Документация SDK

The Platform's Software Development Kit (или просто *SDK*) это универсальный инструментарий на языке Python, предназначенный для взаимодействия с Платформой. SDK позволяет разработчикам создавать и изменять приложения для интеграции или расширения функциональности Платформы. С помощью SDK вы можете легко разрабатывать расчетные модели, конфигурировать мониторинг и отчетность, а также дополнять ваши приложения другими функциями и возможностями.

SDK включает в себя API-классы и функции, необходимых для использования сущностей Платформы. Кроме того, SDK предоставляет список служебных методов, которые также помогают создавать и оптимизировать различные процессы взаимодействия приложения с Платформой. Таким образом, предоставленный инструментарий значительно упрощает создание различных фичей, а также поддерживает чистоту кода в приложениях.

SDK также предоставляет ряд встроенных приложений, которые позволяют оперировать готовыми функциями и возможностями. Данные примеры также могут быть использованы как приложения, готовыми к практическому использованию на Платформе.

Темы документации

Ознакомьтесь руководствами и примерами кода.

Приступить к работе

Установите SDK и создайте свое первое приложение.

- Установка SDK
- Создайте свое первое приложение
- Рекомендации

Руководства для разработчиков

Изучите руководства и примеры кода.

- Сигналы (создание/чтение/запись)
- Приложение с валидацией параметров
- Клиент уведомлений
- Клиент артефактов
- Хартбит
- Логирование

Встроенные приложения

Ознакомьтесь с готовыми приложениями.

- Data player
- Expectations
- Jupyter Notebook Reports
- Data periodicity
- Metrics
- PandasEvaluator
- Watchdog
- Latency measurer
- Signal Notificator

Встроенные приложения

Data Player

Data player - это приложение для создания и заполнения сигналов данными, хранящимися в файле. Запущенный data player перебирает строки данных в файле и многократно записывает значения в указанные источники вывода. Это эмулирует поведение реальных данных, поступающих из внешних источников, например, датчиков, что позволяет тестировать модели и приложения в производственной среде.

Узнать больше →

Приступить к работе

Ниже приведены и описаны представленные инструкции по началу работы с SDK.

Страница *установки SDK* предоставляет подробные инструкции по установке SDK. Эти руководства также включают в себя необходимые требования и зависимости, которые могут потребоваться для успешной установки и последующей разработки.

Если вы только начинаете работать с SDK, рекомендуется следовать инструкциям на странице *Создайте свое первое приложение*. Эта страница показывает пошаговое создание простого приложения, а также демонстрирует интеграцию этого приложения с Платформой. Данные инструкции могут служить отправной точкой для новых разработчиков, а также могут быть полезными для опытных разработчиков в качестве памятки.

Рекомендации включают в себя примеры использования SDK, демонстрирующие, как настроить гранулярности для потоковой передачи данных сигналов, а также как хранить состояние моделей в вашем приложении машинного обучения.

Установка SDK

SDK платформы распространяется в виде пакета Python в формате WHL. Чтобы успешно установить этот пакет и начать использовать SDK, ознакомтесь со следующими предварительными требованиями:

Предварительные требования

- **Среда Python**: Необходимо использовать среду Python 3.9. Рекомендуется развернуть среду этой версии с помощью `conda` или `venv`.
- **Установка PIP**: Установите пакетный инсталлятор `PIP` для Python.

Выполнив эти предварительные требования, ваш компьютер будет готов к установке SDK.

Установка

SDK должен быть установлен используя соответсвующий файл WHL. Установите данный файл с помощью следующей команды pip:

```
pip install platform_sdk-|release|-py3-none-any.whl
```

Возможные проблемы и их решения

Пользователи macOS

1. Если у вас возникли проблемы с установкой `lz4`, то вам может потребоваться установка `xcode-select`:

   ```
   xcode-select --install
   ```

2. Если у вас возникли проблемы с установкой `numpy`, то может помочь установка через conda

   ```
   conda create --name my-sdk-env python=3.9
   conda activate my-sdk-env
   conda install numpy
   pip install platform_sdk
   ```

Пользователи Windows

Если у вас возникли проблемы с установкой `lz4`, вам может потребоваться установить VS BuildTools. Убедитесь, что вы выбрали Windows SDK при установке VS BuildTools, так как он необходим для компиляции пакета `lz4` во время установки SDK.

Пользователи Linux

Если у вас возникли проблемы с установкой SDK на Линуксе, вам потребуется создать среду Python 3.9, что можно легко сделать с помощью `conda`:

```
conda create -n <env_name> python=3.9.11
```

После настройки среды активируете ее с помощью команды: `conda activate <env_name>`.

Создайте свое первое приложение

В этом руководстве вы узнаете, как создать простое приложение, взаимодействующее с платформой. В первой части руководства мы напишем код приложения и запустим его локально, а во второй части развернем его на платформе и запустим из пользовательского интерфейса платформы.

Предварительные условия

- Локально установленная библиотека platform SDK.
- Аккаунт на платформе Тайга.
- Знакомство с понятиями сигналов. Ознакомьтесь с разделом «Сигналы» документации платформы. Чтобы получить доступ к документации платформы, войдите в систему, нажмите на значок пользователя в левом нижнем углу страницы и выберите пункт «Документация по платформе».
- Настроенный реестр Docker. Для развертывания приложения на платформе оно должно быть упаковано в образ Docker и опубликовано в реестре Docker. Если вы не знакомы с реестрами Docker, мы рекомендуем начать с этого руководства.
- Если возникли трудности с использованием реестра Docker, рекомендуется использовать каталог Yandex Cloud. Инструкции можно найти пройдя по этой ссылке.

Код приложения

Приложение записывает одно числовое значение в заданный сигнал. Если сигнал не существует, он будет создан.

Сигнал представляет собой временной ряд, то есть последовательность пар *<время, значение>*. Следовательно, добавление значения к сигналу означает добавление пары. Наше приложение добавит к сигналу значение с текущим моментом времени.

Создайте папку проекта и файл `entrypoint.py`, который содержит код нашего приложения.

entrypoint.py

Следующий фрагмент показывает код нашего приложения.

```
import datetime
from typing import Dict

from platform_sdk import (
    PipelineStep,
    SignalInputValue,
    SignalsOps,
    SignalWriterClient,
    StreamType,
)


class BasicApp(PipelineStep):
    def __init__(
        self,
        signal_ops: SignalsOps,
        signal_writer: SignalWriterClient,
    ):
        super().__init__()
        self._signal_writer = signal_writer
        self._signal_ops = signal_ops

    def start(
        self,
        task_id: int,
        job_params: Dict[str, str],
        input_params: Dict[str, str],
    ):
        signal_id = input_params["OUTPUT_SIGNAL_ID"]

        # create the signal if it does not exist
        res = self._signal_ops.is_exists([signal_id])
        if not res[0]["existing"]:
```

```
            signal = self._signal_ops.create_signal(
                label="signal_name",
                public_id=signal_id,
                data_stream_type=StreamType.ROW,
            )

        # write a value to the signal
        self._signal_writer.write_row(
            [SignalInputValue(signal_id, 3.0)],
            datetime.datetime.now(datetime.timezone.utc),
        )
```

Несколько вещей, на которые стоит обратить внимание:

- Класс приложения должен наследовать от класса `PipelineStep`. Класс `PipelineStep` имеет один метод `start`, который вызывается платформой, когда пользователи платформы запрашивают запуск приложения.
- Этому приложению требуется один входной параметр под названием `OUTPUT_SIGNAL_ID`. Он представляет собой пользовательский идентификатор сигнала, в который мы записываем значение. Входные параметры указываются в пользовательском интерфейсе платформы, когда пользователи запускают приложение.
- В приложении используется `SignalWriterClient.write_row` метод для написания числового значения (`3.0`) в сигнал с данным идентификатором. Значение написано с помощью метки времени, которая соответствует текущему моменту.

Настройки часового пояса

SDK использует часовой пояс UTC+0 в качестве стандартного часового пояса для всех операций. Следовательно, любые данные `datetime` будут преобразованы из заданного часового пояса в UTC+0 перед записью на Платформу. Аналогично, данные `datetime`, полученные SDK из Платформы, также будут преобразованы в UTC+0.

Важно учитывать конвертацию часового пояса при работе с SDK, особенно при запросе данных сигнала за конкретный период. Например, если пользователь запрашивает гранулярные данные с `05:00 UTC+1 01/01/2023` до `05:00 UTC+1 01/01/2023` для конкретного сигнала, SDK вернёт dataframe с индексом, начинающимся с `04:00 UTC+0 01/01/2023` и заканчивающимся `04:00 UTC+0 01/01/2023`.

requirements.txt

Данный файл необходим для указания зависимостей для корректной работы вашего приложения.

В данном примере этот файл пуст.

Запустите приложение локально

Прежде чем развернуть наше приложение на платформе, давайте запустим его локально.

Для этого нам нужно создать `local_run.py` - вспомогательный скрипт для локального запуска приложения. Этот файл необязателен при развертывании его на платформе.

Следующий фрагмент кода представляет скрипт, который запускает приложение локально. Приложение запускается локально, но подключается к платформе так же, как и если бы

приложение было развернуто на платформе.

```
import argparse
import os

from platform_sdk.runtime.runner.wrapper import run_in_wrapper
from platform_sdk.runtime.utils.config import RuntimeConfiguration

PROJECT_DIR = os.path.dirname(__file__)

import os

# if you don't want to provide the instance URL and credentials via environment variables,
# you can specify them here

# os.environ["AUTH_ENDPOINT"] = "<instalce_url>"
# os.environ["AUTH_USERNAME"] = "<login>"
# os.environ["AUTH_CLIENT_SECRET"] = "<password>"


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--signal_id", type=str, required=True)

    args = parser.parse_args()

    input_params = {"OUTPUT_SIGNAL_ID": args.signal_id}
    job_params = {"MAIN_HANDLER": "entrypoint.BasicApp"}
    run_in_wrapper(
        RuntimeConfiguration(
            task_id=3,
            input_params=input_params,
            job_params=job_params,
        ),
        active_prometheus=False,
        log_level="INFO",
    )


if __name__ == "__main__":
    main()
```

Скрипт `local_run.py` запускает приложение в обертке, которая подключается к платформе. Чтобы подключиться к платформе, вы должны указать URL платформы (например, `qa.example.com`) и данные вашей учетной записи для входа (например, `johndoe` and `pass123`). Вы можете сделать это двумя способами:

- Установите следующие переменные окружения:

  ```
  export AUTH_ENDPOINT=<endpoint>
  export AUTH_USERNAME=<login>
  export AUTH_CLIENT_SECRET=<password>
  ```

- В качестве альтернативы, вы можете установить переменные в самом скрипте:

  ```
  os.environ["AUTH_ENDPOINT"] = "<endpoint>"
  os.environ["AUTH_USERNAME"] = "<login>"
  os.environ["AUTH_CLIENT_SECRET"] = "<password>"
  ```

Чтобы узнать больше о переменных окружающей окружения, пожалуйста, обратитесь к руководству Переменные окружения.

Теперь вы можете запустить приложение. Для этого выполните команду, приведенную ниже. Замените `<signal_id>` на публичный идентификатор сигнала. Если сигнал с данным

идентификатором не существует, он будет создан.

```
python local_run.py --signal_id <signal_id>
```

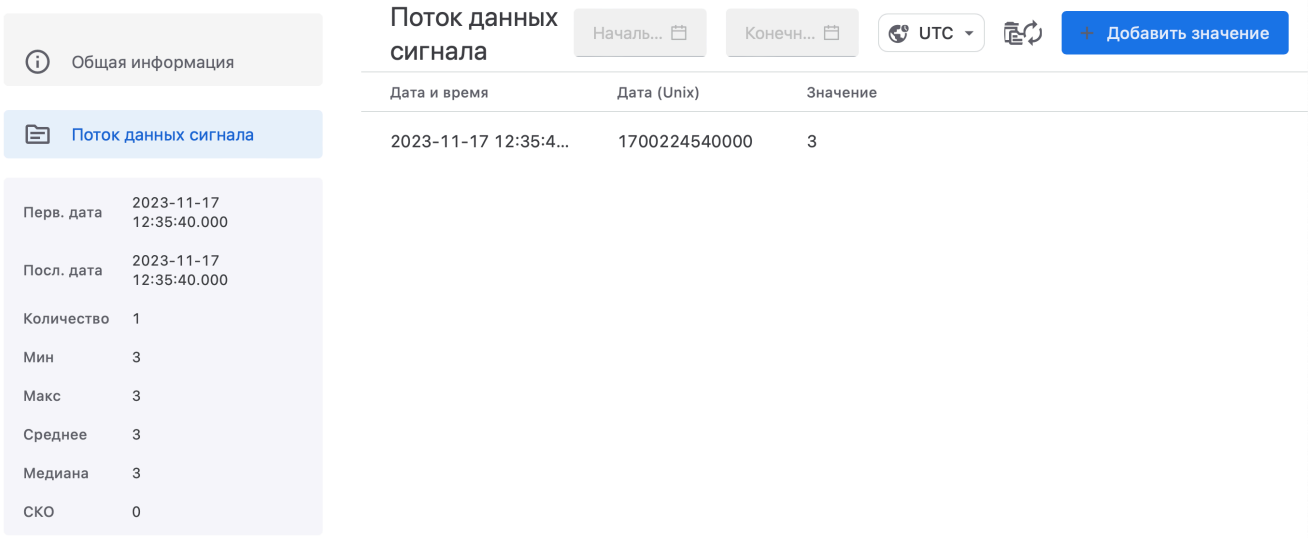
Если приложение завершится успешно, вы должны увидеть вывод, аналогичный следующему:

```
2022-08-16 16:37:56,614 4514 MainProcess platform_sdk.runtime.runner.wrapper:65 INFO     Start job
2022-08-16 16:37:56,620 4514 MainProcess platform_sdk.runtime.runner.wrapper:68 INFO     Join execution
2022-08-16 16:37:58,438 4516 Process-1 platform_sdk.runtime.runner.wrapper:47 INFO     Start handler <entrypoin
2022-08-16 16:37:59,573 4514 MainProcess platform_sdk.runtime.runner.wrapper:70 INFO     Finished with exit cod
```

Проверьте результаты

Теперь вы можете войти на платформу и проверить, что значение было записано в указанный сигнал. Обратите внимание, что отображение значения в пользовательском интерфейсе может занять некоторое время.

- Войдите на платформу и перейдите в раздел **Сигналы**.
- Найдите свой сигнал, вставив публичный идентификатор сигнала в поле поиска.
- Откройте сигнал, нажав на его пользовательский идентификатор, а затем нажмите **Поток данных сигнала**.



Если ваш указанный сигнал отображает установленное значение - это означает, что приложение работает правильно.

Запустите приложение из пользовательского интерфейса платформы.

