доступна в руководстве Connecting Datastore

NEW

```
AXIOM__CONFIG_PREFIX = ''  # устанавливает префикс для всех переменных окружения
                           # по-умолчанию - не установлен

S3__SSL_VERIFICATION = true             # Требуется ли ssl верификация. По умолчанию True
                                        # Возможные значения: True/False/cert_path

AF__SSL_VERIFICATION = true             # Требуется ли ssl верификация. По умолчанию True
                                        # Возможные значения: True/False/cert_path

KEYCLOAK__SSL_VERIFICATION = true       # Требуется ли ssl верификация. По умолчанию True
                                        # Возможные значения: True/False/cert_path

RB__DATASOURCE_SSL_VERIFICATION = true  # Требуется ли ssl верификация. По умолчанию True
                                        # Возможные значения: True/False/cert_path

LOG_FIELDS_REDEFINE = '{}'  # Переопределение имен полей в json-логе.
            # Например: '{"traceback": "aboba"}' переопределит дефолтное поле traceback как aboba

AF__URL = ""                  # URL до Airflow (без логина и пароля)
AF__USER = "user"             # user для Airflow
AF__PASSWORD = "password"     # пароль для Airflow
AF__DAGS_FILE_SYSTEM = "git"  # Тип хранилища для кода DAG-ов Airflow (git/pvc)
AF__DAGS_PVC_MOUNT_PATH = "/var/dags"  # Путь до директории к примонтированному pvc

ENGINE__DROP_OBJECTS_BATCH_SIZE = 100  # Размер батча для метода drop_objects() при datastore reset
```

Metastore

Old name	NEW name	Comment
META_DB_URL	META__URL	...
-	META__CONN_OPTIONS	new!
META_POOL_SIZE	(removed)	moved to META__CONN_OPTIONS
META_MAX_OVERFLOW	(removed)	moved to META__CONN_OPTIONS
META_POOL_TIMEOUT	(removed)	moved to META__CONN_OPTIONS
META_SCHEMA_NM	META__SCHEMA	...
-	META__WORK_SCHEMA	теперь отдельная переменная для meta
ROLLBACK_PASS	META__ROLLBACK_PASS	...
DB_ROLLBACK_INITIAL_CSV_PATH_META	META__ROLLBACK_CSV_PATH	...

Datastore

Old name	NEW name	Comment
DB_URL	DB__URL	...
-	DB__CONN_OPTIONS	new!
STORE_POOL_SIZE	(removed)	moved to DB__CONN_OPTIONS
STORE_MAX_OVERFLOW	(removed)	moved to DB__CONN_OPTIONS
STORE_POOL_TIMEOUT	(removed)	moved to DB__CONN_OPTIONS
STAGE_SCHEMA_NM	DB__STAGE_SCHEMA	...
STORE_SCHEMA_NM	DB__STORE_SCHEMA	...
DATASET_SCHEMA_NM	DB__DATASET_SCHEMA	...
WORK_SCHEMA_NM	DB__WORK_SCHEMA	...
SANDBOX_SCHEMA_NM	(removed)	legacy
PLUS_INFINITY	DB__PLUS_INFINITY	...
MINUS_INFINITY	DB__MINUS_INFINITY	...
DB_OBJECT_NAME_LENGTH_LIMIT	DB__MAX_NAME_LENGTH	...
		...
DB_ROLLBACK_INITIAL_SQL_PATH	DB__ROLLBACK_STAGE_DDL_PATH	legacy
DB_ROLLBACK_INITIAL_CSV_PATH_STAGE	DB__ROLLBACK_STAGE_CSV_PATH	legacy

FS Engine

Old name	NEW name	Comment
FS_ALCHEMY_TEMP_OBJECTS_AS	ENGINE__TEMP_OBJECTS_AS	...
FS_ALCHEMY_ASYNC_FACTOR	ENGINE__ASYNC_FACTOR	...
AXIOM_AUTO_ANALYZE	ENGINE__AUTO_ANALYZE	...
AXIOM_AUTO_INDEX	ENGINE__AUTO_INDEX	...
AXIOM_DISTRIBUTED_BY_OVERRIDE	ENGINE__DEFAULT_STORAGE_KEY	...
AXIOM_SKIP_FEATURES_STATS	ENGINE__SKIP_FEATURES_STATS	...
AXIOM_SKIP_STAT_ENTITY_COUNT	ENGINE__SKIP_STAT_ENTITY_COUNT	...
		...
GREENPLUM_SESSION_OPTS	ENGINE__GREENPLUM_SESSION_OPTS	legacy, не работает, moved to ref_execution_profile
POSTGRES_SESSION_OPTS	ENGINE__POSTGRES_SESSION_OPTS	legacy, не работает, moved to ref_execution_profile

S3 Storage

Old name	NEW name	Comment
S3_ENABLED	S3__ENABLED	...
S3_URL	S3__URL	...
S3_BUCKET	S3__BUCKET	...
S3_ACCESS_KEY	S3__ACCESS_KEY	...
S3_SECRET_ACCESS_KEY	S3__SECRET_ACCESS_KEY	...
dataset_sql_path	S3__DATASET_SQL_PATH	...
dataset_csv_path	S3__DATASET_CSV_PATH	...
dataset_file_limit_size_mb	S3__MAX_FILE_SIZE_MB	...
dataset_buff_limit_size_mb	S3__BUFFER_SIZE_MB	...
dataset_describe_timeout	S3__BUFFER_MALLOC_TIMEOUT	...
dataframe_paralel_processing_count	(removed)	never used
s3_url_expire_interval	S3__URL_EXPIRE_INTERVAL	...

Airflow

Old name	NEW name	Comment
AIRFLOW_URL	AF__URL	...
AIRFLOW_DB_CONN	(removed)	unused
pipeline_id	(removed)	unused
constructor_id	(removed)	unused
dataset_csv_id	(removed)	unused
dataset_split_id	(removed)	unused
dags_factory_id	(removed)	unused
loader_id	AF__LOADER_ID	...
dataset_delete_id	AF__DATASET_DELETE_ID	...
dag_executor_id	AF__DAG_EXECUTOR_ID	...
AIRFLOW_EXECUTION_MODE	AF__EXECUTION_MODE	...
AXIOM_DAGS_PATH	AF__DAG_SRC_PATH	...
DAG_TEMPLATE_LOCATION	AF__DAG_TEMPLATE_PATH	...
DAG_POST_TIMEOUT	AF__DAG_POST_TIMEOUT	...
DAG_TRIGGER_TIMEOUT	AF__DAG_TRIGGER_TIMEOUT	...
DAG_TRIGGER_RETRIES	AF__DAG_TRIGGER_RETRIES	...
CUSTOM_VENV_LOCATION	AF__VENV__PATH	...
CUSTOM_PYTHONPATH	AF__VENV__PYTHONPATH	...
SCRIPT_LOCATION	AF__VENV__DAGS_PATH	...
CUSTOM_SCRIPT_LOCATION	AF__VENV__TASKS_PATH	...
CUSTOM_K8S_PYTHONPATH	AF__K8S__PYTHONPATH	...
K8S_TASKS_IMAGE	AF__K8S__IMAGE	...
K8S_TASKS_PATH	AF__K8S__TASKS_PATH	...
K8S_NAMESPACE	AF__K8S__NAMESPACE	...
K8S_REGCRED	AF__K8S__REGCRED	...
K8S_CONFIGMAP	AF__K8S__CONFIGMAP	...
K8S_SECRET	AF__K8S__SECRET	...
TASK_REQUEST_MEMORY_MB	AF__K8S__REQUEST_MEM	...
TASK_LIMIT_MEMORY_MB	AF__K8S__LIMIT_MEM	...
TASK_REQUEST_CPU_mCore	AF__K8S__REQUEST_CPU	...
TASK_LIMIT_CPU_mCore	AF__K8S__LIMIT_CPU	...
AXIOM_DAGS_PATH	(removed)	unused

LOG

Old name	NEW name	Comment
log_prefix	LOG__PREFIX	...
log_type	LOG__FORMAT	...
log_level	LOG__LEVEL	...
log_sleep	(removed)	legacy
MAX_LOG_LEN	LOG__MAX_LINE_LENGTH	...
LOG_FIELDS_REDEFINE	LOG__KEYS_REDEFINE	...
LOG_PATH	LOG__FILE_NAME	...

Sentry

Old name	NEW name	Comment
SENTRY_ENABLED	SENTRY__ENABLED	...
SENTRY_DSN	SENTRY__DSN	...

Keycloak

Old name	NEW name	Comment
KEYCLOAK_URL	KEYCLOAK__URL	...
OPENID_POSTFIX	KEYCLOAK__OPENID_POSTFIX	...
USERINFO_POSTFIX	KEYCLOAK__USERINFO_POSTFIX	...
KEYCLOAK_ACCESS_ROLE	KEYCLOAK__ACCESS_ROLE	...
CLIENT_ID	KEYCLOAK__CLIENT_ID	...
VERIFY_TOKEN	KEYCLOAK__VERIFY_TOKEN	...