> **Примечание**
>
> Подробные инструкции по регистрации и запуску приложений на платформе приведены в документации платформы в разделе «Приложения» и «Шаблоны приложений».

Запуск приложения на платформе состоит из трех этапов:

- Опубликовать образ с приложением в реестре Docker

- Создать шаблон приложения
- Создать приложение на основе шаблона

Опубликуйте приложение в реестре Docker

Чтобы развернуть приложение на платформу, вы должны упаковать его в изображение Docker и опубликовать изображение в реестр Docker.

Скопируйте пакет SDK в папку проекта. Затем **создайте Dockerfile** со следующим содержимым:

```
FROM python:3.8.9-slim-buster
WORKDIR /opt/app
ENV PYTHONPATH /opt/app/
ENV PROJECT_DIR /opt/app/
ENV LOGGER_JSON=True

COPY ./platform_sdk-2.40.0-py3-none-any.whl .
RUN pip install platform_sdk-2.40.0-py3-none-any.whl
COPY ./requirements.txt .
RUN pip install --no-cache-dir --ignore-installed --prefer-binary -r requirements.txt
COPY . .

ENTRYPOINT ["runtime"]
```

Структура проекта должна выглядеть следующим образом:

```
my_app
 |--Dockerfile
 |--platform_sdk-2.40.0-py3-none-any.whl
 |--entrypoint.py
 |--requirements.txt
```

Создайте образ. Замените строку `<docker-registry>` фактическим адресом реестра.

```
docker build -f Dockerfile . -t <docker-registry>/example_app --platform=linux/amd64
```

Опубликуйте образ в реестре Docker.

Если образ Docker невозможно отправить непосредственно в реестр с компьютера, на котором он был создан, используйте команды сохранения/загрузки.

```
docker save <docker-registry>/example_app > somefolder/example_app.tar

 ...

docker load --input example_app.tar
```

> ⓘ Примечание
>
> Если вы видите следующую ошибку:
>
> ```
> load 4Error: container create failed: time="2022-10-21T08:42:08Z"
> level=error msg="container_linux.go:380: starting container process
> caused: exec: \"....": executable file not found in $PATH"
> ```
>
> Вы, вероятно, использовали `docker import`, а не `docker load`.

Используйте следующую команду, чтобы отправить образ в реестр:

```
docker push example_app
```

Настройка реестра Docker выходит за рамки данного руководства. Вам следует обратиться к официальной документации реестра, который вы используете. Основные требования:

- Реестр должен быть доступен из сети, где запущен экземпляр платформы и
- Он должен быть настроен, чтобы разрешить скачивание образов без аутентификации.

Как только вы отправите образ в реестр, он станет доступен по адресу `<docker_registry>/example_app:latest`. Теперь вы можете зарегистрировать образ в платформе.

Создайте шаблон приложения

В этом разделе мы создаем шаблон нашего приложения в Платформе.

1. Пройдите в меню **Шаблон приложения**, затем нажмите **Создать шаблон приложения**.
2. В открытом диалоге введите имя для шаблона и нажмите **Создать**. Запомните имя шаблона. Вы будете использовать его позже.
3. Нажмите **Создать шаг приложения** и дайте ему имя, например «Шаг 1». Нажмите **Создать**.
4. Заполните следующие параметры:

   `Параметры процесса выполнения`:

   ```
   {
     "IMAGE_NAME": "<registry_url>/example_app:latest",
     "MAIN_HANDLER": "entrypoint.BasicApp",
     "RESOURCE_TYPE": "STEP_JOB"
   }
   ```

`IMAGE_NAME`	Адрес вашего образа в реестре.
`MAIN_HANDLER`	Класс вашего приложения.
`RESOURCE_TYPE`	Значение `STEP_JOB` указывает, что приложение будет запущено ровно один раз.

`Входные параметры`:

```
{
  "fields": [
    {
      "fieldType": "TEXT",
      "name": "OUTPUT_SIGNAL_ID",
      "label": "The public ID of the output signal.",
      "isRequired": true
    }
  ]
}
```

5. Убедитесь, что ваша конфигурация сохранена успешно.

Создайте и запустите приложение

Создайте приложение, используя шаблон, который вы создали на предыдущем шаге.

1. Пройдите в меню **Приложения**, затем нажмите **Создать приложение**.
2. Задайте имя и в поле **Шаблон** выберите шаблон, который вы создали ранее.
3. Откройте ваше приложение после создания и нажмите **Запустить шаг** из вашего ранее созданного шага.
4. В поле **Окружение** выберите **REMOTE**.
5. В поле **The public ID of the output signal.** введите пользовательский идентификатор сигнала. Вы можете использовать идентификатор существующего или нового сигнала.
6. Нажмите кнопку **Запустить**. Таблица **Все запуски** должна содержать одну запись, соответствующую только что запущенному приложению. Первоначально статус шага должен быть `ОЖИДАНИЕ`, затем он перейдет в `ВЫПОЛНЯЕТСЯ`.
7. Дождитесь изменения статуса на `ЗАВЕРШЕНО`, что означает успешное выполнение приложения. Если приложение перейдет в статус `СБОЙ`, вы можете зайти внутрь шага и изучить логи приложения.

Проверьте результаты

Теперь можно открыть сигнал и убедиться, что значение было записано. Выполните те же действия, что описаны в разделе Проверьте результаты.

Рекомендации

В этом разделе мы рассмотрим некоторые основные примеры использования SDK, что может помочь при разработки вашего приложения.

Входные данные

Потоковая передача гранулярных данных

Рассмотрим сценарий, в котором вам нужно работать с агрегированными данными вместо сырых (raw) данных. В этом случае необходимо использовать метод *subscribe*, чтобы эффективно обрабатывать такой поток данных:

`platform_sdk.integrations.stream_storage.signal.subscription.SignalStreaming.subscribe`

В примере ниже демонстрируется, как использовать метод subscribe на Python:

```
from datetime import timedelta
from uuid import uuid4

from platform_sdk import SignalStreaming

streaming = SignalStreaming()

for snapshot in streaming.subscribe(
    signal_public_ids=["my_signal_1", "my_signal_2"],
    history_size=timedelta(minutes=10),
    granularity=timedelta(minutes=1),
    batch_interval=timedelta(seconds=10),
    consumer_group=f"my_subscription_{uuid4()}",
):
    do_smth(snapshot)
```

Таким образом, будет установлена подписка на конкретный сигнал, что позволяет вашему приложению получать новые данные сигнала.

Потоковые данные с различной гранулярностью

Если ваши данные часто обновляются по одному значению за раз, но сигналы имеют различную гранулярность или несовпадающие временные метки, следует использовать метод *subscribe_to_raw_updates* : `platform_sdk.integrations.stream_storage.signal.subscription.SignalStreaming.subscribe_to_raw_updates`.

Такой подход обеспечивает эффективную обработку сигналов с различной гранулярностью и несовпадающими временными метками. Ниже представлен пример использования этого метода на Python:

```
from datetime import timedelta

from platform_sdk import SignalStreaming

streaming = SignalStreaming()

for snapshot in streaming.subscribe_to_raw_updates(
    signal_public_ids=["my_signal_1", "my_signal_2"],
    history_size=timedelta(seconds=10),
    batch_interval=timedelta(seconds=10),
):
    do_smth(snapshot)
```

Пакетные данные

При работе с данными, которые обновляются пакетами, вы можете использовать потоковую передачу данных на основе подписок. Ключевым моментом является выбор правильного метода на основе гранулярности ваших данных.

Например, если ваше приложение работает с данными без предварительной обработки (т.е. сырыми данными), вам может потребоваться метод `subscribe_to_raw_updates`.

С другой стороны, если вашему приложению требуются гранулярные данные или данные слишком большие и должны быть предварительно обработаны перед чтением – должен быть использован метод subscribe.

Таким образом, ваше приложение можете эффективно обрабатывать данные сигналов, даже и при работе с пакетными данными.

Модели с отслеживанием состояния

При работе с моделями с отслеживанием состояния в вашем приложении машинного обучения важно эффективно хранить состояние модели. Существуют два способа для этой цели: хранение состояний модели в сигнале BLOB или хранение состояния модели в артефактах.

Хранение в сигнале BLOB

Если состояние модели может быть представлено в виде строки (лучше всего подходят маленькие JSON файлы), то его можно хранить в BLOB-сигнале.

В следующем примере демонстрируется, как хранить состояние модели в сигнале BLOB с помощью Python:

```
import datetime
import json

from platform_sdk import SignalsOps, SignalReaderClient, SignalWriterClient

signal_ops = SignalsOps()
writer = SignalWriterClient()
reader = SignalReaderClient()

public_id = "model_state_signal"
signal_ops.create_signal(public_id=public_id, data_stream_type="BLOB")
my_model_current_state = {"scale": 0.3}

while True:
    writer.write_strings(
        public_id=public_id,
        strings=[json.dumps(my_model_current_state)],
        timestamps=[timestamp],
    )

    do_smth()
    latest_model_state = reader.read_latest([public_id], skip_timetsamp=True)
```

Хранение в артефактах

Если ваша модель не может быть представлена как строка или такое представление занимает слишком много памяти, вы можете использовать артефакты в качестве хранилища для любого бинарного файла.

Ниже представлен пример хранения состояния модели в артефакте на Python:

```
import datetime

from platform_sdk import Artifacts

artifacts = Artifacts()

my_model_current_state: bytes = b"something"
artifact_label = "model_state_artifact"
```

```
while True:
    artifacts.upload_no_metadata(label=artifact_label, model_bytes=my_model_current_state)

    do_smth()

    latest_model_state = artifacts.get_latest(artifact_label)
```

Руководства для разработчиков

Этот раздел документации создан для помощи в использовании всех возможностей SDK. Здесь предоставлены подробные руководства, примеры кода и лучшие практики для разработчиков. Представленные руковдства можно использовать как и в создании нового приложения, так и для оптимизиции или расширения ваших готовых решений. Данные руководства также описывают функциональность и все концепты, которые необходимо знать для успешного создания и работы с приложениями.

Используйте оглавление ниже, чтобы перейти к функциональности, которое вас больше всего интересует. Вы также можете следовать всем представленным руководствам, чтобы получить полное представление о всех возможностях SDK.

Если вы только начали работать с вашим первым приложением и хотите получить дополнительный опыт работы с SDK, рекомендуется начать с руководства: *приложение с валидацией параметров*. При достаточном опыте работы с SDK, вы можете перейти к изучению других руководств по вашему усмотрению.

Сигналы (создание/чтение/запись)

В этой серии руководств мы рассмотрим концепцию сигналов, которые представляют собой сущности, содержащие последовательность точек данных и метаданные. Сигналы являются ключевым понятием в работе с платформой, поскольку все данные, обрабатываемые или генерируемые в платформе, заключены в сигналы.

В следующих руководствах мы также рассмотрим, как читать и записывать данные из сигналов, включая работу с числовыми и текстовыми значениями, а также как работать с временной природой сигнальных данных.

Клиенты для работы с сигналами

В этом руководстве вы познакомитесь с основами работы с сигналами при помощи клиентов SDK.

Предварительные условия

- Аккаунт на платформе Тайга.
- Локально установленная библиотека platform SDK.
- Минимальная настройка переменных окружения.

Обзор

Сигнал - это основная сущность платформы Тайга. Сигнал представляет собой временной ряд, т.е. последовательность пар <*время, значение*>. поступающих от датчика или другого

источника данных.

> ⓘ Примечание
>
> Для получения более подробной информации о сигналах обратитесь к документации платформы. Чтобы открыть документацию платформы, войдите в систему, нажмите на значок пользователя в левом нижнем углу страницы и выберите **Документация по платформе**.

SDK имеет 5 клиентов для работы с сигналами, каждый из которых играет определенную роль:

`SignalsOps`	Операции создания, обновления и удаления сигналов.
`SignalsGroupOps`	Работа с группами сигналов.
`SignalReaderClient`	Операции чтения данных.
`SignalWriterClient`	Операции записи данных.
`SignalStreaming`	Подписка на обновления данных.

Группы сигналов и подписка сами по себе являются достаточно сложными темами, и поэтому рассматриваются в отдельных руководствах Клиент для работы с группами сигналов и Клиент подписки соответственно.

Основные методы

Рассмотрим основные методы работы с сигналами через клиенты SDK.

Инициализация клиента

Прежде всего, необходимо инициализировать используемый клиент. Каждый из клиентов может быть инициализирован одним и тем же способом:

```
from platform_sdk import SignalsOps, SignalReaderClient, SignalWriterClient, SignalInputValue

signals_ops = SignalsOps()
signal_reader_client = SignalReaderClient()
signal_writer_client = SignalWriterClient()
```

Создание сигналов

Теперь создадим простой числовой сигнал (т.е. с типом данных `ROW`):

```
public_id = "my_first_signal"
new_signal = signals_ops.create_signal(public_id=public_id, data_stream_type="ROW")
```

Обратите внимание, что по умолчанию новый сигнал всегда создается в основном пространстве. Если необходимо создать сигнал в другом пространстве, просто добавьте

дополнительный параметр `space_public_id` в вызов метода `create_signal`, например:

```
new_signal = signals_ops.create_signal(public_id="my_first_signal", space_public_id="my_custom_space")
```

Если в приведенном выше коде возникает ошибка «User not allowed to create entity in space», проверьте, есть ли у вас права на создание сигнала в этом пространстве. Более подробную информацию о работе с пространствами вы можете получить в документации платформы.

Теперь убедимся, что сигнал создан, запросив его либо по его публичному идентификатору:

```
fetched_signal = signals_ops.fetch_by_public_id(public_id)
print(fetched_signal)
```

либо его числовому ID:

```
fetched_signal = signals_ops.fetch(new_signal.signal_id)
print(fetched_signal)
```

Любой из вариантов даст следующий результат:

```
Signal(
    signal_id=101,
    public_id='my_first_signal',
    data_stream_type=<StreamType.ROW: 'ROW'>,
    label=None, category=None,
    updated_at='2022-09-12T08:21:51.826Z',
    updated_by='user-beta',
    created_at='2022-09-12T08:21:51.656Z',
    created_by='user-beta',
)
```

> ⓘ Примечание
>
> Вы также можете создать новый сигнал сразу внутри группы сигналов Клиент для работы с группами сигналов, передав параметр `group_id` методу `create_signal`. Более подробная информация о группах сигналов приведена в руководстве Клиент для работы с группами сигналов.

Обновление и удаление сигналов

Все объекты, возвращаемые методами `SignalsOps`, являются объектами `Signal`. С их помощью вы можете получить доступ к отдельным свойствам сигнала и изменить их:

```
fetched_signal.label = "another_label"
```

Обратите внимание, что любые изменения, произведенные с объектом сигнала влияют только на локальную копию сигнала. Обновить сигнал на платформе можно при помощи команды:

```
updated_signal = signals_ops.update_signal(fetched_signal)
```

> ⚠ Предупреждение
>
> Вы можете изменить только свойства `label` и `category` существующего сигнала. Изменение остальных свойств будет либо проигнорировано, либо вызовет ошибку.

Чтобы удалить сигнал, выполните команду:

```
signals_ops.remove_signal(updated_signal.signal_id)
```

Можно также удалить сигнал непосредственно по его публичному идентификатору:

```
signals_ops.remove_by_public_id(updated_signal.public_id)
```

Однако из-за дополнительных запросов, выполняемых при удалении сигнала по публичному идентификатору, рекомендуется удалять сигналы таким образом, только если у вас нет доступа к их числовому идентификатору.

Запись значений в сигналы

Сначала создадим новый сигнал:

```
public_id = "signal_for_new_values"
new_signal = signals_ops.create_signal(public_id=public_id, data_stream_type="ROW")
```

Записывать значения в сигнал можно либо по одному, используя объект `SignalInputValue`:

```
from datetime import datetime

signal_writer_client.write_row(
    [SignalInputValue(public_id=public_id, value=1)],
    timestamp_dt=datetime.now(),
)
```

В этом случае столбцы должны совпадать с публичными идентификаторами существующих сигналов, а временные метки должны передаваться в виде индекса или дополнительного столбца. При использовании последнего варианта необходимо также передать имя этого столбца в параметре `timestamp_column`:

```
import pandas as pd
from datetime import datetime

df = pd.DataFrame(
    data={
        "my_row_signal": [1.0, 2.0, 3.0],
        "my_blob_signal": ["foo", "bar", "baz"],
    },
    index=pd.date_range(end=datetime.now(), periods=3, freq="1 s"),
)
signal_writer_client.write(df)
```

С помощью этого метода можно записывать значения сразу в несколько сигналов, включая BLOB-сигналы, просто добавляя дополнительные столбцы в таблицу.

> ⓘ Примечание
>
> Любые числа и булевские значения `True`, `False` рассматриваются как значения ROW сигнала и перед записью конвертируются в числа с плавающей точкой.

Чтение данных из сигналов

Здесь мы считываем данные из сигнала, созданного выше. Сначала убедимся, что сигнал не пуст:

```
print(signal_reader_client.empty([public_id]))
```

Выполнение приведенного выше кода дает следующий результат:

```
{'my_row_signal': False}
```

Аналогичным образом можно запросить минимальную и максимальную временную метку для сигнала или списка сигналов, вызвав методы `get_min_dt` и `get_max_dt` соответственно.

> ⓘ Примечание
>
> При вызове со списком сигналов метод `get_min_dt` возвращает *максимум* среди минимальных временных меток всех переданных сигналов.

Существуют различные варианты чтения данных из сигналов. Наиболее простой из них - чтение сырых данных за определенный период:

```
from datetime import datetime

signal_data = signal_reader_client.read_raw(
    min_dt=datetime(2010, 1, 1),
    max_dt=datetime(2030, 1, 1),
    public_ids=[public_id],
)
print(signal_data)
```

Выполнение приведенного выше кода дает следующий результат:

```
signalPublicId                   signal_for_new_values
timestamp
2022-09-12 13:41:00.928000+00:00                    1.0
2022-09-12 13:44:26.084000+00:00                    2.0
2022-09-13 13:44:26.084000+00:00                    3.0
2022-09-14 13:44:26.084000+00:00                    4.0
```

Также можно считывать данные с заданной гранулярностью (например, 1 день) и интервальным оператором (например, среднее за интервал):

```
from datetime import datetime, timedelta
from platform_sdk import IntervalOperator

signal_data = signal_reader_client.read_without_custom_processors(
    min_dt=datetime(2010, 1, 1),
```

```
    max_dt=datetime(2030, 1, 1),
    public_ids=[public_id],
    granularity=timedelta(days=1),
    interval_op=IntervalOperator.AVG,
)
print(signal_data)
```

Выполнение приведенного выше кода дает следующий результат:

```
signalPublicId                   signal_for_new_values
timestamp
2022-09-12 13:41:00.928000+00:00                    1.5
2022-09-13 13:41:00.928000+00:00                    3.0
2022-09-14 13:41:00.928000+00:00                    4.0
```

Вы также можете указать желаемый тип данных выходной таблицы. Поддерживаемые в настоящее время типы данных перечислены в классе `DType`.

```
from datetime import datetime, timedelta
from platform_sdk import IntervalOperator, DType

signal_data = signal_reader_client.read_without_custom_processors(
    min_dt=datetime(2010, 1, 1),
    max_dt=datetime(2030, 1, 1),
    public_ids=[public_id],
    granularity=timedelta(days=1),
    interval_op=IntervalOperator.AVG,
    dtype=DType.float32
)
print(signal_data.dtypes)
```

Выполнение приведенного выше кода дает следующий результат:

```
signal_for_new_values    float32
```

С помощью методов `read_latest` и `read_latest_by` можно считать только последние значения для указанных сигналов:

```
from datetime import datetime

print("Latest values:")
print(signal_reader_client.read_latest([public_id]))

my_datetime = datetime(2022, 9, 14, 0, 0, 0)
print(f"Latest values at {my_datetime}")
print(signal_reader_client.read_latest_by([public_id], by=my_datetime))
```

Первый метод возвращает последние значения и временные метки для указанных сигналов на момент вызова, а второй - последние значения на заданный момент времени. Выполнение приведенного фрагмента кода дает следующий результат:

```
Latest values:
                       value                         timestamp
signal_for_new_values    4.0   2022-09-14 13:44:26.084000+00:00

Latest values at 2022-09-14 00:00:00:
                       value                         timestamp
signal_for_new_values    3.0   2022-09-13 13:41:00.928000+00:00
```

> ⓘ Примечание
>
> Последнее значение, возвращаемое методом `read_latest_by`, является инклюзивным, т.е. если пользователь запрашивает последнее значение по заданному времени, а в сигнале есть значение с точно таким же временем, то будет возвращено именно это значение.

Смотрите также

Изучите более сложные темы по сигналам:

- Руководство по обработке BLOB-сигналов
- Руководство по группам сигналов
- Руководство по подписке на сигналы

Клиент подписки

В этом руководстве вы узнаете об основах работы с подпиской на сигналы через клиент подписки SDK.

Предварительные условия

- Аккаунт на платформе Тайга.
- Локально установленная библиотека platform SDK.
- Минимальная настройка переменных окружения.
- Если вы еще не работали с сигналами через SDK, то руководство Клиенты для работы с сигналами - отличное место для начала работы.

Обзор

Подобно регулярному чтению, подписка позволяет получать данные из сигналов, но в непрерывном режиме. Подписка получает свежие фрагменты данных (называемые *снапшотами*) в виде `pandas.DataFrame` - в бесконечном цикле, если пользователь не прервет его явно.

В SDK имеется клиент `SignalStreaming` для работы с подпиской на сигналы.

> ⓘ Примечание
>
> К сожалению, в текущей версии SDK поддерживается подписка только на сигналы `ROW`. Если вам необходимо читать значения из сигналов `BLOB`, пожалуйста, рассмотрите возможность использования методов обычного чтения. Есть исключение: в случае подписки на последние значения поддерживаются сигналы как BLOB, так и ROW.

> ⓘ Примечание
>
> Если вы используете пользовательский сертификат, то перед инициализацией клиента подписки необходимо установить переменную окружения `SDK_SUBSCRIPTION_CA_BUNDLE` равной пути к вашему сертификату. В противном случае вы получите множество сообщений об ошибках в журнале `SSL: CERTIFICATE_VERIFY_FAILED`.

Основные методы

Рассмотрим основные методы клиента подписки SDK.

Инициализация клиента

Прежде всего, необходимо инициализировать клиент.

```
from platform_sdk import SignalStreaming

streaming = SignalStreaming()
```

Подписка на обновления сырых данных

Подписка на обновления сырых данных осуществляется с помощью метода `subscribe_to_raw_updates`. Для примера рассмотрим следующий фрагмент:

```
from datetime import timedelta

from platform_sdk import SignalStreaming

streaming = SignalStreaming()

for snapshot in streaming.subscribe_to_raw_updates(
    signal_public_ids=["my_signal"],
    history_size=timedelta(seconds=10),
    batch_interval=timedelta(seconds=10),
):
    print(snapshot)
```

Здесь можно выделить основные параметры:

`signal_public_ids`	Список публичных идентификаторов сигналов, на которые необходимо подписаться. Не должен содержать дубликатов.
`history_size`	Максимальный размер снапшота.
`batch_interval`	Интервал между возвращаемыми снапшотами.

Таким образом, приведенный выше код запросит до 10 секунд самых свежих данных в сигнале `my_signal` как `pandas.DataFrame` и выведет их на экран. Предположим, что сигнал имеет такие значения:

```
                    my_signal
2022-01-01 12:00:01        1.0
2022-01-01 12:00:02        2.0
```

```
2022-01-01 12:00:03        3.0
2022-01-01 12:00:04        4.0
```

Сначала вы получите следующий снапшот:

```
                    my_signal
2022-01-01 12:00:03        3.0
2022-01-01 12:00:04        4.0
```

и затем новый снапшот каждые 10 секунд (или этот же снапшот, если в сигнале нет новых значений).

> ⓘ Примечание
>
> Если в параметре `signal_public_ids` указать несколько сигналов, то результирующий снапшот будет равен объединению значений отдельных сигналов (по аналогии с `OUTER JOIN` в SQL).

Подписка на агрегированные обновления

Иногда могут требоваться данные с определенной аггрегацией или просто не требоваться все значения сигнала как таковые. Вместо этого можно получить данные, сгруппированные по некоторому временному периоду. Для этого подходит метод `subscribe`. У него есть дополнительный параметр `granularity` для управления гранулярностью запрашиваемых данных:

```python
from datetime import timedelta
from uuid import uuid4

from platform_sdk import IntervalOperator, SignalStreaming

streaming = SignalStreaming()

for snapshot in streaming.subscribe(
    signal_public_ids=["my_signal"],
    history_size=timedelta(minutes=10),
    granularity=timedelta(minutes=1),
    batch_interval=timedelta(seconds=10),
    consumer_group=f"my_subscription_{uuid4()}",
    columns_mapping_fill_strategy=DefaultFillStrategy.PREFILL_AVG_REINDEX,
    history_interval_op=IntervalOperator.AVG
):
    print(snapshot)
```

Здесь в качестве окна агрегации выбрана 1 минута. Таким образом, если сигнал имеет следующие значения:

```
                    my_signal
2022-01-01 12:00:30        1.0
2022-01-01 12:01:00       -1.0
2022-01-01 12:01:30        2.0
2022-01-01 12:02:00       -2.0
...
```

Вы получите следующий первый снапшот:

```
                    my_signal
2022-01-01 12:01:00        0.0
```

```
2022-01-01 12:02:00       0.0
```

Параметры `history_interval_op` и `columns_mapping_fill_strategy` управляют агрегацией данных. `history_interval_op` определяет агрегацию для исторических данных, а `columns_mapping_fill_strategy` - для всех новых данных, поступающих по подписке. Более подробную информацию о доступных операторах и стратегиях заполнения можно найти в разделе Агрегация данных.

Подписка только на новые данные

Также могут потребоваться только новые данные, т.е. данные, поступающие между запросами двух снапшотов без объединения с ними исторических данных. В этом случае следует выбрать метод `subscribe_live_data_only`:

```
from datetime import timedelta
from uuid import uuid4

from platform_sdk import SignalStreaming

streaming = SignalStreaming()

for snapshot in streaming.subscribe_live_data_only(
    signal_public_ids=["my_signal"],
    history_size=timedelta(seconds=10),
    granularity=timedelta(seconds=10),
    check_for_new_data_interval=timedelta(seconds=10),
    consumer_group=f"my_subscription_{uuid4()}",
):
    print(snapshot)
```

В этом случае период между двумя снапшотами в основном определяется скоростью поступления данных и может быть больше или равен `check_for_new_data_interval`.

Получение снапшота при появлении нового сообщения

В некоторых случаях может потребоваться получение снапшота при получении нового сообщения по WebSocket сединению. В этом случае можно использовать методы `subscribe_raw_trigger_on_message` и `subscribe_granular_trigger_on_message` для сырых и гранулярных данных соответственно. Обе эти подписки возвращают новый снапшот каждый раз, когда в любой из сигналов в подписке записывается новое значение, но не чаще, чем `check_for_new_data_interval`.

```
from datetime import timedelta
from uuid import uuid4

from platform_sdk import SignalStreaming

streaming = SignalStreaming()

for snapshot in streaming.subscribe_granular_trigger_on_message(
    signal_public_ids=["signal_1", "signal_2"],
    history_size=timedelta(seconds=10),
    granularity=timedelta(seconds=10),
    check_for_new_data_interval=timedelta(seconds=10),
    consumer_group=f"my_subscription_{uuid4()}",
):
    print(snapshot)
```

В этом случае, если сигналы обновляются не одновременно, а последовательно, вы можете, например, получить сначала такой снапшот:

```
  signal_1 signal_2
05:00. 1.        2
05:01. 3.        4
05:02. 5.        nan
05:03  7.        nan
```

а затем следующий снапшот:

```
      signal_1 signal_2
05:00. 1.        2
05:01. 3.        4
05:02. 5.        6
05:03  7.        nan
```

Апериодическая подписка

Некоторые приложения могут требовать снапшоты в нерегулярном режиме. В этом случае проще запрашивать каждый снапшот вручную (как при чтении данных), но пользуясь при этом всеми преимуществами подписки. Подписка с пользовательским расписанием доступна с помощью методов `subscribe_aperiodic` и `subscribe_raw_aperiodic`. В отличие от вышеупомянутых методов подписки, они являются неблокирующими, а позволяют запрашивать снапшот в любое время. Рассмотрим следующий фрагмент:

```python
from datetime import timedelta
from uuid import uuid4

from platform_sdk import SignalStreaming

streaming = SignalStreaming()

subscription = streaming.subscribe_aperiodic(
    signal_public_ids=["my_signal"],
    history_size=timedelta(seconds=10),
    granularity=timedelta(seconds=10),
    consumer_group=f"my_subscription_{uuid4()}",
)
print(subscription.get_snapshot())
subscription.close()
```

Нетрудно видеть, что параметр `batch_interval` отсутствует, так как пакет можно запросить в любое время. Для этого нужно вызвать метод `get_snapshot` возвращаемого объекта `subscription`. Также обратите внимание на вызов `close` в последней строке, закрывающий WebSocket-соединение.

Метод `subscribe_raw_aperiodic` работает аналогично, но вместо гранулярных данных возвращает сырые сигнальные данные без какой-либо предварительной обработки.