Git

Old name	NEW name	Comment
GIT_ENABLED	GIT__ENABLED	...
GIT_TYPE	GIT__AUTH_TYPE	...
GIT_URL	GIT__URL	...
GIT_USERNAME	GIT__USER_NAME	...
GIT_USER_EMAIL	GIT__USER_EMAIL	...
GIT_REPO_ID	GIT__REPO_ID	...
GIT_BRANCH	GIT__BRANCH	...
GIT_LOGIN	GIT__LOGIN	...
GIT_PASSWORD	GIT__PASSWORD	...
GIT_SSH_KEY_PATH	GIT__SSH_KEY_PATH	...
GIT_SSH_KEY_PATH_INIT	GIT__SSH_KEY_PATH_INIT	...
GIT_SSH_KEY_PATH_DAG	GIT__SSH_KEY_PATH_DAG	...
GIT_SSH_KEY_NAME	GIT__SSH_KEY_NAME	...
GIT_KEY_SECRET_NAME	GIT__SSH_KEY_SECRET	...

Rosbank

Old name	NEW name	Comment
DATASOURCE_DAGS_ENABLED	RB__DATASOURCE_DAGS_ENABLED	...
ROWNUM_ENABLED	RB__ROWNUM_ENABLED	...
LOADER_ROWNUM_ENTITY_RK	RB__ROWNUM_ENTITY_RK	...
CALCULATING_DENSE_DISTRIBUTION_PLOT	RB__DENSE_PLOT_ENABLED	...
DISTRIBUTION_PLOT_RESULT_FILENAME	RB__DENSE_PLOT_FILENAME	...
DIST_PLOT_CALCULATION_TIMEOUT	(removed)	unused

1.11 Release 0.10 notes

1.11.1 Обновление справочников

1. Если таблица meta.ref_fs_type пустая, необходимо выполнить инициализацию справочников, вызвав POST v100/admin/ref_init, иначе выполнить обновление поля db_name:

```
update meta.ref_db_type
set db_name = 'postgresql'
where db_name = 'postgres';

update meta.ref_type_default
set db_name = 'postgresql'
where db_name = 'postgres';
```

2. Если на момент установки релиза используются кампании с пользовательскими формулами, в которых присутствует "cast(... as interval)", необходимо учесть тип данных "Интервал" в справочнике типов данных:

```
insert into meta.ref_fs_type(fs_type, description, py_type, author, created_dttm)
select 'INTERVAL', 'Интервал', 'str', '_root', current_timestamp;

insert into meta.ref_db_type(db_name, fs_type, db_type, author, created_dttm)
select 'postgresql', 'INTERVAL', 'interval', '_root', current_timestamp;

update meta.ref_type_default
set default_fs_type = 'INTERVAL'
where db_type = 'interval';
```

1.11.2 New settings

Environment Variables

```
AXIOM_SKIP_STAT_ENTITY_COUNT = '"true"'  # Не считать статистику ENTITY_COUNT
MAX_LOG_LEN = '"16000"'     # Максимальная длина сообщения с логом
LOG_FIELDS_REDEFINE = '""'  # Переопределение имен полей в json-логе.
         # Например: '''{"traceback": "aboba"}''' переопределит дефолтное поле traceback как aboba
MAX_CHAIN_LENGTH = 4        # Максимальная длина цепочки связей в методе Get Entity Link Chain
                            # Значение выше 6 может привести к зависанию приложения
                            # Может быть изменено в параметрах соответствующего запроса аргументом max_chain_length
                            # В запросе крайне не рекомендуется ставить значение
                            # выше 6 без назначенного параметра filter_entity_rk
MAX_CHAINS_COUNT = 5000   # Максимальное количество цепочек, возвращаемых в методе Get Entity Link Chain
```

Airflow dependencies

```
Поддерживаемая версия: 2.5.3.
Версия чарта bitnami: 14.0.17.
```

1.12 Release 0.9 notes

1.12.1 New environment variables

Secrets

```
-
```

Variables

```
FS_ALCHEMY_TEMP_OBJECTS_AS = "table"    # Создавать промежуточные объекты как table/view
FS_ALCHEMY_ASYNC_FACTOR = '"4"'         # Фактор асинхронности в fs_alchemy
AXIOM_AUTO_ANALYZE = '"true"'           # Автоматически выполнять ANALYZE таблиц после их создания
AXIOM_AUTO_INDEX = '"false"'            # Выполнять CREATE INDEX по бизнес-ключу таблиц после их создания
AXIOM_DISTRIBUTED_BY_OVERRIDE = ''      # Глобальный override на ключ дистрибуции таблиц
                                        # например 'ID' или 'ID,NAME'
AXIOM_SKIP_FEATURES_STATS = '"true"'      # Пропускать расчет бизнес-статистик по колонкам
META_POOL_SIZE = '"50"'                 # Размер пула подключений для meta
META_MAX_OVERFLOW = '"0"'               # Количество подключений в overflow meta
META_POOL_TIMEOUT = '"60"'              # Таймаут ожидания подключения к meta
STORE_POOL_SIZE = '"50"'                # Размер пула подключений для datastore
STORE_MAX_OVERFLOW = '"0"'              # Количество подключений в overflow datastore
STORE_POOL_TIMEOUT = '"60"'             # Таймаут ожидания подключения к meta
LOG_PATH = '/var/log/axiom/axiom.log'   # Путь до файла с логом приложения
```

Airflow dependencies

```
Поддерживаемая версия: 2.5.3.
Версия чарта bitnami: 14.0.17.
```

1.13 Release 0.7.3 notes

1.13.1 New environment variables

Secrets

```
-
```

Variables

```
TEMP_OBJECTS_AS = "view"     # Создавать промежуточные объекты как table/view
META_POOL_SIZE = '"50"'      # Размер пула подключений для meta
META_MAX_OVERFLOW = '"0"'    # Количество подключений в overflow meta
META_POOL_TIMEOUT = '"60"'   # Таймаут ожидания подключения к meta
STORE_POOL_SIZE = '"50"'     # Размер пула подключений для datastore
STORE_MAX_OVERFLOW = '"0"'   # Количество подключений в overflow datastore
STORE_POOL_TIMEOUT = '"60"'  # Таймаут ожидания подключения к meta
AXIOM_AUTO_ANALYZE = '"1"'   # 0/1 - надо ли запускать analyze на временные объекты после create
```

Airflow dependencies

```
Поддерживаемая версия: 2.3.2.
```

2. Установка

Состав поставки:

- docker image + helm чарт `axiom` - все компоненты области **Application** разворачиваются из него, отличается только команда, которой запускается компонент
- docker image + helm чарт `axiom-web` - web интерфейс

Зависимые компоненты типа *Redis*, *PostgreSQL* и прочее не являются частью приложения и разворачиваются из публичных образов, например bitnami. В поставляемом helm чарте этих зависимостей нет.

Данная инструкция написана исходя из предположения, что установка будет производиться в существующий kubernetes кластер, но описаные шаги можно адаптировать под аналогичные технологии управления контейнерами.