> ⚠ Предупреждение
>
> Пожалуйста, убедитесь, что вы **всегда** вызываете метод `close`, когда вам больше не требуются новые снапшоты или в конце вашей программы.

Эмуляция подписки при помощи чтения

Иногда чтение данных (через HTTP-соединение) предпочтительнее подписки (через WebSocket-соединение). Метод `subscribe_no_ws` класса `SignalReaderClient` предоставляет интерфейс, похожий на подписку, но использует чтение для получения данных:

```
from datetime import timedelta

from platform_sdk import SignalReaderClient

reader = SignalReaderClient()

for snapshot in reader.subscribe_no_ws(
    signal_public_ids=["my_signal"],
    history_size=timedelta(seconds=10),
    batch_interval=timedelta(seconds=10),
    granularity=None,
):
    print(snapshot)
```

Обратите внимание, что параметр `granularity` является необязательным. Если он не указан или явно установлен равным `None`, то клиент ведет себя как обычная подписка, в противном случае как подписка с аггрегацией.

Подписка на последние значения сигналов

Если вы хотите получать только последнее значение для каждого из сигналов, то можете использовать `subscribe_last_value`. Вы будете получать новый снапшот каждый раз, когда хотя бы для одного сигнала изменится последняя метка времени. Каждый раз вы будете получать таблицу следующего вида:

```
                 timestamp                  value
public_id_1     2022-05-04 18:53:08+00:00    35
public_id_2     2022-05-04 18:53:08+00:00    76
```

Пример кода:

```
from datetime import timedelta
from uuid import uuid4

from platform_sdk import SignalStreaming

streaming = SignalStreaming()

for snapshot in streaming.subscribe_last_value(
    signal_public_ids=["my_signal"],
    check_for_new_data_interval=timedelta(seconds=10),
    consumer_group=f"my_subscription_{uuid4()}",
):
    print(snapshot)
```

В этом случае период между двумя запросами снапшотов в основном определяется скоростью поступления данных и может быть больше или равен `check_for_new_data_interval`.

> ⓘ Примечание
>
> По умолчанию метод `subscribe_last_value` возвращает новый снапшот только в том случае, если хотя бы один из сигналов был обновлен. Однако если ваше приложение требует регулярные снапшоты, вы можете установить флаг `always` равным `True`.

Апериодическая подписка на последние значения сигналов

Если вы хотите постоянно получать одно последнее значение для каждого из сигналов, но у вас нет регулярного периода для получения последних значений, вы можете использовать метод `subscribe_last_value_aperiodic`, возвращающий новый снапшот по запросу. Снапшот имеет следующий вид:

```
                 timestamp                  value
public_id_1      2022-05-04 18:53:08+00:00    35
public_id_2      2022-05-04 18:53:08+00:00    76
```

Пример кода:

```
from datetime import timedelta
from uuid import uuid4

from platform_sdk import SignalStreaming

streaming = SignalStreaming()

obj = streaming.subscribe_last_value_aperiodic(
    signal_public_ids=["my_signal"],
    consumer_group=f"my_subscription_{uuid4()}",
)
print(obj.get_snapshot()
```

В этом случае *get_snapshot* вернет текущие последние значения указанных сигналов.

Устранение неисправностей, связанных с зависанием подписки

Подписка предназначена для регулярного потребления потока данных. При использовании перечисленных выше методов подписки следует ожидать получения обновленных снапшотов при каждой итерации. Если вы получаете десятки одинаковых снапшотов подряд, это может указывать на проблемы с вашими данными или платформой.

Для контроля потока данных методы подписки могут выдавать предупреждения или завершаться с ошибкой в случае, если новые данные не поступают в течение некоторого количества итераций. Каждый метод `SignalReaderClient` имеет два следующих необязательных параметра:

1. `freeze_warning_every` - определяет, через сколько итераций без обновлений подписка будет выдавать предупреждение.
2. `freeze_failing_after_iters` - определяет, через сколько итераций без обновлений подписка завершится с ошибкой.

Подробное описание приведено ниже.

Предупреждения о зависании подписки

Для настройки предупреждений при зависании подписки необходимо указать параметр `freeze_warning_every`. Этот параметр определяет, какое количество итераций без обновления данных считается нормальным для конкретного случая использования. Значение по умолчанию равно 20 итерациям, что обычно позволяет избежать излишних предупреждений, но мы настоятельно рекомендуем настроить его в соответствии с вашими потребностями.

Если в течение определенного количества последовательных итераций не происходит обновления данных, то выводится предупреждающее сообщение, которое отображается в логах приложения и имеет следующий формат:

```
No new subscription data for more than 20 iterations (max datetime in buffer 2022-01-01 12:00:00).
If this is expected, you can increase the 'freeze_warning_every' parameter to suppress this warning.
```

Если с момента начала подписки не приходило никаких данных, то сообщение будет выглядеть следующим образом:

```
Subscription has not received any data since it started (max datetime in buffer 2022-01-01 12:00:00).
Please make sure that your signals are receiving data.
```

Завершение с ошибкой при зависании подписки

Если вы хотите, чтобы в случае предположительного зависания подписки ваше приложение завершалось с ошибкой, необходимо указать параметр `freeze_failing_after_iters`. Этот параметр определяет, когда подписка должна завершаться с ошибкой `SubscriptionFreezingError`, в случае отсутствия новых данных.

Для методов `subscribe_aperiodic`, `subscribe`, `subscribe_to_raw_updates`, `subscribe_live_data_only`, `subscribe_last_value` и `subscribe_last_value_aperiodic` необходимо указать количество итераций (целое число) `freeze_failing_after_iters`:

```python
from datetime import timedelta
from uuid import uuid4

from platform_sdk import SignalStreaming

streaming = SignalStreaming()

subscription = streaming.subscribe_aperiodic(
    signal_public_ids=["my_signal"],
    history_size=timedelta(seconds=10),
    granularity=timedelta(seconds=10),
    freeze_failing_after_iters=50,
    consumer_group=f"my_subscription_{uuid4()}",
)
print(subscription.get_snapshot())
subscription.close()
```

Этот код завершится с ошибкой `SubscriptionFreezingError` после 50 итераций без появления новых данных.

Примечания

Ниже в произвольном порядке приводится список моментов, на которые следует обратить внимание при работе с подпиской, включая возможные трудности, особенности логики работы с подпиской и краевые случаи.

Первый снапшот

Первый полученный снапшот в большинстве случаев содержит только прошлые (или исторические) данные в указанных сигналах. Размер снапшота всегда равен параметру `history_size` (если он задан). Этот снапшот заканчивается последней временной меткой и

значениями по всем указанным сигналам. Это означает, что если указанные сигналы не обновлялись некоторое время, то первый полученный снапшот может быть достаточно далеко в прошлом, даже если размер истории относительно небольшой. Например, первый снапшот может содержать только данные двухдневной давности, даже если размер истории составляет всего пару минут.

Однако по мере поступления данных вы начнете получать снапшоты, близкие к текущему моменту времени.

Следующие шаги

Чтобы получить практический опыт работы с подпиской, попробуйте запустить приложение Плеера и проделать все вышеописанные шаги с реальными данными.

Обработка BLOB-сигналов

В этом руководстве вы получите более глубокое представление о работе с BLOB-сигналами и их основных отличиях от ROW-сигналов. В первой части руководства вы получите полный обзор типов данных, поддерживаемых в SDK. Вторая часть представляет собой краткое рассмотрение методов, предназначенных только для BLOB сигналов.

Предварительные условия

Данное руководство является продолжением предыдущего руководства Клиенты для работы с сигналами. Убедитесь, что вы прочитали его, прежде чем двигаться дальше.

- Аккаунт на платформе Тайга.
- Локально установленная библиотека platform SDK.
- Минимальная настройка переменных окружения.

Обзор

BLOB-сигналы содержат текст в качестве значений. Как и сигнаы ROW, они позволяют привязать некоторые данные к определенному моменту времени, но данные в этом случае могут быть более сложными, включая JSON строки, логи или любую другую информацию в человеко-читаемом формате.

> Предупреждение
>
> В настоящее время одно значение BLOB ограничено 10 000 символами. Убедитесь, что ваши данные соответствуют этому требованию.

> ⓘ Примечание
>
> Для получения более подробной информации о сигналах обратитесь к документации платформы. Чтобы открыть документацию платформы, войдите в систему, нажмите на значок пользователя в левом нижнем углу страницы и выберите **Документация по платформе**.

Некоторые операции с BLOB-сигналами могут выполняться только через специальные интерфейсы, особенно когда речь идет о чтении и записи данных. Поэтому работа с BLOB-сигналами может быть особенно утомительной на начальном этапе, и это руководство призвано сгладить углы.

Поддержка типов данных

Ниже приведен полный список поддерживаемых типов данных (т.е. ROW или BLOB) для всех публичных методов сигналов.

Клиенты `SignalsOps` (работает с сущностями сигналов) и `SignalsGroupOps` (работает с группами сигналов) без проблем поддерживают оба типа данных, то клиент подписки `SignalStreaming` доступен только для сигналов ROW, за исключением подписки на последние значения. Подписка на последние значения *subscribe_last_value* и *subscribe_last_value_aperiodic* работает как для BLOB, так и для ROW сигналов.

Поддержка типов данных для операций чтения приведена ниже:

Метод	ROW	BLOB
`read`	да	нет
`read_without_custom_processors`	да	нет
`read_without_prefill`	да	нет
`read_latest`	да	да
`read_latest_by`	да	да
`read_raw`	да	нет
`read_raw_blob`	да	да
`subscribe_no_ws`	да	нет
`get_latest_available_dt`	да	да
`get_min_dt`, `get_max_dt`, и `empty`	да	да

> ⓘ Примечание
>
> Все вышеперечисленные методы, поддерживающие сигналы ROW и BLOB, могут обрабатывать их в одном запросе.

Поддержка типов данных для операций записи:

Метод	ROW	BLOB
`write`	да	да
`write_row`	да	нет
`write_strings`	нет	да

Методы для работы с BLOB сигналами

Ниже приведен краткий обзор методов работы с BLOB сигналами, подчеркивающий отличия в рабочем процессе.

Создание и обновление BLOB сигналов

Создать и обновить ROW и BLOB сигналы можно при помощи одних и тех же методов (они подробно описаны в разделе Клиенты для работы с сигналами). Однако при создании BLOB-сигнала необходимо обязательно передать параметр `data_stream_type` (по умолчанию равный `ROW`):

```
from platform_sdk import SignalsOps

signals_ops = SignalsOps()
my_blob_signal = signals_ops.create_signal(
    public_id="my_blob_signal",
    data_stream_type="BLOB",  # don't omit this parameter
)
```

Обратите внимание, что нельзя изменить тип данных существующего сигнала (например, сделать ROW сигнал BLOB сигналом и наоборот), даже если он пуст.

Запись значений в BLOB сигналы

Запись значений в BLOB-сигналы осуществляется с помощью методов `write` и `write_strings` класса `SignalWriterClient`. Использование первого метода рассматривается в разделе Запись значений в сигналы руководства Клиенты для работы с сигналами. Второй метод может быть использован для записи нескольких значений в один BLOB-сигнал за один раз:

```
from datetime import datetime

from platform_sdk import SignalWriterClient

writer = SignalWriterClient()
writer.write_strings(
```

```
    public_id="my_blob_signal",
    strings=['{"Hello": "World"}', "Hi!"],
    timestamps=[datetime(2022, 1, 1), datetime(2023, 1, 1)]
)
```

Убедитесь, что длина обоих параметров `strings` и `timestamps` одинакова.

Несмотря на то что сигналы BLOB позволяют хранить не только строки, данные всегда хранятся в виде строк (даже если это валидные JSON строки). Это означает, что все, что вы записываете в BLOB сигнал, должно быть сначала преобразовано в строку.

Чтение из BLOB сигналов

Чтение последних значений доступно с помощью методов `read_latest` и `read_latest_by` соответственно, которые работают и с ROW, и с BLOB сигналами. Оба метода подробно рассмотрены в части Чтение данных из сигналов руководства Клиенты для работы с сигналами.

Поскольку агрегация по времени не имеет особого смысла, когда речь идет о текстовых данных, единственным доступным методом чтения данных из BLOB сигналов является метод `read_raw_blob`:

```
from datetime import datetime

signal_data = signal_reader_client.read_raw_blob(
    min_dt=datetime(2010, 1, 1),
    max_dt=datetime(2030, 1, 1),
    public_ids=[public_id],
)
print(signal_data)
```

Выполнение приведенного выше кода дает следующий результат:

```
signalPublicId                          my_blob_signal
timestamp
2022-01-01 00:00:00.000000+00:00   '{"Hello": "World"}'
2023-01-01 00:00:00.000000+00:00                'Hi!'
```

Стоит отметить, что хотя некоторые из возвращаемых значений являются вполне валидными JSON строками (например, первая строка в приведенном выше выводе), они всегда возвращаются в виде строк. Вы можете применить дополнительный парсинг или постобработку самостоятельно.

Смотрите также

Изучите другие темы по сигналам:

- Пересмотрите базовое руководство по сигналам
- Руководство по группам сигналов
- Руководство по подписке на сигналы

Клиент для работы с группами сигналов

В этом руководстве вы познакомитесь с основами работы с группами сигналов при помощи клиента групп сигналов в SDK.

Предварительные условия

- Аккаунт на платформе Тайга.
- Локально установленная библиотека platform SDK.
- Минимальная настройка переменных окружения.

Обзор

В большинстве реальных сценариев ваши приложения должны оперировать десятками сигналов. Изменения в таком количестве сигналов сложно отслеживать. Группы сигналов добавляют новый уровень абстракции, позволяя объединять все сигналы и определять удобные операции для работы над этими группами.

В SDK имеется клиент `SignalsGroupOps` для работы с группами сигналов.

> ⓘ Примечание
>
> Для получения более подробной информации о сигналах и группах сигналов обратитесь к документации платформы. Чтобы открыть документацию платформы, войдите в систему, нажмите на значок пользователя в левом нижнем углу страницы и выберите **Документация по платформе**.

Основные методы

В этом разделе рассматривается интерфейс клиента SDK для работы с сигналами `SignalsGroupOps` и его основные методы.

Инициализация клиента

Прежде всего, необходимо инициализировать клиент `SignalsGroupOps`:

```
from platform_sdk import SignalsGroupOps

signals_group_ops = SignalsGroupOps()
```

Создать новую группу

Для `создания` новой группы сигналов необходимо указать ее имя и необязательный уникальный публичный идентификатор:

```
my_signal_group = signals_group_ops.create(name="my-signal-group", public_id="my-signal-group-1")
print(signal_group)
```

Выполнение приведенного выше кода дает следующий результат:

```
SignalGroup(
    id_=123,
    name='my-signal-group',
    public_id='my-signal-group-1'
    ...
)
```

> ⓘ Примечание
>
> Обратите внимание, что имя группы должно быть уникальным.

Если вам необходимо указать публичный идентификатор группы, используйте параметр `space_public_id`

```
my_signal_group = signals_group_ops.create(name="my-signal-group", space_public_id="my-space")
print(signal_group)
```

Убедиться в том, что группа создана, можно, вызвав метод `exists`:

```
print("Exists:", signals_group_ops.exists(my_signal_group.name))
```

Чтобы запросить созданный объект `SignalGroup` по его идентификатору, вызовите метод `get`:

```
print("You can get group both by id:", signals_group_ops.get(my_signal_group.id_))
```

В качестве альтернативы можно запросить группу сигналов по ее имени, вызвав метод `get_by_name`:

```
print("And by name:", signals_group_ops.get_by_name(my_signal_group.name))
```

Или по его `public_id`

```
print("And by public_id:", signals_group_ops.get_by_public_id(my_signal_group.public_id))
```

> ⓘ Примечание
>
> Заметим, что метод `get()` быстрее, поэтому метод `get_by_name` рекомендуется использовать только в том случае, если у вас нет доступа к ID группы.

Обновление и удаление группы

Вы также можете обновить имя существующей группы сигналов, вызвав метод `update_name`:

```
my_signal_group = signals_group_ops.update_name(my_signal_group.id_, new_name="my-new-group")
print("Name has been changed to:", my_signal_group.name)
```

Выполнение приведенного выше кода дает следующий результат:

```
Name has been changed to: my-new-group
```

Если необходимо удалить группу, вызовите метод `delete` с соответствующим идентификатором группы:

```
signals_group_ops.delete(my_signal_group.id_)
```

`SignalsGroupOps` также поддерживает массовое удаление. Рассмотрим следующий фрагмент:

```
ids = []

for i in range(10):
    signal_group = signals_group_ops.create(name=f"my-signal-group-{i}")
    ids.append(signal_group.id_)

signals_group_ops.delete_bulk(ids=ids)
```

Приведенный выше код создает 10 групп и удаляет их все одним вызовом `delete_bulk`.

Добавление сигналов в группу

До сих пор мы работали только с пустыми группами сигналов. Добавление сигналов в группы (как по одному, так и по несколько сигналов за раз) поддерживается методом `attach_to_group` клиента `SignalsOps`:

```
from platform_sdk import SignalsOps

signals_ops = SignalsOps()

my_signal_group = signals_group_ops.create(name="my-signal-group")
my_signal = signals_ops.create_signal(public_id="my-signal")
signals_ops.attach_to_group(public_ids=[my_signal.public_id], group_name=my_signal_group.name)
```

Теперь вы можете `запросить` свой сигнал и убедиться, что он имеет непустое свойство `groups`:

```
print("Note the 'groups' property:", signals_ops.fetch(my_signal.signal_id))
```

Выполнение приведенного выше кода дает следующий результат:

```
Note the 'groups' property: Signal(
    signal_id=101,
    public_id='my-signal',
    groups=[
        SignalGroup(
            id_=123,
            name='my-signal-group',
            ...
        )
    ],
    ...
)
```

Обратите внимание, что сигнал можно добавить и в несколько групп:

```
my_other_signal_group = signals_group_ops.create(name="my-other-signal-group")
signals_ops.attach_to_group(public_ids=[my_signal.public_id], group_name=my_other_signal_group.name)
```

> ⓘ Примечание
>
> Более подробно о клиенте `SignalsOps` и работе с сигналами через SDK можно узнать из руководства Клиенты для работы с сигналами.

Запросить все сигналы в группе

Метод `list_signals_by_group` является крайне удобным, поскольку позволяет запросить информацию обо *всех сигналах* в указанной группе:

```
print("All signals in my group:", signals_group_ops.list_signals_by_group(my_signal_group.id_))
```

Убедитесь, что в списке есть ваш сигнал:

```
All signals in my group: [Signal(signal_id=101, public_id='my-signal', ...)]
```

Теперь уберем сигнал из группы с помощью метода `detach_from_group:` (отметим, что это можно сделать и с несколькими сигналами одновременно, добавив в список дополнительные идентификаторы сигналов):

```
signals_ops.detach_from_group(public_ids=[my_signal.public_id], group_name=my_signal_group.name)
print("All signals in my group:", signals_group_ops.list_signals_by_group(my_signal_group.name))
```

Убедитесь, что теперь список пуст:

```
All signals in my group: []
```

> ⓘ Примечание
>
> В качестве альтернативы можно вызвать все описанные выше методы (т.е. `attach_to_group`, `detach_from_group:` и `list_signals_by_group`) с аргументом `group_id` вместо `group_name`.

Полный пример

Полный пример кода приведен ниже:

```
# Imports

from platform_sdk import SignalsOps, SignalsGroupOps

# Initialize the client

signals_group_ops = SignalsGroupOps()
signals_ops = SignalsOps()

# Create a new group
my_signal_group = signals_group_ops.create(name="my-signal-group")

print("Exists:", signals_group_ops.exists(my_signal_group.name))
print("You can get group both by id:", signals_group_ops.get(my_signal_group.id_))
print("And by name:", signals_group_ops.get_by_name(my_signal_group.name))
```

```python
# Update/delete the group

my_signal_group = signals_group_ops.update_name(my_signal_group.id_, new_name="my-new-group")
print("Name has been changed to:", my_signal_group.name)

signals_group_ops.delete(my_signal_group.id_)

ids = []

for i in range(10):
    signal_group = signals_group_ops.create(name=f"my-signal-group-{i}")
    ids.append(signal_group.id_)

signals_group_ops.delete_bulk(ids=ids)

# Attach signal to the group

my_signal_group = signals_group_ops.create(name="my-signal-group")
my_signal = signals_ops.create_signal(public_id="my-signal")
signals_ops.attach_to_group(public_ids=[my_signal.public_id], group_name=my_signal_group.name)

print("Note the 'groups' property:", signals_ops.fetch(my_signal.signal_id))

# Get all signals attached to the group

print("All signals in my group:", signals_group_ops.list_signals_by_group(my_signal_group.name))

signals_ops.detach_from_group(public_ids=[my_signal.public_id], group_name=my_signal_group.name)
print("All signals in my group:", signals_group_ops.list_signals_by_group(my_signal_group.name))
```

Смотрите также

Если вы пропустили предыдущие руководства по сигналам или хотите получить более подробную информацию:

- Основное руководство для клиентов по сигналам
- Руководство по подписке на сигналы

Агрегация данных

В этом руководстве мы рассмотрим, как более эффективно и организованно работать с временными рядами, используя интервальные операторы и стратегии заполнения. При работе с временными рядами, важно понимать, как эффективно манипулировать ими и осуществлять преобразования. Интервальные операторы и стратегии заполнения представляют собой удобные инструменты, позволяющие выполнять различные операции с данными и обрабатывать пропуски.

В этом руководстве мы рассмотрим основы интервальных операторов и стратегий заполнения, а также приведем практические примеры, которые помогут вам начать применять их к своим данным.

Операторы

При обработке сигналов обычно требуется анализировать сигналы по дискретным сегментам или интервалам. Класс `IntervalOperator` содержит доступные интервальные операторы: `AVG`, `MIN`, `MAX`, `MEDIAN`, `LAST`. Эти операторы используются для агрегирования данных в пределах фиксированного интервала при получении данных из сигналов при помощи клиентов `SignalReaderClient` или `SignalStreaming`.

Использование с SignalReaderClient

Для задания агрегации при чтении данных необходимо определить параметр `interval_op`:

```
from platform_sdk import SignalReaderClient, IntervalOperator

signal_reader_client = SignalReaderClient()

signal_data = signal_reader_client.read_without_custom_processors(
    min_dt=datetime(2010, 1, 1),
    max_dt=datetime(2030, 1, 1),
    public_ids=[public_id],
    granularity=timedelta(days=1),
    interval_op=IntervalOperator.MAX
)
print(signal_data)
```

Использование с подпиской

Соответствующий параметр для подписки называется `history_interval_op`. Этот оператор применяется только к историческим данным. Для всех свежих данных, поступающих при работе подписки, тип агрегации задается при помощи параметра `columns_mapping_fill_strategy`.

```
from datetime import timedelta
from uuid import uuid4

from platform_sdk import IntervalOperator,  SignalStreaming

streaming = SignalStreaming()

for snapshot in streaming.subscribe(
    signal_public_ids=["my_signal"],
    history_size=timedelta(minutes=10),
    granularity=timedelta(minutes=1),
    batch_interval=timedelta(seconds=10),
    consumer_group=f"my_subscription_{uuid4()}",
    history_interval_op=IntervalOperator.MIN,
    columns_mapping_fill_strategy={"my_signal": DefaultFillStrategy.PREFILL_MIN_REINDEX},
):
    print(snapshot)
```

Доступные операторы

Ниже приведен обзор доступных интервальных операторов:

- `AVG`: вычисляет среднее значение точек данных в интервале.

Исходный сигнал

datetime	values
2023-01-01 10:00:00	1.0
2023-01-01 10:01:00	2.0
2023-01-01 10:02:00	3.0
2023-01-01 10:03:00	4.0
2023-01-01 10:04:00	5.0
2023-01-01 10:05:00	6.0
2023-01-01 10:06:00	7.0
2023-01-01 10:07:00	8.0
2023-01-01 10:08:00	9.0

С оператором, гранулярность 2 мин

datetime	values
2023-01-01 10:00:00	1.5
2023-01-01 10:02:00	3.5
2023-01-01 10:04:00	5.5
2023-01-01 10:06:00	7.5
2023-01-01 10:08:00	9.0

Исходный сигнал

datetime	values
2023-01-01 10:00:00	1.0
2023-01-01 10:01:00	2.0
2023-01-01 10:02:00	3.0
2023-01-01 10:03:00	4.0
2023-01-01 10:04:00	5.0
2023-01-01 10:05:00	6.0
2023-01-01 10:06:00	7.0
2023-01-01 10:07:00	8.0
2023-01-01 10:08:00	9.0

С оператором, гранулярность 3 мин

datetime	values
2023-01-01 10:00:00	2.0
2023-01-01 10:03:00	5.0
2023-01-01 10:06:00	7.0

- `MIN`: Вычисляет минимальное значение точек данных в интервале.

Исходный сигнал

datetime	values
2023-01-01 10:00:00	1.0
2023-01-01 10:01:00	2.0
2023-01-01 10:02:00	3.0
2023-01-01 10:03:00	4.0
2023-01-01 10:04:00	5.0
2023-01-01 10:05:00	6.0
2023-01-01 10:06:00	7.0
2023-01-01 10:07:00	8.0
2023-01-01 10:08:00	9.0

С оператором, гранулярность 2 мин

datetime	values
2023-01-01 10:00:00	1.0
2023-01-01 10:02:00	3.0
2023-01-01 10:04:00	5.0
2023-01-01 10:06:00	7.0
2023-01-01 10:08:00	9.0

Исходный сигнал

datetime	values
2023-01-01 10:00:00	1.0
2023-01-01 10:01:00	2.0
2023-01-01 10:02:00	3.0
2023-01-01 10:03:00	4.0
2023-01-01 10:04:00	5.0
2023-01-01 10:05:00	6.0
2023-01-01 10:06:00	7.0

С оператором, гранулярность 3 мин

datetime	values
2023-01-01 10:00:00	1.0
2023-01-01 10:03:00	4.0
2023-01-01 10:06:00	7.0

Исходный сигнал | **С оператором, гранулярность 3 мин**

```
2023-01-01 10:07:00     8.0
2023-01-01 10:08:00     9.0
```

- `MAX`: вычисляет максимальное значение точек данных в пределах интервала.

Исходный сигнал | **С оператором, гранулярность 2 мин**

```
datetime                values
2023-01-01 10:00:00     1.0
2023-01-01 10:01:00     2.0
2023-01-01 10:02:00     3.0
2023-01-01 10:03:00     4.0
2023-01-01 10:04:00     5.0
2023-01-01 10:05:00     6.0
2023-01-01 10:06:00     7.0
2023-01-01 10:07:00     8.0
2023-01-01 10:08:00     9.0
```

```
datetime                values
2023-01-01 10:00:00     2.0
2023-01-01 10:02:00     4.0
2023-01-01 10:04:00     6.0
2023-01-01 10:06:00     8.0
2023-01-01 10:08:00     9.0
```

Исходный сигнал | **С оператором, гранулярность 3 мин**

```
datetime                values
2023-01-01 10:00:00     1.0
2023-01-01 10:01:00     2.0
2023-01-01 10:02:00     3.0
2023-01-01 10:03:00     4.0
2023-01-01 10:04:00     5.0
2023-01-01 10:05:00     6.0
2023-01-01 10:06:00     7.0
2023-01-01 10:07:00     8.0
2023-01-01 10:08:00     9.0
```

```
datetime                values
2023-01-01 10:00:00     3.0
2023-01-01 10:03:00     6.0
2023-01-01 10:06:00     7.0
```

- `MEDIAN`: Вычисляет медианное значение точек данных в интервале.

Исходный сигнал | **С оператором, гранулярность 3 мин**

```
datetime                values
2023-01-01 10:00:00     1.0
2023-01-01 10:01:00     2.0
2023-01-01 10:02:00     3.0
2023-01-01 10:03:00     4.0
2023-01-01 10:04:00     5.0
2023-01-01 10:05:00     6.0
2023-01-01 10:06:00     7.0
2023-01-01 10:07:00     8.0
2023-01-01 10:08:00     9.0
```

```
datetime                values
2023-01-01 10:00:00     2.0
2023-01-01 10:03:00     5.0
2023-01-01 10:06:00     7.0
```

Исходный сигнал | **С оператором, гранулярность 3 мин**

Исходный сигнал

datetime	values
2023-01-02 10:00:00	1.0
2023-01-02 10:01:00	6.0
2023-01-02 10:04:00	7.0
2023-01-02 10:05:00	12.0
2023-01-02 10:06:00	2.0
2023-01-02 10:08:00	1.0
2023-01-02 10:11:00	3.0

С оператором, гранулярность 3 мин

datetime	values
2023-01-02 10:00:00	3.5
2023-01-02 10:03:00	3.5
2023-01-02 10:06:00	1.5
2023-01-02 10:09:00	1.5

Исходный сигнал

datetime	values
2023-01-02 10:00:00	1.0
2023-01-02 10:01:00	6.0
2023-01-02 10:04:00	7.0
2023-01-02 10:05:00	12.0
2023-01-02 10:06:00	2.0
2023-01-02 10:08:00	1.0
2023-01-02 10:11:00	3.0

С оператором, гранулярность 4 мин

datetime	values
2023-01-02 10:00:00	3.5
2023-01-02 10:04:00	7.0
2023-01-02 10:08:00	1.0

- `LAST`: выбирается последняя точка данных в интервале.