2.1 TL;DR

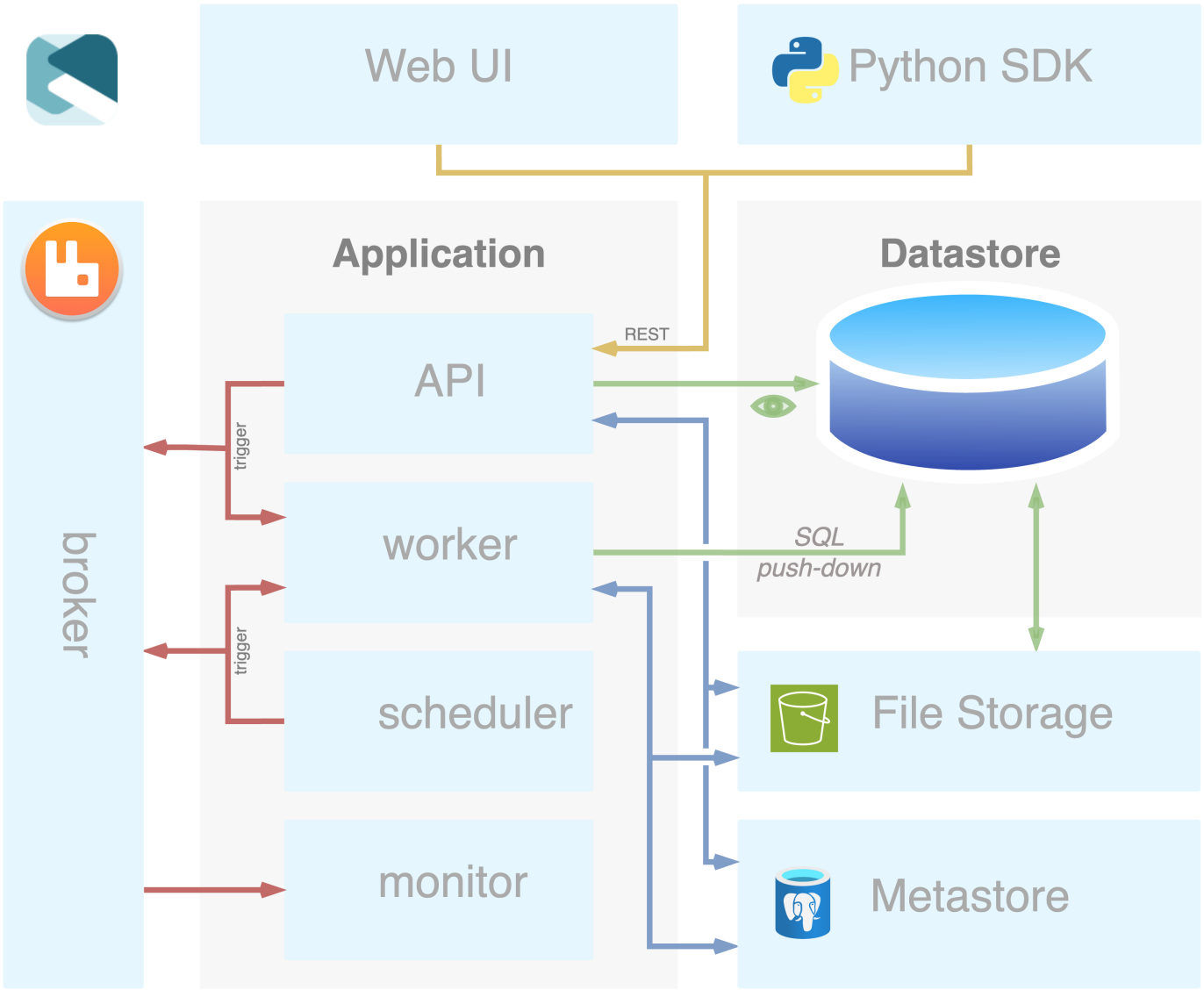
```
helm install metastore oci://registry-1.docker.io/bitnamicharts/postgresql
helm install broker oci://registry-1.docker.io/bitnamicharts/rabbitmq
helm install my-release oci://harbor.k8s.datasapience.ru/datasapience-charts/axiom
```

Команды выше установят последний релиз приложения со всеми зависимыми компонентами:

- `my-release` - API
- `my-release-executor` - worker, scheduler, monitor
- `broker` - брокер RabbitMQ (`user`:`password`)
- `metastore` - СУБД PostgreSQL под метаданные и под данные (`postgres`:`postgres`)

Такая инсталляция не зависит от внешних компонент и предоставляет достаточно адекватную конфигурацию для dev среды.

2.2 Архитектура



- **Metastore** - база данных PostgreSQL, где приложение хранит свои метаданные, необходимые для работы
- **Datastore** - одна из поддерживаемых СУБД для обработки бизнес-данных
- **File Storage** - S3 хранилище для обмена файлами с пользователем системы
- **Broker** - очередь сообщений на базе RabbitMQ или Redis
- **Application** - само приложение. Состоит из нескольких компонентов, общающихся между собой через **broker**:
- **API** - FastAPI приложение. Предоставляет доступ к функциям системы через REST API
- **worker** - Обработчик заданий на базе Celery. Управляет трансформациями, спускаемыми в **Datastore**
- **scheduler** (опционально) - планировщик задач на расписании на базе Celery Beat
- **monitor** (опционально) - система мониторинга Celery на базе Flower
- **Web UI** - web-интерфейс приложения на базе React. Работает с приложением через REST API
- **Python SDK** - python библиотека-wrapper для API приложения

2.3 Подготовка среды

1. **k8s** кластер.

2. утилита **helm** версии 3 или выше.
3. PostgreSQL под **Metastore**. Для работы приложению необходима СУБД PostgreSQL 12+ под метаданные. Развернуть ее можно любым удобным способом, например, воспользовавшись опциями от bitnami. В **Metastore** должна быть предварительно создана схема под метаданные. Учетная запись для доступа к базе данных должна иметь достаточные права для создания и удаления любых объектов в этой схеме.
4. Одна из поддерживаемых СУБД для **Datastore**. В **Datastore** должны быть предварительно созданы 3 схемы: под выходные датасеты, под загрузчики и под временные объекты трансформаций. Учетная запись для доступа к базе данных должна иметь достаточные права для создания и удаления любых объектов в этих схемах.

2.4 Установка Axiom API

Скопировать чарт и `values.yaml`

```
helm pull oci://harbor.k8s.datasapience.ru/datasapience-charts/axiom

helm show values axiom-*.tgz > values.yaml

ls
  axiom-X.Y.Z.tgz   values.yaml
```

Настройки приложения прописаны в формате кросс-ссылок по service name и хранятся в следующих конфигмапах и секретах:

- `{{ .Release.Name }}-configmap` - Основные несекретные настройки приложения
- `{{ .Release.Name }}-secret` - Настройки подключений к базам данных, minio и прочие параметры, которые могут содержать пароли

Отредактировать файл `values.yaml`:

1. Указать корректные значения для образа приложения

```
# values.yaml

global:
  image:
    repository: harbor.k8s.datasapience.ru/datasapience-registry/kolmogorov/axiom/axiom
    tag: latest  # версия образа из harbor
                 # соответствует номеру релиза приложения
```

2. Указать коррекные значения для переменных:

- `META__URL`: строка подключения к **Metastore** в формате `postgresql://<user>:<pass>@<db-host>:<db-port>/<database>`
- `META__SCHEMA` (default `"meta"`): схема в БД под метаданные приложения.

3. *(optional)* Если указать также следующие переменные, то при старте приложения в метаданных автоматически зарегистрируется источник данных по-умолчанию:

- `DB__URL`: строка подключения к **Datastore** в корректном формате
- `DB__CONN_OPTIONS`: дополнительные настройки
- `DB__WORK_SCHEMA`: схема в БД под временные объекты трансформаций
- `DB__STORE_SCHEMA`: схема в БД под основную витрину с данными
- `DB__DATASET_SCHEMA` (default `"dataset"`): схема в БД под выходные датасеты

4. Указать прочие желаемые переменые приложения

Выполнить установку приложения командой:

```
helm upgrade my-release axiom-*.tgz --install --dependency-update --atomic \
     --debug \
     --namespace ${NAMESPACE} \
     --values values.yaml
```

2.5 Доступ к API извне кластера через ingress

Для включения ингрессов необходимо в файле `values.yaml` изменить следующие значения:

```yaml
# values.yaml

global:
  # домен на котором будет работать ingress
  baseDomain: example.com

service:
  ingress:
    enabled: true
    # Корректно заполнять annotations и tls для работы по HTTPS
    annotations:
      cert-manager.io/cluster-issuer: ...
    tls:
      secretName: ...

# опционально
# для доступа к интефейсу Flower
executor:
  ingress:
    enabled: true
    annotations:
      cert-manager.io/cluster-issuer: ...
    tls:
      secretName: ...
```

2.6 Аутентификация через Keycloak

По-умолчанию аутентификация в приложении отключена. Для ее включения необходим сервис аутентификации и авторизации **Keycloak**. Развернуть его можно любым удобным способом, например, воспользовавшись опциями от bitnami. Ниже описана минимальная настройка интеграции с keycloak, но есть и другие опции.

В `values.yaml` указать следующие переменные внутри `{{ .Release.Name }}-secret`:

- `KEYCLOAK__AUTH_REQUIRED`: установить `true`
- `KEYCLOAK__URL`: адрес реалма Keycloak
- `KEYCLOAK__CLIENT_ID`: имя приватного клиента, в котором заведены роли доступа
- `KEYCLOAK__CLIENT_SECRET`: секрет для доступа к клиенту

2.7 Файловое хранилище S3

По-умолчанию функционал загрузки файлов приложение не включен. Для его включения необходимо S3-совместимое хранилище. Развернуть его можно любым удобным способом, например, воспользовавшись опциями от bitnami.

Затем указать следующие переменные внутри `{{ .Release.Name }}-secret`:

- `S3__ENABLED`: Установить `true`
- `S3__URL`: адрес API хранилища
- `S3__BUCKET`: имя бакета в S3 хранилище
- `S3__ACCESS_KEY`: S3 access key
- `S3__SECRET_ACCESS_KEY`: S3 secret access key

2.8 Сбор телеметрии по стандарту OpenTelemetry

Если системе развернут сборщик телеметрии, например Jaeger приложение может отправлять телеметрию в него по протоколу OTLP/HTTP. Для этого необходимо установить:

- переменную приложения `TRACING__ENABLED=true`
- прочие стандартные `OTEL_*` переменные для OTLP/HTTP экспортера.

Для минимальной конфигурации экспортера необходимо установить переменную `OTEL_EXPORTER_OTLP_ENDPOINT=http://<jaeger-host>:4318`. Протокол OTLP/gPRC не поддерживается, только OTLP/HTTP.

2.9 Альтернативные конфигурации брокера сообщений

Кроме RabbitMQ в качестве брокера можно использовать Redis или Redis Sentinel, примеры заполнения URL приведены в `values.yaml` в поставляемом чарте.