Исходный сигнал

datetime	values
2023-01-01 10:00:00	1.0
2023-01-01 10:01:00	2.0
2023-01-01 10:02:00	3.0
2023-01-01 10:03:00	4.0
2023-01-01 10:04:00	5.0
2023-01-01 10:05:00	6.0
2023-01-01 10:06:00	7.0
2023-01-01 10:07:00	8.0
2023-01-01 10:08:00	9.0

С оператором, гранулярность 2 мин

datetime	values
2023-01-01 10:00:00	2.0
2023-01-01 10:02:00	4.0
2023-01-01 10:04:00	6.0
2023-01-01 10:06:00	8.0
2023-01-01 10:08:00	9.0

Исходный сигнал

datetime	values
2023-01-01 10:00:00	1.0
2023-01-01 10:01:00	2.0
2023-01-01 10:02:00	3.0
2023-01-01 10:03:00	4.0
2023-01-01 10:04:00	5.0
2023-01-01 10:05:00	6.0
2023-01-01 10:06:00	7.0
2023-01-01 10:07:00	8.0
2023-01-01 10:08:00	9.0

С оператором, гранулярность 3 мин

datetime	values
2023-01-01 10:00:00	3.0
2023-01-01 10:03:00	6.0
2023-01-01 10:06:00	7.0

Данные в сигналах часто содержат пропуски или поступают с разной гранулярностью, что затрудняет их анализ. Стратегия заполнения - это метод восполнения недостающих точек данных в сигнале.

Использование с подпиской

```
from datetime import timedelta
from uuid import uuid4

from platform_sdk import IntervalOperator,  SignalStreaming

streaming = SignalStreaming()

for snapshot in streaming.subscribe(
    signal_public_ids=["my_signal"],
    history_size=timedelta(minutes=10),
    granularity=timedelta(minutes=1),
    batch_interval=timedelta(seconds=10),
    consumer_group=f"my_subscription_{uuid4()}",
    columns_mapping_fill_strategy={
                    "my_signal": DefaultFillStrategy.PREFILL_MIN_REINDEX
            },
):
    print(snapshot)
```

Для параметра `columns_mapping_fill_strategy` можно указать либо словарь, сопоставляющий имена столбцов с определенными стратегиями заполнения, либо объект `defaultdict`, задающий единую стратегию для всех сигналов сразу. Все приведенные ниже примеры являются корректными:

```
from collections import defaultdict

# Apply 'MIN' resampling for the first signal and 'MAX' for the second one
# Note that you must list all signals this way
columns_mapping_fill_strategy = {
        "signal1": DefaultFillStrategy.PREFILL_MIN_REINDEX,
        "signal2": DefaultFillStrategy.PREFILL_MAX_REINDEX,
}

# Apply 'AVG' resampling to all signals
columns_mapping_fill_strategy = defaultdict(
        lambda: DefaultFillStrategy.PREFILL_AVG_REINDEX
)
```

Если задан параметр `columns_mapping_fill_strategy`, но не задан параметр `interval_op`, он будет выбран из списка стратегий заполнения:

```
DefaultFillStrategy.PREFILL_AVG_REINDEX: IntervalOperator.AVG,
DefaultFillStrategy.PREFILL_MIN_REINDEX: IntervalOperator.MIN,
DefaultFillStrategy.PREFILL_MAX_REINDEX: IntervalOperator.MAX,
DefaultFillStrategy.PREFILL_MEDIAN_REINDEX: IntervalOperator.MEDIAN,
DefaultFillStrategy.PREFILL_LAST_REINDEX: IntervalOperator.LAST,
```

Пользовательская стратегия заполнения

Вы также можете предоставить пользовательскую стратегию заполнения в виде `lambda` выражения, которое принимает объект `pandas.Series` и значение гранулярности и возвращает обработанный объект `pandas.Series`.

```
for snapshot in signal_streaming.subscribe(
    consumer_group=f"e2e-ml-applications-{uuid.uuid4()}",
    history_size=timedelta(minutes=10),
    granularity=timedelta(minutes=1),
    signal_public_ids=[public_id],
    batch_interval=datetime.timedelta(seconds=1),
    columns_mapping_fill_strategy=defaultdict(lambda: lambda series, granularity: series.resample(granularity
):
```

DefaultFillStrategy

Класс `DefaultFillStrategy` определяет доступные стратегии заполнения сигналов по умолчанию. В число этих стратегий входят:

- `FILLNA`: заполняет пропущенные значения значением NaN.

```
series.resample(granularity).asfreq()
```

- `FFILL`: заполняет недостающие значения, используя предыдущее ненулевое значение в сигнале.

```
series.resample(granularity).ffill()
```

- `PREFILL_AVG_REINDEX`: ре-индексирует сигнал с фиксированным интервалом и заполняет недостающие значения средним значением точек данных в интервале.

```
series.resample(granularity).mean().ffill()
```

- `PREFILL_MIN_REINDEX`: ре-индексирует сигнал с фиксированным интервалом и заполняет пропущенные значения минимальным значением точек данных в интервале.

```
series.resample(granularity).min().ffill()
```

- `PREFILL_MAX_REINDEX`: ре-индексирует сигнал с фиксированным интервалом и заполняет пропущенные значения максимальным значением точек данных в интервале.

```
series.resample(granularity).max().ffill()
```

- `PREFILL_MEDIAN_REINDEX`: ре-индексирует сигнал с фиксированным интервалом и заполняет недостающие значения медианным значением точек данных в интервале.

```
series.resample(granularity).median().ffill()
```

- `PREFILL_LAST_REINDEX`: ре-индексирует сигнал с фиксированным интервалом и заполняет пропущенные значения последней точкой данных в интервале.

```
series.resample(granularity).last().ffill()
```

Экспорт сигналов

В этом руководстве вы познакомитесь с основами работы с экспортом через клиент экспорта SDK.

Предварительные условия

- Аккаунт на платформе Тайга.
- Локально установленная библиотека platform SDK.
- Минимальная настройка переменных окружения.

Обзор

В некоторых реальных сценариях необходимо получение данных с платформы для анализа, проверки гипотез и т.д. Экспорт сигналов - это метод получения данных из сигналов, работающий быстрее, чем через интерфейсы чтения и подписки.

В SDK есть клиент `SignalExporter` для выполнения экспорта данных.

Основные методы

В этом разделе рассматривается клиентский интерфейс `SignalExporter` и его основные методы.

Инициализация клиента

Прежде всего, необходимо инициализировать клиент `SignalExporter`:

```
from platform_sdk import SignalExporter

signal_exporter = SignalExporter()
```

Экспорт сигналов по списку публичных идентификаторов

Для `экспорта` данных по списку публичных идентификаторов необходимо указать его в параметре `public_ids`. Также необходимо указать диапазон дат `min_dt` - `max_dt`, и ряд опций:

- Выполнение `pandas`-подобного метода pivot контролируется флагом `to_pivot`
- Удаление артефакта после загрузки контролируется флагом `to_delete_artifact`
- Публичный идентификатор пространства (ABAC) можно указать в параметре `space_public_id`

```
from platform_sdk import SignalExporter

signal_exporter = SignalExporter()
signal_exporter.export_signals(
        public_ids=["public_id_1", "public_id_2"]
        min_dt=datetime(2024, 1, 1, 12, 10),
        max_dt=datetime(2024, 1, 1, 12, 12),
    )
```

Выполнение приведенного выше кода дает следующий результат:

```
timestamp                      signal        value
2023-01-01 12:10:00+00:00     signal_1        21.0
2023-01-01 12:11:00+00:00     signal_1        22.0
2023-01-01 12:11:00+00:00     signal_2        23.0
2023-01-01 12:12:00+00:00     signal_2        24.0
```

С флагом `to_pivot=True`:

```
                        my_signal_1  my_signal_2
2023-01-01 12:10:00+00:00        21.0          NaN
2023-01-01 12:11:00+00:00        22.0         23.0
2023-01-01 12:12:00+00:00         NaN         24.0
```

Экспорт сигналов по списку имен групп

Для экспорта данных на основе списка имен групп необходимо указать его в параметре `group_names`. Также необходимо указать диапазон дат `min_dt` - `max_dt`, и ряд опций:

- Выполнение `pandas`-подобного метода pivot контролируется флагом `to_pivot`
- Удаление артефакта после загрузки контролируется флагом `to_delete_artifact`
- Публичный идентификатор пространства (ABAC) можно указать в параметре `space_public_id`

```
from platform_sdk import SignalExporter

signal_exporter = SignalExporter()
signal_exporter.export_groups(
        group_names=["group_1", "group_2"]
        min_dt=datetime(2023, 1, 1, 12, 10),
        max_dt=datetime(2023, 1, 1, 12, 12),
    )
```

Предположим, что у нас есть группа `group_1` с `signal_1` и группа `group_2` с `signal_2`. Выполнение приведенного выше кода дает следующий результат:

```
timestamp                      signal        value
2023-01-01 12:10:00+00:00     signal_1        21.0
2023-01-01 12:11:00+00:00     signal_1        22.0
2023-01-01 12:11:00+00:00     signal_2        23.0
2023-01-01 12:12:00+00:00     signal_2        24.0
```

С флагом `to_pivot=True`:

```
                        my_signal_1  my_signal_2
2023-01-01 12:10:00+00:00        21.0          NaN
2023-01-01 12:11:00+00:00        22.0         23.0
2023-01-01 12:12:00+00:00         NaN         24.0
```

Приложение с валидацией параметров

В этом руководстве вы узнаете, как автоматически валидировать параметры приложения с помощью `ParameterizedPipelineStep` и pydantic.

Предварительные условия

- Данное руководство является продолжением руководства Создайте свое первое приложение. Если вы еще не прочитали его, пожалуйста, сделайте это, прежде чем продолжить.
- Желательно хотя бы поверхностное знакомство с понятиями датаклассов или моделями Pydantic.

Краткий обзор предыдущих руководств

Вот краткий обзор предыдущего руководства:

- В руководстве Создайте свое первое приложение рассматривается создание самого базового приложения, которое можно реализовать с помощью SDK. Его рекомендуется использовать как основу для ваших проектов. Единственное, что меняется между вашими проектами - это сам код приложения. Все ваши приложения должны быть подклассами класса `PipelineStep` (или его подклассов), предоставляющего интерфейс, с которым может взаимодействовать платформа.

Обзор

При запуске приложения на платформе необходимо указать схему входных параметров приложения. Данная схема позволяет заполнять входные параметры по отдельности через удобный пользовательский интерфейс при последующем запуске приложения, однако при этом все еще остается возможность совершения ряда ошибок:

- Пользователь может забыть о некоторых параметрах.
- Пользователь может опечататься в имени параметра.
- Пользователь может передать неверные значения параметров, например, строку вместо числа.

Это может привести к ошибкам во время выполнения программы, которые могут быть изначально неочевидны.

Таким образом, добавление валидации входных параметров перед запуском приложения может существенно сэкономить время на разработку и отладку. Именно это и делает шаблон приложения `ParameterizedPipelineStep`.

Пример приложения

В этом разделе вы получите практический опыт работы с базовым классом `ParameterizedPipelineStep` через пошаговую реализацию простого приложения с валидацией параметров. Приложение в примере ниже принимает один целочисленный, один строковый и один необязательный JSON-параметр и выводит их значения в логах.

Определите свои параметры как модель

Базовый класс `ParameterizedPipelineStep` делегирует всю валидацию библиотеке `pydantic`, созданной специально для этих целей.

Поэтому перед написанием приложения необходимо задать схему входных параметров в виде `pydantic`-модели. Рассмотрим простое приложение, которое принимает один целочисленный, один строковый и один необязательный JSON-параметр. В этом случае схема входных параметров должна выглядеть следующим образом:

```
import json
from typing import Optional
from pydantic import BaseModel, validator

class InputParamsSchema(BaseModel):
    """Input Params Schema"""

    int_parameter: int
    str_parameter: str
    json_parameter: Optional[dict]

    @validator("json_parameter", pre=True)
    def load_json(cls, value: Optional[str]) -> Optional[dict]:
        if value is None:
            return value
        else:
            return json.loads(value)
```

Также необходимо указать схему параметров процесса выполнения. Если же доступ к параметрам процесса выполнения не нужен, то можно просто создать пустую модель:

```
class JobParamsSchema(BaseModel):
    """Job Params Schema"""
```

Создайте класс приложения при помощи наследования

В руководстве Создайте свое первое приложение вы создали класс приложения, наследуя от базового класса `PipelineStep`, который является наиболее общим родительским классом для всех приложений. Здесь мы создаем класс приложения, наследуя от класса `ParameterizedPipelineStep`, который также поддерживает работу со схемами параметров.

Ваше приложение на основе класса `ParameterizedPipelineStep` должно выглядеть так же, как приложение из руководства Создайте свое первое приложение, но для его работы необходимо указать два дополнительных атрибута класса: `_input_params_schema` и `_job_params_schema`. Они должны возвращать соответствующие классы параметров, которые вы указали выше:

```
from typing import Type

from platform_sdk import ParameterizedPipelineStep

class CustomApplication(ParameterizedPipelineStep):
    def __init__(self) -> None:
        super().__init__()
        self._signal_ops = signal_ops

    @property
    def _input_params_schema(self) -> Type[InputParamsSchema]:
        return InputParamsSchema

    @property
    def _job_params_schema(self) -> Type[JobParamsSchema]:
        return JobParamsSchema
```

Используйте значения параметров

Если вы указали модели параметров, то можете обращаться к параметрам из тела приложения непосредственно по имени параметра следующим образом:

```
class CustomApplication(ParameterizedPipelineStep):
    ...

    def _start(self, task_id: int, job_params: JobParamsSchema, input_params: InputParamsSchema) -> None:
        print("This is an int parameter", input_params.int_parameter)
        print("This is a str parameter", input_params.str_parameter)
        print("And this is a dict parameter", input_params.json_parameter)
```

> ⓘ Примечание
>
> Убедитесь, что вы определяете логику приложения в методе `_start` (обратите внимание на нижнее подчеркивание `_`, с которого начинается имя метода), а не в методе `start`, как это было сделано в руководстве Создайте свое первое приложение.

Перед выполнением кода внутри метода `_start` приложение `ParameterizedPipelineStep` валидирует все параметры в соответствии с заданными моделями и вызывает ошибку `ValidationError`, если что-то пошло не так.