Приложение тестировалось только с RabbitMQ и Redis, но должно работать с любой amqp совместимой очередью сообщений.

Чтобы подключить приложение к внешнему брокеру необходимо отредактировать `values.yaml` и указать следующие переменные внутри `{{ .Release.Name }}-secret`:

- `EXECUTOR__BROKER_URL`: адрес брокера в формате `redis://...` если redis или `amqp://...` если rabbitmq

2.10 Хранилище секретов Vault

Чтобы не указывать значения секретов напрямую в чарте можно воспользоваться хранилищем секретов **Vault**. Пример создания секретов с использованием **Vault** можно посмотреть в `chart/values.vault.yaml`.

2.11 Масштабирование

Поставляемый чарт описывает запуск основных компонент приложения в двух контейнерах. При этом с настройками по умолчанию, уже обеспечивается некоторый параллелизм:

- `API` запускае количество тредов, равное `$WEB_CONCURRENCY`
- `worker` запускает количество тредов, равное количеству `cpu` контейнера
- `scheduler` запускается в единственном экземпляре
- `monitor` запускается в единственном экземпляре

Для небольшой инсталляции такой конфигурации должно быть достаточно, но если есть желание запускать компоненты системы в несколько реплик, то необходимо подправить `values.yaml`, подправив значения `replicas` или `hpa` в секциях `service` и `executor`

Для брокера RabbitMQ есть родная инструкция для настройки конфигурации Disaster Recovery и High Availability.

2.12 Установка из whl-пакетов

1. Получить доступ к `pypi` репозиторию (например на базе `nexus`) в который выгружены whl-пакеты:
   - `axiom`: основной пакет, содержащий логику api/worker/scheduler/monitor
   - `fs-core` (зависимость): бизнес-логика
   - `fs-alchemy` (зависимость): кодогенерация SQL
2. Запомнить или экспортировать переменные:
   - `NEXUS_USER`: имя пользователя или `__token__`
   - `NEXUS_PASS`: пароль или токен
   - `NEXUS_REPO`: адрес репозитория без ведущего `https://` в формате `.../pypi/simple` например `git.angara.cloud/api/v4/projects/535/packages/pypi/simple`
3. Установить основной пакет приложения командой:

```
export AXIOM_VERSION=... # устанавливаемая версия, например 0.16.0
python -m pip install -U axiom==$AXIOM_VERSION \
```

```
      --index-url https://${NEXUS_USER}:${NEXUS_PASS}@${NEXUS_REPO} \
      --extra-index-url https://${NEXUS_USER}:${NEXUS_PASS}@${NEXUS_REPO}
```

Запуск приложения осуществляется командой (с точностью до запускаемых компонент):

```
axiom start api worker scheduler monitor --run-migrations
```

Доступные опции запуска можно посмотреть командой:

```
axiom --help
```

2.13 Axiom Web

Скопировать чарт и `values.yaml`

```
helm pull oci://harbor.k8s.datasapience.ru/datasapience-charts/axiom-web

helm show values axiom-*.tgz > values.web.yaml

ls
  axiom-web-X.Y.Z.tgz   values.web.yaml
```

Отредакировать в `values.yaml`:

1. Указать настройки образа и ingress

```
service:
  image:
    repository: harbor.k8s.datasapience.ru/datasapience-registry/kolmogorov/axiom/web
    tag: latest  # версия образа из harbor
                 # соответствует номеру релиза приложения
  ingress:
    baseDomain: example.com
    host: "{{ .Release.Name }}"  # можно указать в явном виде
  extra_vars:
    - name: FS_API
      value: /api  # Если API на том же хосте. Полный путь если на другом
```

2. Указать прочие желаемые переменные окружения

Выполнить деплой приложения командой:

```
helm upgrade my-release-web axiom-web*.tgz --install --dependency-update --atomic \
     --debug \
     --namespace ${NAMESPACE} \
     --values values.web.yaml
```

2.13.1 Использование суффикса

При необходимости использования суффикса к домену WEB интерфейса нужно внести изменения в values.yaml на примере суффикса `axiom-web`:

1. uriPrefix в ingress: /axiom-web
2. value для ROOT_PATH_PREFIX: "/axiom-web"
3. Добавить перед /index.html в livenessProbe и readinessProbe

```
livenessProbe:
  httpGet:
    path: /axiom-web/index.html
...

readinessProbe:
  httpGet:
    path: /axiom-web/index.html
```

3. Configuration

3.1 Настройки

Раздел описывает доступные переменные окружения для образов `axiom` и `axiom-web`.Установка приложения описана в инструкции по установке

3.1.1 Приложение Axiom (API / worker)

Значения по-умолчнию указаны через `=` после имени переменной. Значения, указанные как `= secret:...` задаются через хранилище секретов `Vault`.

- `ENV_NM` = `"prod"` : имя среды
- `DEFAULT_PERMISSION_LEVEL` = `"full"` : уровень доступа по-умолчанию
- `APP` - настройки приложения
- `APP__NAME` = `"axiom"` : отображаемое имя приложения
- `APP__AUTO_LINKS` = `true` : создавать ли авто-связи между сущностями
- `APP__MAX_DELETE_BATCH_SIZE` = `50` : максимальное количество датасетов при удаление через `/dataset/delete`
- `APP__DEFAULT_ITEMS_LIMIT` = `100` : количество элементов в ответе пагинированного эндпоинта по умолчанию
- `APP__MAX_ITEMS_LIMIT` = `5000` : максимальное количество элементов в ответе пагинированного эндпоинта
- `APP__DEFAULT_PREVIEW_LIMIT` = `100` : количество строк, возвращаемое методами `/dataset/preview` и `/v100/sources/{source_rk}/preview` по-умолчанию
- `APP__MAX_PREVIEW_LIMIT` = `100` : максимально допустимое для запроса количество строк в методах `/dataset/preview/` и `/v100/sources/{source_rk}/preview`
- `APP__MAX_CHAIN_LENGTH` = `4` : максимальная длина цепочки связей
- `APP__MAX_CHAINS_COUNT` = `5000` : максимальное количество цепочек связей в ответе
- `APP__DEFAULT_SOURCE_RK` = `-1` : суррогатный ключ источника по умолчанию для запуска заданий
- `APP__DEFAULT_ASYNC_DELETE` = `false` : удалять датасеты в фоновом процессе по умолчанию
- `APP__STAT_DISTINCT_LIMIT` = `1000` : максимальное количество возвращаемых значений в статистике `DISTINCT`
- `APP__LOCK_TIMEOUT` = `7200` - максимальное время блокировки (в секундах)
- `API` - настройки web-сервиса
- `API__ROOT_PATH_PREFIX` = `null` : корневой префикс для всех эндпоинтов
- `EXECUTOR` - Настройки Celery
- `EXECUTOR__BROKER_URL` = `secret:executor-broker-url` : адрес брокера сообщений redis/sentinel или rabbit-mq
- `EXECUTOR__OPTIONS` = `{}` : дополнительные опции приложения Celery в формате JSON
- `TASK_RETRY_COUNT` = `5` : количество попыток повторного запуска задачи после ошибки
- `TASK_RETRY_DELAY` = `10` : задержка между попытками
- `DELETE_DATASET_BATCH_SIZE` = `null` : Размер пачки для разового удаления
- `DELETE_DATASET_DELAY` = `0` : Задержка после удаления пачки
- `META` - настройки БД с метаданными приложения
- `META__URL` = `secret:meta-db-url` : строка подключения до postgre с метаданными
- `META__CONN_OPTIONS` = `{}` : дополнительные опции движка SQLAlchemy в формате JSON
- `META__SCHEMA` = `"meta"` : схема в БД, где приложение хранит метаданные
- `DB` - настройки БД, где находится витрина с данными
- `DB__URL` = `secret:db-url` : строка подключения до движка с витриной (*deprecated*)
- `DB__CONN_OPTIONS` = `{}` : дополнительные опции движка SQLAlchemy в формате JSON (*deprecated*)
- `DB__STAGE_SCHEMA` = `"stage"` : схема в БД, используемая как *staging area* (*deprecated*)
- `DB__STORE_SCHEMA` = `"store"` : схема в БД, используемая загрузчиками (*deprecated*)
- `DB__DATASET_SCHEMA` = `"dataset"` : схема в БД, используемая для сохранения датасетов в API V2
- `DB__WORK_SCHEMA` = `"work"` : схема в БД, используемая для создания временных объектов (*deprecated*)
- `DB__MAX_NAME_LENGTH` = `63` : максимальное допустимая длина имени объектов в БД