Полный пример

Ниже приведен полный пример кода приложения. Это приложение принимает один целочисленный, один строковый и один необязательный JSON-параметр и выводит их в логе приложения:

```
"""Paste this to entrypoint.py"""
import json
from typing import Optional, Type

from pydantic import BaseModel, validator

from platform_sdk import ParameterizedPipelineStep


class JobParamsSchema(BaseModel):
    """Job Params Schema"""


class InputParamsSchema(BaseModel):
    """Input Params Schema"""

    int_parameter: int
    str_parameter: str
    json_parameter: Optional[dict]

    @validator("json_parameter", pre=True)
    def load_json(cls, value: Optional[str]) -> Optional[dict]:
        if value is None:
            return value
        else:
            return json.loads(value)


class CustomApplication(ParameterizedPipelineStep):
    def __init__(self) -> None:
        super().__init__()

    @property
    def _input_params_schema(self) -> Type[InputParamsSchema]:
        return InputParamsSchema
```

```
    @property
    def _job_params_schema(self) -> Type[JobParamsSchema]:
        return JobParamsSchema

    def _start(self, task_id: int, job_params: JobParamsSchema, input_params: InputParamsSchema):
        print("This is an int parameter", input_params.int_parameter)
        print("This is a str parameter", input_params.str_parameter)
        print("And this is a dict parameter", input_params.json_parameter)


if __name__ == "__main__":
    # get input params schema for platform UI
    params_schema = CustomApplication().get_input_params_schema()
    print(json.dumps(params_schema, indent=4))
```

Вы также можете получить схему входных параметров, чтобы вставить ее в пользовательский интерфейс платформы, выполнив файл напрямую:

```
$ python3 entrypoint.py
```

Чтобы запустить это приложение локально или на платформе, вставьте этот код в файл `entrypoint.py` и выполните те же действия, что и в руководстве Создайте свое первое приложение.

Смотрите также

Более показательные примеры можно найти во встроенных приложениях SDK, так же реализованных на основе `ParameterizedPipelineStep`:

- Data player
- Metrics
- PandasEvaluator

Приложение с уведомлениями

В этом руководстве вы узнаете, как встроить уведомления в ваше приложение с помощью `AlertPipelineStep`.

Предварительные условия

- Данное руководство является продолжением руководств Создайте свое первое приложение и Приложение с валидацией параметров. Если вы еще не прочитали их, пожалуйста, сделайте это, прежде чем продолжить.
- Ознакомьтесь с концепцией уведомлений. Прочитайте раздел «Уведомления» в документации платформы. Чтобы получить доступ к документации платформы, войдите в систему, нажмите на значок пользователя в левом нижнем углу страницы и выберите «Документация по платформе».

Краткий обзор предыдущих руководств

Вот краткий обзор предыдущих руководств:

- В руководстве Создайте свое первое приложение рассматривается создание самого базового приложения, которое можно реализовать с помощью SDK. Его рекомендуется использовать как основу для ваших проектов. Единственное, что меняется между вашими проектами - это сам код приложения. Все ваши приложения должны быть подклассами класса `PipelineStep` (или его подклассов), предоставляющего интерфейс, с которым может взаимодействовать платформа.
- Однако, поскольку реальные приложения обычно имеют очень сложные интерфейсы с большим количеством параметров, они редко реализуются как подкласс `PipelineStep`. Наилучший выбор в этом случае - использование класса `ParameterizedPipelineStep`, предусматривающего встроенную валидацию входных параметров. Валидация параметров подробно рассматривается в руководстве Приложение с валидацией параметров.

Обзор

Хотя надежные схемы валидации параметров могут предотвратить запуск приложения с недопустимыми входными параметрами, ошибки во время выполнения все-равно случаются, даже если ваш код имеет хорошее тестовое покрытие. Вы можете не иметь возможности круглосуточного мониторинга приложения на платформе и, следовательно, замечать перезапуски или сбои в работе приложения спустя длительное время после их возникновения.

В этом случае может помочь класс `AlertPipelineStep`. Это подкласс `ParameterizedPipelineStep`, который имеет с ним общий интерфейс, но также имеет встроенный Клиент уведомлений. Его основными возможностями являются:

- Простая работа с уведомлениями: вам необходимо только создать группу уведомлений, а приложение позаботится обо всем остальном - вы можете опубликовать уведомления в любом месте тела вашего приложения, указав только сообщение уведомления.
- Приложение отслеживает ранее опубликованное сообщение и блокирует отправку двух или более одинаковых сообщений подряд, пока не истечет заданный таймаут. Это позволяет избежать случайной рассылки спама получателям (или самому себе). Более подробную информацию см. в разделе Избегание спама.
- При определенной настройке приложения можно отправить уведомление всем пользователям в случае непредвиденной ошибки в рантайме перед выходом из приложения: пользователи получат текст ошибки и сообщение трассировки (`Traceback`).

Функциональность

В этом разделе мы более подробно рассмотрим основные возможности `AlertPipelineStep`.

Публикация уведомлений

Класс `AlertPipelineStep` добавляет к базовому интерфейсу `ParameterizedPipelineStep` новый метод `publish_notifications` для публикации уведомлений. Этот метод может быть использован внутри вашего приложения следующим образом:

```
class CustomApplication(AlertPipelineStep):
    ...
```

```
    def custom_method(self) -> None:
        self.publish_notifications("Hello, world!")
```

Нетрудно видеть, что данный метод не требует ссылок на саму группу уведомлений. Это связано с тем, что `AlertPipelineStep` принимает один дополнительный входной параметр `notification_group_name`, который задает имя групп уведомлений (одной или нескольких). Этот параметр будет рассмотрен в следующем разделе).

Параметр группы уведомлений

`AlertPipelineStep` принимает один дополнительный строковый входной параметр `notification_group_name` для указания группы уведомлений для вашего приложения. Если вы запускаете свое приложение на платформе, просто добавьте его в схему параметров следующим образом:

```
{
    "fields": [
        {
            "fieldType": "TEXT",
            "name": "notification_group_name",
            "label": "notification_group_name",
            "description": "Name of the notification groups separated by comma",
            "isRequired": true
        }
        // the rest of your schema
    ]
}
```

Затем все отправленные сообщения будут подставлены в переменную `${message}` настроенной группы уведомлений. Например:

```
You got new important notification: ${message}
```

Если у вас уже есть заранее настроенные группы уведомлений, вы можете передать их имена в приложение. Если нет, обратитесь к документации по уведомлениям на платформе.

Уведомление о сбое приложения

Приложение также может отправлять уведомление при возникновении непредвиденной ошибки в рантайме. Для этого необходимо добавить два новых входных параметра:

```
{
    "fields": [
        {
            "fieldType": "TEXT",
            "name": "notify_if_failed",
            "label": "notify_if_failed",
            "description": "If True, send notification in case the application fails",
            "isRequired": true,
            "defaultValue": "True"
        },
        {
            "fieldType": "TEXT",
            "name": "notify_if_failed_groups",
            "label": "notify_if_failed_groups",
            "description": "Notification groups for alerting in case app is failed, separated by comma",
            "isRequired": true,
            "defaultValue": "True"
        }
        // the rest of your schema
```

```
    ]
}
```

затем установить параметр `notify_if_failed` в значение `True` и указать значение `notify_if_failed_groups`.

В этом случае в группы уведомлений `notify_if_failed_groups` будет отправлено уведомление с ошибкой и , который будет доступен, как и все остальные уведомления, по ключу *message*. Например:

```
{"message": "ValueError ..."}
```

Если не указать этот параметр, то приложение не будет отправлять никаких уведомлений в случае ошибок.

Приостановка уведомлений по значению в сигнале

Если вам необходимо, чтобы приложение перестало отправлять уведомления в зависимости от значения в определенном сигнале, вы можете добавить следующий входной параметр:

```
{
    "fields": [
        {
            "fieldType": "LONG_TEXT",
            "name": "stop_notifications_indicator",
            "label": "stop_notifications_indicator",
            "description": "Configuration for stop notifications indicator signal",
            "isRequired": true,
            "defaultValue": "{\"signal\": \"indicator\", \"notification_groups\": \"my_groups\", \"signal_type
        }
        // the rest of your schema
    ]
}
```

В зависимости от значений в приведенном сигнале приложение приостановит отправку уведомлений. Сигнал, используемый в качестве индикатора, может быть одного двух видов:

1. Статусный сигнал: в этом случае сигнал должен содержать только нули и единицы. Если значение сигнала равно 0, приложение приостанавливает отправку уведомлений. В этом случае конфигурация в формате JSON принимает следующий вид:

   ```
   {
       "signal": "my_signal",
       "notification_groups": "group1,group2",
       "signal_type": "status"
   }
   ```

   Если в сигнал приходит значение 0, приложение отправит уведомление в группы уведомлений, перечисленные во входном параметре `notification_groups` и приостановит дальнейшие уведомления.

2. Хартбит сигнал. В этом случае, сигнал должен содержать чередующиеся значения 0 и 1, поступающие с определенной частотой. Если значения перестают приходить в сигнал, он считается неактивным, и приложение прекращает отправку уведомлений. Вы можете задать данную конфигурацию в формате JSON следующим образом:

```
{
    "signal": "my_signal",
    "notification_groups": "group1,group2",
    "signal_type": "heartbeat",
    "periodicity": "1s"
}
```

Если в хартбит сигнал не будет поступать значение с заданной периодичностью, приложение приостановит отправку уведомлений.

> ⓘ Примечание
>
> Параметр `notification_groups` опциональный. Если вы не зададите этот параметр, то приложение не оповестит вас о дальнейшем прекращении уведомлений. Вместо этого приложение лишь выведет эту информацию в лог.

Отсылать одинаковые сообщения после истечения таймаута

По умолчанию приложение отправляет только уникальные уведомления. Это означает, что если статус был отправлен однажды, он не будет отправлен снова, если не произошли изменения. Однако вы можете использовать параметры `duplicate_message_timeouts` или `duplicate_message_timeout`, чтобы указать таймаут для идентичных сообщений. Если вы устанавливаете такой таймаут, приложение сможет отправить одно и то же уведомление вторично после истечения таймаута. Разница между `duplicate_message_timeouts` и `duplicate_message_timeout` заключается в том, что в параметре `duplicate_message_timeout` необходимо установить единый таймаут для всех сигналов, в то время как в `duplicate_message_timeouts` вы можете указать индивидуальные таймауты для каждого сигнала.

Чтобы использовать `duplicate_message_timeout` параметр вам нужно добавить следующие строки в `input_params`:

```
{
    "fields": [
        {
            "fieldType": "TEXT",
            "name": "duplicate_message_timeout",
            "label": "duplicate_message_timeout",
            "description": "Timeout for sending duplicated messages in Pandas Timedelta format",
            "isRequired": true,
            "defaultValue": "10min"
        }
        // the rest of your schema
    ]
}
```

Чтобы использовать `duplicate_message_timeouts` параметр вам нужно добавить следующие строки в `input_params`:

```
{
    "fields": [
        {
            "fieldType": "TEXT",
            "name": "duplicate_message_timeouts",
            "label": "duplicate_message_timeouts",
            "description": "List of individual timeouts for sending duplicated messages in Pandas Timedelta f
            "isRequired": true,
```

```
            "defaultValue": "10min,15min,5min"
        }
        // the rest of your schema
    ]
}
```

> ⓘ Примечание
>
> Вместо того, чтобы использовать `duplicate_message_timeouts` параметр для задания индивидуальных таймаутов для Приложения Ожиданий, задавайте их внутри `meta`.

Избегание спама

Другая особенность метода `publish_notifications` заключается в том, что он отслеживает ранее опубликованные сообщения и не публикует их при попытке отправки двух или более одинаковых сообщений подряд. В большинстве случаев такое поведение нежелательно, поскольку это может привести к сильному спаму подписчиков. Рассмотрим следующий фрагмент:

```
class CustomApplication(AlertPipelineStep):
    ...

    def spam(self) -> None:
        denominator = 0

        for i in range(100):
            try:
                result = i / denominator
            except:
                self.publish_notifications("An error occurred!")
```

Вместо того чтобы отправлять одно за другим 100 одинаковых сообщений, приложение отправит только первое и проигнорирует все остальные.

> ⓘ Примечание
>
> Несмотря на то, что мы настоятельно рекомендуем использовать метод `publish_notifications` для уведомления пользователя только об *обновлениях* приложения (например, приложение поймало ошибку №1, приложение вернулось в нормальное состояние, приложение поймало ошибку №2), если вам необходимо получать каждое сообщение, вы можете обойти эту проблему. Для этого просто добавьте в сообщения уникальные значения, например, дату и время или текст исключения.

Если все же необходимо, чтобы приложение отправляло идентичные сообщений подряд, можно добавить в схему входных параметров опциональный параметр `duplicate_message_timeout`:

```
{
    "fields": [
        {
            "fieldType": "TEXT",
            "name": "duplicate_message_timeout",
            "label": "duplicate_message_timeout",
            "description": "If set, allow sending duplicate messages after specified timeout. The value must
```

```
            "isRequired": false,
            "defaultValue": "100 min"
        }
        // the rest of your schema
    ]
}
```

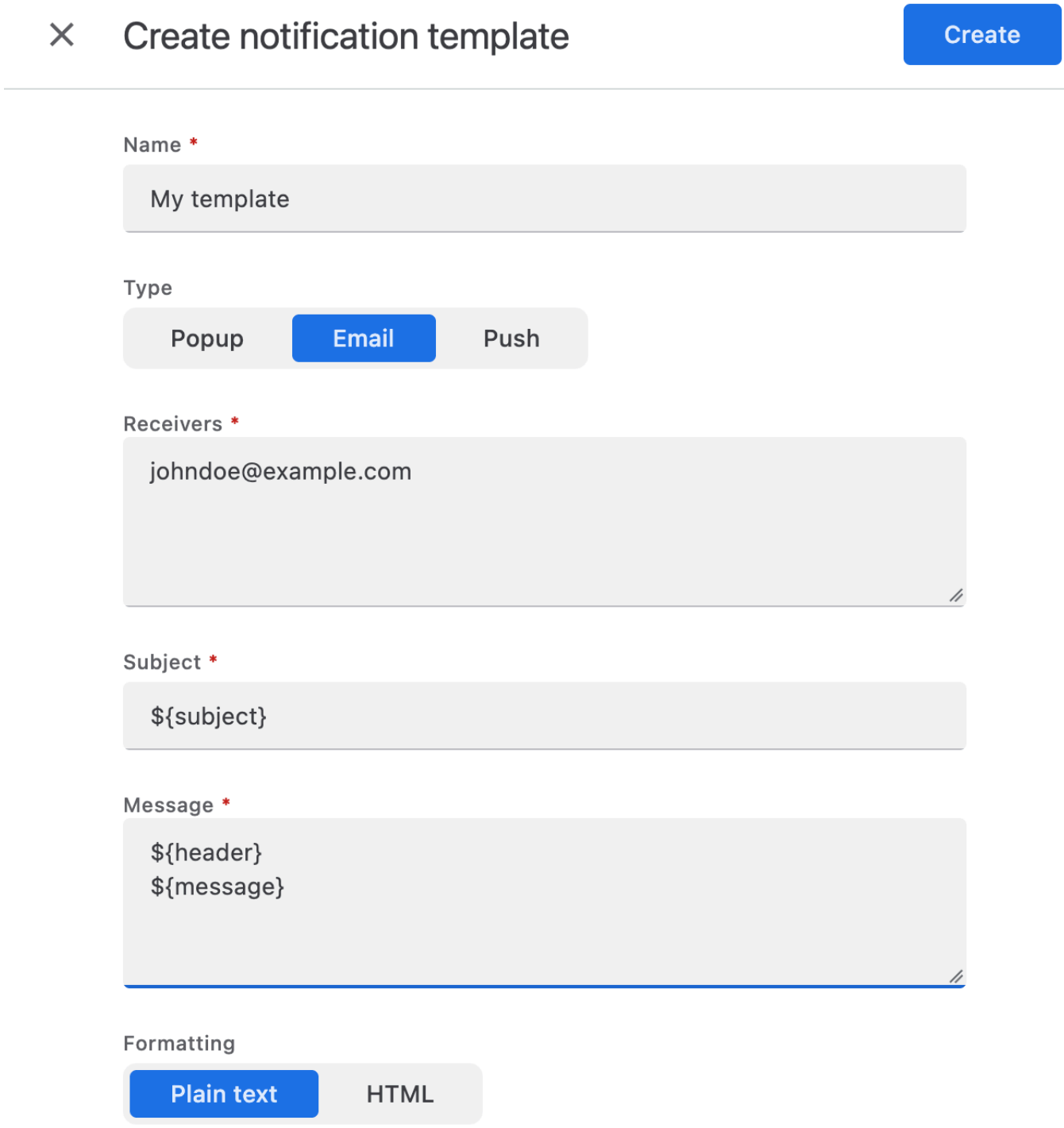
Если этот параметр указан, то идентичные сообщения могут быть отправлены по истечении указанного таймаута.