- `ENGINE` - настройки поведения конструктора
- `ENGINE__TEMP_OBJECTS_DROP` = `true` : удалять ли временные объекты после завершения джоба
- `ENGINE__TEMP_OBJECTS_AS` = `"cte"` : как создавать временные объекты: table/view/cte/subquery
- `ENGINE__FINAL_OBJECTS_AS` = `"table"` : способ создания целевых таблиц: table/ctas. Если `table`, то таблица создается через `create table + insert`, иначе через `create table as select`
- `ENGINE__ASYNC_FACTOR` = `4` : максимальное количество одновременных SQL запросов
- `ENGINE__AUTO_ANALYZE` = `true` : запускать ли сбор статистики в БД над созданными объектами
- `ENGINE__AUTO_INDEX` = `false` : создавать ли индекс на таблицах по ключу сущности
- `ENGINE__DEFAULT_STORAGE_KEY` = `null` : ключ распределения по-умолчанию, если движок его поддерживает (*deprecated*)
- `LOG` - настройки логирования
- `LOG__PREFIX` = `"axiom-log"` : префикс строчки в stout, для текстового формата логов
- `LOG__LEVEL` = `"INFO"` : уровень логирования
- `LOG__FILE_NAME` = `"/var/log/axiom/axiom.log.jsonl"` : имя файла для логов
- `TELEMETRY` - настройки OpenTelemetry
- `TELEMETRY__ENABLED` = `false` : Включить отправку OpenTelemetry данных по протоколу OTLP
- `TELEMETRY__SERVICE_NAME` = `$APP__NAME::$SUBSYSTEM` : Имя сервиса с которым будут сохраняться данные телеметрии в сборщике
- `TELEMETRY__EXCLUDE_URLS` = `['/health/liveness', '/health/readiness', 'favicon.ico']` : Список URL, по которых не надо собирать телеметрию
- `TELEMETRY__EVENT_LOGGERS` = `['$APP__NAME', 'fs-core', 'fs-alchemy']` : Список логгеров, записи которых следует добавлять в качестве события в span
- `KEYCLOAK` - настройки аутентификации и авторизации
- `KEYCLOAK__AUTH_REQUIRED` = `false` : требуется ли авторизация
- `KEYCLOAK__URL` = `secret:keycloak-url` : адрес сервиса Keycloak
- `KEYCLOAK__SSL_VERIFICATION` = `true` : использовать ли ssl верификацию при подключении к keycloak: true/false/<путь до сертификата>
- `KEYCLOAK__STRATEGY` = `"introspect"` : стратегия, используемая для авторизации: userinfo/introspect
- `KEYCLOAK__ACCESS_ROLES` = `{"read":"axiom_read","load":"axiom_load","transform":"axiom_transform","admin":"axiom_admin"}` : имена ролей, используемых для ролевого доступа
- `KEYCLOAK__CLIENT_ID` = `"axiom_client"` : имя клиента, в котором заведны роли доступа
- `KEYCLOAK__CLIENT_SECRET` = `secret:keycloak-client-secret` : секрет для доступа к клиенту, если выбрана стратегия introspect
- `KEYCLOAK__CACHE_LIFETIME` = `7200` : время жизни кеша с токенами в секундах
- `KEYCLOAK__SIGN_ALGORITHM` = `"RS256"` : алгоритм подписи токена, используемый в keycloak

- `S3` - настройки файлового хранилища S3
- `S3__ENABLED` = `false` : присутствует ли S3 в инсталляции
- `S3__URL` = `null` : адрес API хранилища S3
- `S3__SSL_VERIFICATION` = `true` : использовать ли ssl верификацию при подключении к s3: true/false/<путь до сертификата>
- `S3__BUCKET` = `null` : имя бакета в S3 хранилище
- `S3__ACCESS_KEY` = `secret:s3-access` : S3 access key
- `S3__SECRET_ACCESS_KEY` = `secret:s3-secret` : S3 secret access key
- `S3__DATASET_CSV_PATH` = `"dataset_csv"` : каталог внутри бакета, куда выгружаются CSV датасетов
- `S3__MAX_FILE_SIZE_MB` = `64` : максимальный размер файла в мегабайтах
- `S3__FILE_ENCODING` = `"utf-8"` : кодировка, используемая для текстовых файлов
- `S3__URL_EXPIRE_INTERVAL` = `300` : как долго живет ссылка для скачивания файлов, в секундах
- `S3__HTTP_TIMEOUT` = `15` : максимальное время ожидания ответа от S3 в секундах
- `SWAGGER` - настройки работы swagger
- `SWAGGER__ENABLED` = `true` : Доступен ли интерфейс Swagger UI по адресу `/api/doc`, `/tech/doc` и `/api/v100/doc`
- `SWAGGER__USE_BUNDLED` = `true` : Использовать встроенную версию `.js` и `.css` ресурсов для сваггера
- `SWAGGER__TOKEN_URL` = `null` : кастомный адрес, по которому получать токен

Web-сервер: guincorn

В качестве веб сервера для API приложения используется `gunicorn` (WSGI) с воркерами `uvicorn` (ASGI), полный список настроек доступен в официальной документации.

Рекомендуемые:

- `WEB_CONCURRENCY` = `4` - количество воркеров `uvicorn`
- `GUNICORN_CMD_ARGS` = `--keep-alive 65` - дополнительные параметры командной строки
- `FORWARDED_ALLOW_IPS` = `'*'` - фильтр на IP, с которых принимаются соединения

Планировщик celery: redbeat

Запуск заданий на расписании осуществляется через плагин к `redbeat`.

При запуске планировщик смотрит на следующие переменные окружения:

- `EXECUTOR__BROKER_URL` = `secret:executor-broker-url` : адрес брокера сообщений redis/sentinel или rabbit-mq
- `EXECUTOR__OPTIONS` = `{}` : дополнительные опции Redbeat

Мониторинг celery: flower

Настройка мониторинга описана на странице Monitoring

Менеджер процессов: supervisor

Все компоненты приложения стартуют и управляются через supervisor, полный список настроек доступен в официальной документации.

Настройки менеджера распроложены в `axiom/etc/supervisor.conf`, дополнительная ручная конфигурация не предусмотрена.

3.1.2 Web-интерфейс

```
// адрес API бэкэнда
FS_API = '/api'
```

```
// адрес Keycloak
AUTH_URL = 'https://auth.k8s.datasapience.ru/auth'
// клиент авторизации в keycloak
CLIENT_ID = 'frontend'
// Включение / отключение документации в интерфейсе
DOCS_ENABLED = 'false'
// Суффикс после домена адреса WEB интерфейса
ROOT_PATH_PREFIX = ''
// Доступен ли SQL лоадер в интерфейсе
LOADER_SQL = 'false'
// адрес kolmogorov predicate, если предусмотрен в инсталляции
PREDICATE_URL = 'https://predicate-dev.k8s.datasapience.ru'
// Наличие дага расчета статистик датасетов
DENSITY_PLOT = 'false'
// Есть ли sentry сервер
SENTRY_ENABLED = 'true'
// адрес sentry сервера
SENTRY_DSN = 'https://<token>@sentry-kolmogorov.k8s.datasapience.ru/4'
// переименования ключей в логгере
LOG_KEY_REDEFINE = '{\"level\":\"lvl\", \"sdkProcessingMetadata\": \"sdkPM\"}'
// Версия приложения, которая будет отображаться в углу интерфейса,
// при необходимости указать отличную от релиза
DISPLAY_VERSION = ''
// Адрес сервиса генератора Excel каталога
INFOMAP_GENERATOR_API = ''
// Отображаемое название приложения
APP_NAME = 'Morphism'
// Папка с набором логотипов для приложения
IMAGES_FOLDER_KEY = 'morphism'
```

Поддержка Excel каталога (инфокарты)

Для реализации возможности импорта каталога в формате Excel необходимо:

- Развернуть сервис генератор инфокарты
- Указать в values.yaml адрес сервиса в переменную `INFOMAP_GENERATOR_API`

Настройка иконок и логотипов в web-интерфейсе

Предустановлены логотипы/иконки для следующих продуктов: Morphism, Audience, Alphyn. Значения переменной `IMAGES_FOLDER_KEY` соответственно `morphism`, `audience`, `alphyn`.

Для настройки собственных иконок и логотипов следующий порядок действий:

- Создать новую папку в `public/images/<folder name>`
- Загрузить в нее изображения со следующими именами:
- `logo_favicon.svg` - иконка для вкладки браузера
- `logo_main.svg` - иконка на главной странице входа
- `logo_title.svg` - логотип в верхнем левом углу интерфейса
- Указать `<folder name>` в переменной `IMAGES_FOLDER_KEY`

3.1.3 Пояснения для SSL settings

Настройка SSL верификации доступна для разделов `S3` и `KEYCLOAK`. Правила настройки общие:

- Если SSL верификация не требуется, указать значение переменной `<PREFIX>__SSL_VERIFICATION = false`
- Если все сервисы, к которым идет обращение, поднимаются с тем же самым сертификатом, что и приложение axiom, достаточно указать значение `<PREFIX>__SSL_VERIFICATION = true`.
- Если какой-то из сервисов поднимается с самоподписанным сертификатом, то необходимо:
- Выгрузить его сертификат в файл `<prefix>_cert.crt`.
- Создать секрет в кластере k8s из файла-сертификата с именем ssl-cert-secret, содержащий в себе все необходимые самоподписанные сертификаты.
- Примонтировать файлы-сертификат к приложению Axiom (закомметированные строки в values.yaml: `service.persistentVolumes -> ssl-verification-cert`)
- Указать значение переменной `<PREFIX>__SSL_VERIFICATION` = `'/usr/certs/<prefix>_cert.crt'`

3.2 Подключение к данным

Данный раздел описывает подключение в *Датастору* - движку, который занимается обработкой данных. Это может быть как классическая СУБД типа *PostgreSQL*, так и что-то похожее на неё как *Apache Spark*.

Помимо URL можно указать дополнительные опции соединения к датастору в `v100.source.options`. Ниже перечислены некоторые из них:

- `pool_size` - The number of connections to keep open inside the connection pool.
- `max_overflow` - the number of connections to allow in connection pool “overflow”, that is connections that can be opened above and beyond the `pool_size` setting.
- `pool_timeout` - Number of seconds to wait before giving up on getting a connection from the pool.
- `pool_recycle` - This setting causes the pool to recycle connections after the given number of seconds has passed. It defaults to -1, or no timeout.
- `pool_reset_on_return` - Action upon returning connection back to pool: `'rollback'`, `'commit'` or `null`.
- `pool_use_lifo` - Use LIFO (last-in-first-out) when retrieving connections from the pool instead of FIFO (first-in-first-out).
- `max_identifier_length` integer; Override the max_identifier_length determined by the dialect. if `null` or zero, has no effect.
- `connect_args` - A dictionary of options which will be passed directly to the driver as additional arguments.
- `hide_parameters` - When set to True, SQL statement parameters will not be displayed in INFO logging nor will they be formatted into the string representation of StatementError objects.
- `isolation_level` - This string parameter is interpreted by various dialects in order to affect the transaction isolation level of the database connection. The parameter essentially accepts some subset of these string arguments: `"SERIALIZABLE"`, `"REPEATABLE READ"`, `"READ COMMITTED"`, `"READ UNCOMMITTED"` and `"AUTOCOMMIT"`. Behavior here varies per backend, and individual dialects should be consulted directly.

3.2.1 Рекомендованные параметры

Info

В зависимости от типа СУБД и используемого драйвера аргумент, задающий timeout при подключения может называться по-разному, но его *обязательно* необходимо задавать, иначе можно попасть в ситуацию, когда на транспортном уровне есть проблемы, и пакеты не доходят до клиента и попытка подключения висит бесконечно (например MTU в виртуальной сети не соответствует настройкам по размеру пакета в кластере greenplum)

Казалось бы, что `pool_timeout` ровно за это и отвечает, но нет. Опция `pool_timeout` срабатывает только в случае, когда сервер отвечает да/нет, но если ожидание ответа подвисает, то подвисает и приложение.

```
# v100source.options
{  # Параметры движка
    "pool_size": 50,     # Максимальное количество одновременных подключений
    "max_overflow": 0,   # Максимально допустимый перелимит
    "pool_timeout": 60,  # Таймаут на получение подключения из пула
    "connect_args" {     # параметры дравера
        "connect_timeout": 10,
        ...
    }
}
```

PostgreSQL & Greenplum

URL Подключения к базе должен иметь правильную схему:

- **PostgreSQL**: `postgres://...`, `postgresql://...` или `postgressql+psycopg2://...`
- **Greenplum**: `greenplum://...`

Рекомендуется использование параметров `keepalives...` для надежного определения ситуаций, когда соединение с БД разрывается в процессе работы, из-за сбоя БД или потери сетевой связанности.

```
# v100source.options
{
    ...
    "connect_args": {  # Опции драйвера psycopg2
        "connect_timeout": 10,
        "keepalives": 1,
        "keepalives_idle": 10,
        "keepalives_interval": 5,
        "keepalives_count": 2
    }
}
```

Также через `"options": "-c ..."` можно дополнительно передать параметры сессии по-молчанию. Полный список таких параметров в документации PostgreSQL и по Greenplum

```
# PostgreSQL
{
    ...
    "connect_args": {
        ...
        "options": "-c idle_in_transaction_session_timeout=5min",
    }
}
```

```
# Greenplum
{
    ...
    "connect_args": {
        ...
        "options": "-c max_statement_mem=4000MB -c gp_default_storage_options=orientation=column,compresstype=zstd,compresslevel=1"
    }
}
```

Кастомизировать опции, с которыми создаются таблицы, можно через параметры в ключе `fs_settings.aux` таблицами являются те, которые удаляются после выполнения трансформации.

```
{
    ...
    "fs_settings":{
        "create_table_prefix": "",
        "create_table_postfix": "WITH (appendoptimized=true,orientation=column,compresstype=zstd,compresslevel=1)",
        "create_table_aux_prefix": "UNLOGGED",
        "create_table_aux_postfix": "WITH (appendoptimized=true,autovacuum_enabled=false,orientation=column,compresstype=zstd,compresslevel=1)"
    },
    "connect_args": {
        ...
    }
}
```

Apache Spark

Работа с кластером Apache Spark реализована через отправку запросов к hive thift серверу. Который должен быть предварительно запущен и сконфигурирован в кластере.

URL Подключения должен иметь схему `spark://...`

```
# v100source.options
{
    ...
    "connect_args": {
        "connect_timeout": 10
    }
}
```

Clickhouse

URL Подключения должен иметь схему `clickhouse://...`

```
{
    "connect_args": {
        "connect_timeout": 10,
        "settings": {
            "handshake_timeout_ms": 1,
        },
```

```
    }
}
```

В ключе `connect_args` задаются параметры подключения. Полный список возможных значений можно посмотреть в описании конструктора объекта `Connection` пакета `clickhouse-driver`.

Также `connect_args` может содержать ключ `settings`, в котором можно задать:

- сессионные настройки Clickhouse;
- дополнительные параметры объекта `Client` пакета `clickhouse-driver`.

ОГРАНИЧЕНИЯ CLICKHOUSE

- По причине ошибки в Clickhouse в источниках данных невозможно использовать колонки с именами, зарезервированными для внутреннего использования приложением: from_dttm, to_dttm, report_dttm,_created_dttm,_updated_dttm,_run_rk.

Apache Impala

Подключение к Apache Impala возможно двумя способами:

- посредством python библиотеки `impyla`
- посредством ODBC драйвера

ПОДКЛЮЧЕНИЕ ПОСРЕДСТВОМ БИБЛИОТЕКИ IMPYLA

URL подключения должен иметь вид `impala://[user[:password]]host[:port][/database]`.

В ключе `connect_args` параметров подключения возможно задать дополнительные параметры, соответствующие аргументам функции `connect` библиотеки.

ПОДКЛЮЧЕНИЕ ПОСРЕДСТВОМ ODBC ДРАЙВЕРА

Для подключения в системе должен быть установлен и сконфигурирован ODBC драйвер. Система тестировалась с драйвером Cloudera 2.6.11.

В соответствии с инструкциями, приложенными к драйверу необходимо прописать название драйвера и (опционально) конфигурации источников в соответствующих ini файлах.

Далее необходимо сформировать ODBC строку подключения (см. документацию к драйверу) и сконвертировать ее в форму, пригодную для использования в URL. Для этого необходимо использовать python функцию `quote_plus`.

В простейшем случае строка подключения может иметь вид:

```
params = quote_plus(
    "DRIVER={Cloudera Impala ODBC Driver 64-bit};"
    "Host=impala;"
    "Port=21050;"
)
```

Пример более сложной конфигурации с сертификатами (должны находиться в соответствующей директории).

```
params = quote_plus(
    "DRIVER={Cloudera Impala ODBC Driver 64-bit};"
    "Host=host;"
    "Port=21050;"
    "AuthMech=3;"
    "UID=user;"
    "PWD=password;"
    "SSL=1;"
    "AllowSelfSignedServerCert=1;"
    "AllowHostNameCNMismatch=1;"
    r"TrustedCerts=/root/root-certs.pem;"
)
```

Конечный URL подключения будет иметь вид:

```
url = f"impala+pyodbc:///?odbc_connect={params}&autocommit=True"
```

Oracle Database

URL подключения должен иметь вид `oracle://[user[:password]@]host[:port]/?service_name=PDB`.

где `PDB` - имя PDB (например `XEPDB1`).

В `connect_args` возможно задать дополнительные параметры подключения, например

```
{
    "connect_args": {
        "tcp_connect_timeout": 10
    }
}
```

Внимание! В БД должны быть созданы все необходимые схемы (пользователи в терминологии Oracle) для работы системы. Для системного пользователя (поле `user` в строке подключения), а также для всех созданных пользователей должен быть выдан полный список прав для совершения операций в БД. Для настройки БД проконсультируйтесь с руководством по БД Oracle.

СПРАВОЧНИК ДЛЯ ORACLE

Для корректной работы с БД oracle необходимо для всех численных типов в справочнике fs_type указать `py_type=decimal`

NOTE: Oracle не поддерживает тип данных `bool` до версии 23c. В качестве альтернативы можно использовать CHAR(5) / NUMBER(1).

Использование типов данных с плавающей точкой

Если в справочнике имплементаций используется численный тип данных с плавающей точкой как целое число, например `NUMERIC(18,0)`, то его нужно добавить в справочник типов данных по умолчанию с указанной в справочнике имплементаций точностью:

db_type	default_fs_type	db_name
NUMERIC	FLOAT	oracle
NUMERIC(18,0)	INT	oracle

Имплементации:

fs_type	db_type	db_name
INT	NUMERIC(18,0)	oracle
AMOUNT	NUMERIC(12,4)	oracle
FLOAT	NUMERIC(18,6)	oracle

3.3 Обработка данных

Для тюнинга производительности запросов в приложении предусмотрены профили исполнения.

Настройка профиля делается прямым INSERT или UPDATE в таблице meta.ref_execution_profile:

- `name` - идентификатор профиля
- `job_options` - опции профиля в формате json
- `author` - кто сделал запись (техническое поле)
- `create_dttm` - дата создания записи (техническое поле)
- `updated_dttm` - дата изменения записи (техническое поле)

Как высчитываются итоговые параметры исполнения с учетом дефолтных можно почитать в разделе "Execute job" документации Product Map, ниже приведена выжимка по основным:

- `async_factor` - максимальное количество SQL запросов в параллель в рамках одного исполнения `job`
- `params` - раздел с перемеными, влияющий на логику работы узла
- `storage_key` - значения ключа дистрибуции по-умолчанию, если движок данных это поддерживает (default: [])
- `create_index` - создавать ли индекс по бизнес ключу, после создания таблицы (default: False)
- `analyze` - запускать ли сбор статистик после создания таблицы (default: True)
- `advanced` - параметры сесии (SET опции) для различных СУБД, задаются как список строк, без ведущего SET
- `greenplum` - любые SET опции greenplum, например `"greenplum": ["max_statement_mem=4000MB", "gp_default_storage_options=orientation=column"]`
- `postgresql` - любые SET опции postgresql, например `"postgresql": ["idle_in_transaction_session_timeout=5min" ]`

3.3.1 Дополнительные параметры

В данном разделе приведены дополнительные настройки для различных движков баз данных, позволяющие оптимизировать загрузку данных из внешних таблиц.

В режиме быстрой загрузки создается т.н. внешняя таблица, то есть данные физически не перемещаются в базу данных а остаются в S3, а в базе создается таблица, ссылающаяся на эти данные.

Создание внешней таблицы осуществляется узлом `import`. Конфигурации такого режима отличаются для разных движков.

Greenplum

Для быстрой загрузки данных к кластере должен быть установлен и сконфигурирован Greenplum Platform Extension Framework (PXF).

Для создания внешней таблицы должно быть известно имя PXF профиля указывающего на S3 хранилище, из которого осуществляется загрузка.

Для создания внешней таблицы в узле `import` необходимо задать следующие параметры:

```
{
   "type": "import",                          # быстрое чтение (создание внешней таблицы)
   "body": {
        "path": "s3://<file name>",           # путь на s3
        "columns": {                          # список стобцов в файле с типами
            "column name": "fs_type",
            "column name": "fs_type",
            ...
        }
    },
    "options": {
        "format": "csv" | "parquet",          # file format
        "delimiter": '<delimiter char>'       # default: ',' только для csv
    },
```

```
    "params": {
        "pxf_profile": '<имя pxf профиля>'
    }
}
```

Задание кодировки и создание первичных ключей во внешних таблицах не поддерживается. Кодировка предполагается `utf8`.

Spark

Для быстрой загрузки файлов кластер Spark должен быть настроен для работы с выбранным S3 провайдером, пример настроек.

Для создания внешней таблицы в узле `import` необходимо задать следующие параметры:

```
{
   "type": "import",                          # быстрое чтение (создание внешней таблицы)
   "body": {
        "path": "s3://<file name>",           # путь на s3
        "columns": {                          # список стобцов в файле с типами
            "column name": "fs_type",
            "column name": "fs_type",
            ...
        }
    },
    "options": {
        "format": "csv",                      # file format
        "delimiter": '<delimiter char>',      # default: ','
        "encoding": '<encoding>',             # default: 'utf8'
    },
    "params": {
        "s3a_mode": true,
    }
}
```

На данный момент поддерживаются только `csv` файлы.

3.4 Ролевая модель

Ролевая модель приложения подразумевает разграничение доступа для пользователя приложения:

- к функциональности приложения в зависимости от роли пользователя
- к конкретным объектам в зависимости от уровня доступа

3.4.1 Настройка авторизации

Для авторизации пользователей приложение использует провайдер Keycloak версии не ниже 23.0.

Гайдлайн актуален для `KEYCLOAK__STRATEGY=introspect`.

Для работы приложения необходимо не менее двух клиентов - публичный (получение токена) и приватный (валидация токена). Порядок действий по настройке Keycloak с нуля:

1. Создать приватный клиент авторизации для приложения (`KEYCLOAK__CLIENT_ID` в настройках) и указать его секрет в `KEYCLOAK__CLIENT_SECRET`.
2. В приватном клиенте приложения создать клиентские (!) роли, соответствующие уровням доступа, описанным в таблице ниже. Названия ролей соответствуют конфигурации в `KEYCLOAK__ACCESS_ROLES`.
3. Выдать пользователям соответствующие клиентские роли приватного клиента. Роли на уровне realm не будут работать, даже с идентичными названиями.
4. Создать новый клиент, который будет использоваться пользователями для получения токена при авторизации в приложении. Публичных клиентов может быть несколько.

3.4.2 Доступ к функциональности (эндпоинтам)

Роль пользователя определяет доступные пользователю действия:

Роль	Описание
axiom_read	Доступ к чтению каталога
axiom_transform	axiom_read + Доступ к конструктору и трансформам
axiom_load	axiom_transform + Доступ к загрузке данных
axiom_admin	Полный доступ ко всем функциям приложения

Роль, необходимая для доступа к конкретному эндпоинту, описана в Product Map.

3.4.3 Доступ к объектам

Уровень доступа задается к следующим объектам:

- загрузчики
- датасеты
- источники данных

Уровень доступа к объекту определяет доступные пользователю действия по отношению к объекту:

Уровень доступа	Описание
no_access	Ничего нельзя, запрещен даже просмотр
read	Можно только просматривать и читать данные
full	Полный доступ к объекту

Уровень доступа пользователя определяется по приоритетам: персональный уровень доступа назначенный пользователю -> базовый уровень доступа к объекту.

NB!: пользователь с ролью `axiom_admin` имеет доступ `full` ко всем объектам по умолчанию.

Базовый уровень доступа назначается при создании объекта и отображается в каталоге прав доступа со знаком `*`. Базовый уровень доступа при создании объекта настраивается через переменную окружения `DEFAULT_PERMISSION_LEVEL`.

При создании объекта его автор получает уровень доступа = full.

Уровень доступа по объекту (в т.ч. базовый) может быть изменен только пользователем с уровнем доступа к объекту = full или при наличии роли axiom_admin через специальные эндпоинты.

Для сегментов уровень доступа к загрузчику и датасету меняется синхронно и всегда совпадает.

3.5 Мониторинг

Раздел описывает инструменты мониторинга приложения

3.5.1 Мониторинг meta: pg_stat_statements

Официальная документация

Включение мониторинга

1. В конфигурационном файле postgresql.conf указать: `shared_preload_libraries='pg_stat_statements'`
2. Перезапустить сервис
3. Включить расширение под пользователем с ролью `pg_read_all_stats`: `CREATE EXTENSION IF NOT EXISTS pg_stat_statements;`

Настройка в docker: **docker-compose.yml**

```
services:
  postgres:
    command:
      postgres  -c shared_preload_libraries='pg_stat_statements'
```

Настройка в kubernetes/bitnami/postgresql: **values.yaml**

```
postgres:
  auth:
    postgresPassword: ...
  primary:
    extendedConfiguration: |
      shared_preload_libraries = 'pg_stat_statements'
    initdb:
      scripts:
        statistic.sh: |
          #!/bin/sh
          export PGPASSWORD=$POSTGRES_POSTGRES_PASSWORD
          psql -U postgres -d meta -c "create extension if not exists pg_stat_statements;"
```

Отключение мониторинга

1. Выключить расширение под пользователем с ролью pg_read_all_stats `DROP EXTENSION pg_stat_statements;`
2. Убрать параметр `shared_preload_libraries` из конфигурационного файла postgresql.conf

Использование

`SELECT * FROM pg_stat_statements;`

Пример запроса для вывода 20 самых медленных запросов:

```
SELECT
    query,
    round(total_exec_time::numeric/1000, 2) AS total_time_s,
    round(min_exec_time::numeric/1000, 2) AS min_time_s,
    round(max_exec_time::numeric/1000, 2) AS max_time_s,
    round(mean_exec_time::numeric/1000, 2) AS mean_time_s,
    calls,
    round((100 * total_exec_time / sum(total_exec_time::numeric) OVER ())::numeric, 2)
      AS usage_percentage
FROM
    pg_stat_statements
ORDER BY total_time_s DESC
LIMIT 20;
```

3.5.2 Мониторинг celery: flower

Мониторинг Celery осуществляется через приложение flower, полный список настроек доступен в официальной документации.

При запуске `flower` смотрит на следующие переменные окружения:

- `EXECUTOR__BROKER_URL` = `secret:executor-broker-url` : адрес брокера сообщений redis/sentinel или rabbit-mq
- `FLOWER_CMD_ARGS` = `null` - дополнительные аргументы командной строки для celery flower

3.6 Логирование

3.6.1 Логирование

События для логирования (см. таблицу ниже):

- **Code** - S=system, B=backend, A=Async_execution O=other
- **Event** - событие для логирования
- **FS event** - событие, в котором производится вызов функции логирования
- **Level** - уровень события: DEBUG, INFO, WARNING, ERROR, CRITICAL
- **User** - пользователь инициировавший событие
- **Location** - где произошло событие: url эндпоинта
- **Context** - аргументы исполнения, приведшие к событию
- **Message** - тестовое сообщение события
- **Traceback** - если +, то в функцию передаётся текст traceback'a.

Коды событий

Code	Event	Level	Description
S001	Старт работы контейнера	INFO	Первый лог контейнера
S002	Остановка контейнера	CRITICAL	Любая остановка контейнера
S003	Начало работы AC	INFO	Успешное завершение миграций
B001	Успешное создание объекта	INFO	...
B002	Чтение объекта	INFO	...
B003	Редактирование объекта	INFO	...
B004	Удаление объекта	INFO	...
B005	Неуспешная валидация данных	ERROR	Невозможно обработать запрос
B006	Сбой подсистемы	CRITICAL	Непредвиденная ошибка
B008	Объект отсутствует	ERROR	Любая ошибка 404
B009	Чтение списка объектов	INFO	Все get list методы
A001	Старт обработки данных	INFO	Старт работы worker-процесса
A002	Окончание обработки данных	INFO	Окончание работы worker-процесса
A003	Ошибка обработки данных	ERROR	Ошибка при работе worker-процесса
O001	Прочая информация	DEBUG	...

3.6.2 Настройка логирования

Данный раздел описывает возможности настройки логирования в приложении. Логирование настраивается при помощи конфигов в формате json.

Настройки логирования называются **профиль логирования**. Количество профилей логирования не ограничено. Активный профиль логирования имеет значение флага `active_flg=True`.

Форматтер - определяет правила форматирования текстовых и json логов. Количество конфигураций форматтеров не ограничено.

Хендлер - определяет куда будет отправлен лог - в консоль (stdout) и/или в файл.

Логгер - позволяет определить уровень логирования (INFO | DEBUG etc.) и хендлер для компонента приложения. Если компонент в явном виде не определен в списке логгеров, то будет использован логгер `root`.
Для отключения хендлера логов компонента необходимо указать `"handlers": []`. В таком случае логов от компонента не будет.

JSON-конфиг

Пример конфигурации:

```
{
    "version": 1,
    "disable_existing_loggers": false,
    "formatters": {
      "text": {
        "()": "axiom.log_formatters.TextFormatter",
        "len_limit": -1
      },
      "text_limited": {
        "()": "axiom.log_formatters.TextFormatter",
        "len_limit": 16000
      },
      "json": {
        "()": "axiom.log_formatters.JSONFormatter",
        "fmt_keys": {},
        "len_limit": -1
      },
      "json_limited": {
        "()": "axiom.log_formatters.JSONFormatter",
        "fmt_keys": {},
        "len_limit": 16000
      },
      "json_full": {
        "()": "axiom.log_formatters.JSONFormatter",
        "fmt_keys": {},
        "len_limit": -1,
        "builtin_flds": true
      }
    },
    "handlers": {
      "stdout": {
        "class": "logging.StreamHandler",
        "level": "DEBUG",
        "formatter": "json_limited",
        "stream": "ext://sys.stdout"
      },
      "file": {
        "class": "logging.handlers.TimedRotatingFileHandler",
        "level": "DEBUG",
        "formatter": "json_full",
        "filename": "./axiom.log",
        "when": "midnight",
        "backupCount": 30
      }
    },
    "loggers": {
      "root": {
        "level": "DEBUG",
        "handlers": ["stdout", "file"]
      },
      "sqlalchemy": {
        "handlers": []
      },
      "sqlalchemy.engine": {
        "level": "INFO"
      },
      "fastapi": {
        "handlers": []
```

```
      },
      "uvicorn": {
        "handlers": []
      },
      "boto3": {
        "level": "ERROR",
        "handlers": []
      },
      "botocore": {
        "level": "ERROR",
        "handlers": []
      },
      "urllib3": {
        "level": "ERROR",
        "handlers": []
      },
      "httpx": {
        "level": "ERROR",
        "handlers": []
      },
      "requests": {
        "level": "ERROR",
        "handlers": []
      },
      "sentry_sdk": {
        "level": "ERROR",
        "handlers": []
      }
    }
}
```

Параметры, указанные в конфигурации:

- `version` - Обязательный параметр с неизменяемым значением 1
- `disable_existing_loggers` - отключение существующих логгеров, не входящих в конфигурацию (по умолчанию false)
- `formatters` - Список форматтеров логгера
- `text_*` - Преднастроенный форматтер для вывода лога в текстовом формате, возможные параметры:
- `()` - Класс форматтера, преднастроенный из axiom `axiom.log_formatters.TextFormatter`
- `len_limit` - Ограничение длины лога
- `json_*` - Преднастроенный форматтер для вывода лога в формате json, возможные параметры:
- `()` - Класс форматтера, преднастроенный из axiom `axiom.log_formatters.JSONFormatter`
- `fmt_keys` - json с переименованием имен полей в логе
- `len_limit` - Ограничение длины лога
- `builtin_flds` - Вывод всех возможных полей лога (нужно для самого расширенного логирования)
- `handlers` - Список хэндлеров логгера
- `stdout` - Вывод лога в stdout, параметры:
- `class` - Класс хэндлера (по умолчанию встроенный `logging.StreamHandler`, подробнее см. документацию)
- `level` - Уровень логирования
- `formatter` - Форматтер из секции `formatters`
- `stream` - Выбор стрима (по умолчанию `ext://sys.stdout`)
- `file` - Вывод лога в файл, параметры:
- `class` - Класс хэндлера (по умолчанию встроенный `logging.handlers.TimedRotatingFileHandler`)
- `level` - Уровень логирования
- `formatter` - Форматтер из секции `formatters`
- `filename` - Абсолютный путь до файла с логом
- `when` - Когда осуществлять закрытие файла и открытие нового (по умолчанию `midnight`, подробнее о формате см. документацию)
- `backupCount` - Количество файлов с логами (по умолчанию 30)
- `loggers` - Список логгеров
- `root` - Основной логгер, где производится вся настройка, параметры:
- `level` - Уровень логирования
- `handlers` - Список хэндлеров из секции `handlers`
- `sqlalchemy.engine` - Логгер sqlalchemy, отвечает за вывод выполняемых запросов в лог
- `level` - Уровень логирования. Для вывода основных запросов указать `INFO`.
- `*` - Все остальные логгеры
- `handlers` - необходимо каждому из логгеров значение `[]`, чтобы отключить встроенные хэндлеры.

Настройка API

В API существуют преднастроенные профили логирования для всех 5-ти уровней (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`).

Чтобы посмотреть все существующие конфигурации, необходимо вызвать метод `/sys/logging_profiles`.

Чтобы обновить конфигурацию, необходимо вызвать метод `/sys/logging_profiles/{name}` передав имя профиля и json-конфиг, в формате, описанном выше.

Отправка логов в fluent через UDP-plugin

Для подключения к fluent в конфигурацию необходимо добавить новый `handler` и подключить его к желаемым логгерам.

```
{
    ...
    "handlers": {
      "fluent": {
            "class": "fluent.handler.FluentHandler",
            "host": "localhost",
            "port": 24224,
            "tag": "test.logging",
            "formatter": "json_full",
            "level": "DEBUG"
      },
      ...
    },
    "loggers": {
      "root": {
        "level": "DEBUG",
        "handlers": ["stdout", "file", "fluent"]
      },
      ...
    },
    ...
}
```

4. Troubleshooting

4.1 Долгий отклик API

Увеличить количество воркеров uvicorn - параметр в чарте `WEB_CONCURRENCY`. Для определения допустимого значения `WEB_CONCURRENCY` можно руководствоваться следующей формулой: `WEB_CONCURRENCY = <количество ядер> * 2 + 1`

4.2 При перезагрузке интерфейса возникает 404 код ошибки

Ошибка может возникнуть при установке WEB не из готового образа, выданного командой приложения.

В конфиг-файле nginx убедиться в наличии следующей настройки `try_files $uri $uri/ /index.html;`:

```
..
- name: nginx-config
  image:
    ..
  command:
    ..
  args:
    - |
    ..
    location / {
        ..
        index  index.html index.htm;
        try_files $uri $uri/ /index.html;
    }
..
```