Отправка дополнительных пар ключ-значение

Приложение также может отправлять дополнительные пары ключ-значение, используемые в шаблоне уведомления. Для этого добавьте дополнительный параметр в схему входных параметров:

```
{
    "fields": [
        {
            "fieldType": "TEXT",
            "name": "additional_keys",
            "label": "additional_keys",
            "description": "Dictionary with key value pairs.",
            "isRequired": true,
            "defaultValue": "{\"subject\": \"super-important\", \"header\": \"ALARM\"}"
        },
        // the rest of your schema
    ]
}
```

При добавлении этого параметра вы можете настроить шаблон таким образом, чтобы заданные пары ключ-значение были подставлены в сообщение. Например:



Отправка дополнительных пар ключ-значение

В некоторых случаях требуется задать различные дополнительные ключи для сигналов, которые являются причиной нотификации. Это может быть сделано с помощью следующей концигурации параметра `additional_keys`

```
{
    "per_signal":
        {
            "pid1": {"key1: "val1"},
            "pid2": {"key2: "val2"}
        }
}
```

Для примера выше - дополнительное значение `{"key1: "val1"}` будет передано, если алерт возник с этим сигналом.

> ⓘ Примечание
>
> Пожалуйста обратите внимание, что определение исходного сигнала для нотификации возможно только для приложений `Data Periodicity` и `Watchdog`.

Код приложения

Вот простое приложение, которое можно запустить на платформе, чтобы получить некоторый практический опыт работы с `AlertPipelineStep`:

```
"""entrypoint.py"""
import time
from typing import Type

from pydantic import BaseModel
from platform_sdk import (
    AlertPipelineStep,
    Notifications,
    SignalsOps,
    SignalWriterClient,
    SignalsGroupOps,
    SignalReaderClient,
    JobParamsSchema,
)


class InputParamsSchema(BaseModel):
    """Input params schema"""

    my_parameter: str


class CustomApplication(AlertPipelineStep):

    def __init__(
        self,
        reader: SignalReaderClient,
        writer: SignalWriterClient,
        signals_ops: SignalsOps,
        signals_group_ops: SignalsGroupOps,
        notifications: Notifications,
    ) -> None:
        super().__init__(
            writer=writer,
            signals_ops=signals_ops,
            signals_group_ops=signals_group_ops,
            notifications=notifications,
        )  # AlertPipelineStep constructor requires these 4 clients
        # Add your custom steps to constructor
        self._reader = reader

    @property
    def _input_params_schema(self) -> Type[InputParamsSchema]:
        return InputParamsSchema

    @property
    def _job_params_schema(self) -> Type[JobParamsSchema]:
        return JobParamsSchema

    def _start(self, task_id: int, input_params: InputParamsSchema, job_params: JobParamsSchema) -> None:
        self.publish_notifications("Application has been launched!")

        for i in range(1, 101):
            if i % 25 == 0:  # publish notifications every 25 steps
```

```
                self.publish_notifications(f"Running step {i}/100...")
            time.sleep(5)

        self.publish_notifications("Duplicated message. You will get only the first one")
        self.publish_notifications("Duplicated message. You will get only the first one")

        raise ValueError("You will get notification about this despite application has failed!")
```

Это приложение отправляет уведомление при запуске, затем считает до 100, отправляя уведомление на 25-м, 50-м и 75-м шаге. Затем оно пытается отправить два одинаковых сообщения, из которых вы получите только одно из-за функциональности запоминания последнего сообщения. Наконец, приложение завершится с ошибкой `ValueError`, симулирующей непредвиденное исключение, но, тем не менее, отправит вам уведомление об ошибке.

Вы также можете получить схему входных параметров, чтобы вставить ее в пользовательский интерфейс платформы, выполнив файл напрямую:

```
$ python3 entrypoint.py
```

Чтобы запустить это приложение локально или на платформе, вставьте этот код в файл `entrypoint.py` и выполните те же действия, что и в руководстве Создайте свое первое приложение.

Смотрите также

Более показательные примеры приведены в приложениях на основе `AlertPipelineStep`:

- Периодичность данных
- Expectations

Клиент артефактов

В этом руководстве вы познакомитесь с основами и получите практический опыт работы с артефактами через клиент артефактов SDK.

Предварительные условия

- Аккаунт на платформе Тайга.
- Локально установленная библиотека platform SDK.
- Минимальная настройка переменных окружения.

Обзор

Артефакты - это большие файлы, хранящиеся в бинарном формате в хранилище артефактов платформы. Артефакты обычно используются для хранения весов моделей или больших конфигурационных файлов.

> ⓘ Примечание
>
> Для получения более подробной информации об артефактах обратитесь к документации платформы. Чтобы открыть документацию платформы, войдите в систему, нажмите на значок пользователя в левом нижнем углу страницы и выберите **Документация по платформе**.

Для работы с артефактами в SDK платформы служит клиент `Artifacts`.

Основные методы

В данном разделе рассмотрены основные методы клиента артефактов SDK.

Инициализация клиента

Прежде всего, необходимо инициализировать клиент артефактов.

```
from platform_sdk import Artifacts

artifacts = Artifacts()
```

Загрузка артефакта на платформу

Артефактом может быть любой объект, представимый в бинарном формате. В качестве примера загрузим закодированную строку `Hello, world!`:

```
model_bytes = "Hello, world!".encode()
uploaded_artifact = artifacts.upload_no_metadata(label="my_artifact", model_bytes=model_bytes, ttl_hours=1)

print(uploaded_artifact)
```

Здесь `label` - название артефакта, `model_bytes` - содержимое бинарного файла, `ttl_hours` - настройка TTL для данного артефакта (подробнее об этом параметре можно узнать в разделе TTL параметр)

Выполнение приведенного выше кода дает следующий результат:

```
{
    'id': 72074,
    'sha256': '330c723f25267587db0b9f493463e017011239169cb57a6db216c63774367115',
    'label': 'my_artifact',
    'createdAt': '2022-09-09T16:12:25.598Z'
}
```

> ⓘ Примечание
>
> Обратите внимание, что идентификатор артефакта уникален, а название - нет.

Поскольку название артефакта не является уникальным, загрузка другого артефакта с тем же названием не вызовет ошибок:

```
model_bytes = "It's understood that Hollywood sells Californication".encode()
uploaded_artifact = artifacts.upload_no_metadata(label="my_artifact", model_bytes=model_bytes, ttl_hours=1)
```

Если необходимо указать публичный идентификатор артефакта, используйте параметр `space_public_id`.

```
model_bytes = "Hello, world!".encode()
uploaded_artifact = artifacts.upload_no_metadata(
    label="my_artifact",
    model_bytes=model_bytes,
    space_public_id="my-space",
    ttl_hours=1,
)

print(uploaded_artifact)
```

Параметр TTL

Параметр TTL (Time to Live) позволяет контролировать время жизни артефактов. Аргумент `ttl_hours` определяет количество времени в часах, после которого артефакт автоматически удаляется. Если параметр ttl не указан, артефакт будет храниться, пока его не удалят вручную. Установка TTL помогает поддерживать чистоту платформы, экономить место для хранения и вычислительные ресурсы.

> ⓘ Примечание
>
> Настоятельно рекомендуется задавать TTL для всех артефактов. В противном случае платформа может быть переполнена устаревшими артефактами, занимающими место в памяти.

Установка TTL может быть крайне важна в различных сценариях. В общем случае установка TTL рекомендуется для быстро устаревающих артефактов и артефактов, генерируемых приложениями в большом количестве.

Вот несколько конкретных случаев, когда рекомендуется использовать TTL:

1. Хранение моделей в виде артефактов: при непрерывном обучении на и хранении весов моделей в виде артефактов важно установить TTL. Постоянное добавление артефактов может в конечном итоге замедлить поиск и загрузку по названиям.
2. Динамический импорт из артефактов: хорошей практикой является хранение только последних версий кода и удаление устаревших версий по дате их создания. Установка TTL при загрузке артефактов облегчает этот процесс.
3. Хранение состояния модели в виде артефактов: аналогично хранению моделей в виде артефактов, установка TTL позволяет автоматически удалять устаревшие наборы данных, тем самым ускоряя поиск и получение артефактов по названиям.
4. Хранение часто меняющихся конфигураций: если конфигурационные файлы быстро устаревают, целесообразно установить TTL, что позволяет экономить ресурсы хранения.

> ⓘ Примечание
>
> Если вы используете артефакты для хранения данных или конфигураций, которые могут понадобиться в течение нескольких месяцев, безопаснее оставить значение ttl_hours равным None и вручную выполнять периодическую очистку, удаляя устаревшие артефакты каждые полгода.

Скачивание артефактов по числовому ID

Теперь давайте скачаем артефакт, загруженный на предыдущем шаге:

```
model_bytes = artifacts.download(uploaded_artifact["id"])

print(model_bytes.decode())
```

Выполнение приведенного выше кода дает следующий результат:

```
"Hello, world!"
```

Скачивание последнего артефакта по названию

Вы также можете скачать артефакт по его названию. В этом случае вы получите последний артефакт с этим названием. Однако такой способ скачивания не рекомендуется, так как может привести к неожиданным результатам, если другой пользователь загрузит новый артефакт с точно таким же названием:

```
latest_artifact = artifacts.get_latest("my_artifact")
model_bytes = artifacts.download(latest_artifact["id"])

print(model_bytes.decode())
```

Выполнение приведенного выше кода дает следующий результат:

```
"It's understood that Hollywood sells Californication"
```

Клиент уведомлений

В этом руководстве вы познакомитесь с основами работы с уведомлениями через клиент уведомлений SDK.

Предварительные условия

- Аккаунт на платформе Тайга.
- Локально установленная библиотека platform SDK.
- Минимальная настройка переменных окружения.

Обзор

Уведомления - это функциональность платформы, позволяющая оповещать пользователей вне платформы при помощи сообщений на электронную почту, всплывающие окна и баннеры на пользовательском интерфейсе платформы. Полный список поддерживаемых Протоколов доставки приведен ниже. Механизм уведомлений аналогичен механизму AWS Simple Notification Service.

> ⓘ Примечание
>
> Для получения более подробной информации об уведомлениях обратитесь к документации платформы. Чтобы открыть документацию платформы, войдите в платформу, нажмите на значок пользователя в левом нижнем углу страницы и выберите **Документация по платформе**.

В SDK есть специальный клиент для работы с уведомлениями.

Процесс

Процесс отправки уведомлений состоит из 3 этапов:

1. Создание новой группы уведомлений.
2. Добавление шаблонов уведомлений в группу уведомлений из доступных протоколов доставки. Обратите внимание, что в одной группе уведомлений может быть несколько шаблонов уведомлений, в том числе с разными протоколами доставки.
3. Публикация сообщения в группе уведомлений.

В разделе Основные методы ниже вы сможете попрактиковаться в работе с уведомлениями, пройдя все эти шаги.

Основные методы

В данном разделе рассмотрены основные методы клиента уведомлений SDK.

Инициализация клиента

Прежде всего, необходимо инициализировать клиент уведомлений

```
from platform_sdk import Notifications

notifications = Notifications()
```

Создание группы уведомлений

Группа уведомлений является главной сущностью, поскольку она представляет собой полную конфигурацию о том, какие пользователи должны быть уведомлены и какие типы уведомлений должны использоваться для каждого пользователя при новом оповещении в группе уведомлений. Группы уведомлений обычно создаются и настраиваются для одной цели, например, группа уведомлений для уведомления команды DS о некорректной работе модели ML или группа уведомлений для уведомления команды поддержки в случае возникновения аварийной ситуации.

Теперь создадим новую группу уведомлений:

```
from platform_sdk import Notifications

notifications = Notifications()
my_notification_group = notifications.create_notification_group(name="my_group")
print(my_notification_group)
print(my_notification_group.name)
```

Выполнение приведенного выше кода дает следующий результат:

```
from platform_sdk import NotificationGroup

NotificationGroup(name='my_group', notification_group_id=101)
'my_group'
```

Добавление шаблона уведомления в группу уведомлений

После создания новой группы уведомлений можно добавить шаблон уведомления для любого доступного протокола и указанных пользователей. Например, так можно добавить шаблон уведомления по рабочей почте

```
from platform_sdk import Notifications

notifications = Notifications()
notifications.add_notification_template_email(
    name="my_receiver1",
    notification_group_id=my_group.notification_group_id,
    to=["<your_email>"],
    subject_template="Hello from ${friend}",
    message_template="${friend} sends you a message: ${message}",
)
```

В параметре `to` можно указать до 100 адресов. Обратите внимание, что если указать список адресов, то все получатели получат одно общее сообщение. Параметры `subject_template` и `message_template` поддерживают подстановку значений по имени переменной (по аналогии с Bash или f-строками в Python). Соответствующие имена должны быть указаны в фигурных скобках со знаком доллара: `${}`. В приведенных примерах мы определяем 2 переменные: `friend` и `message`.

Если подстановки не нужны, можно передать обычные строки, например `subject_template="Alert!"`.

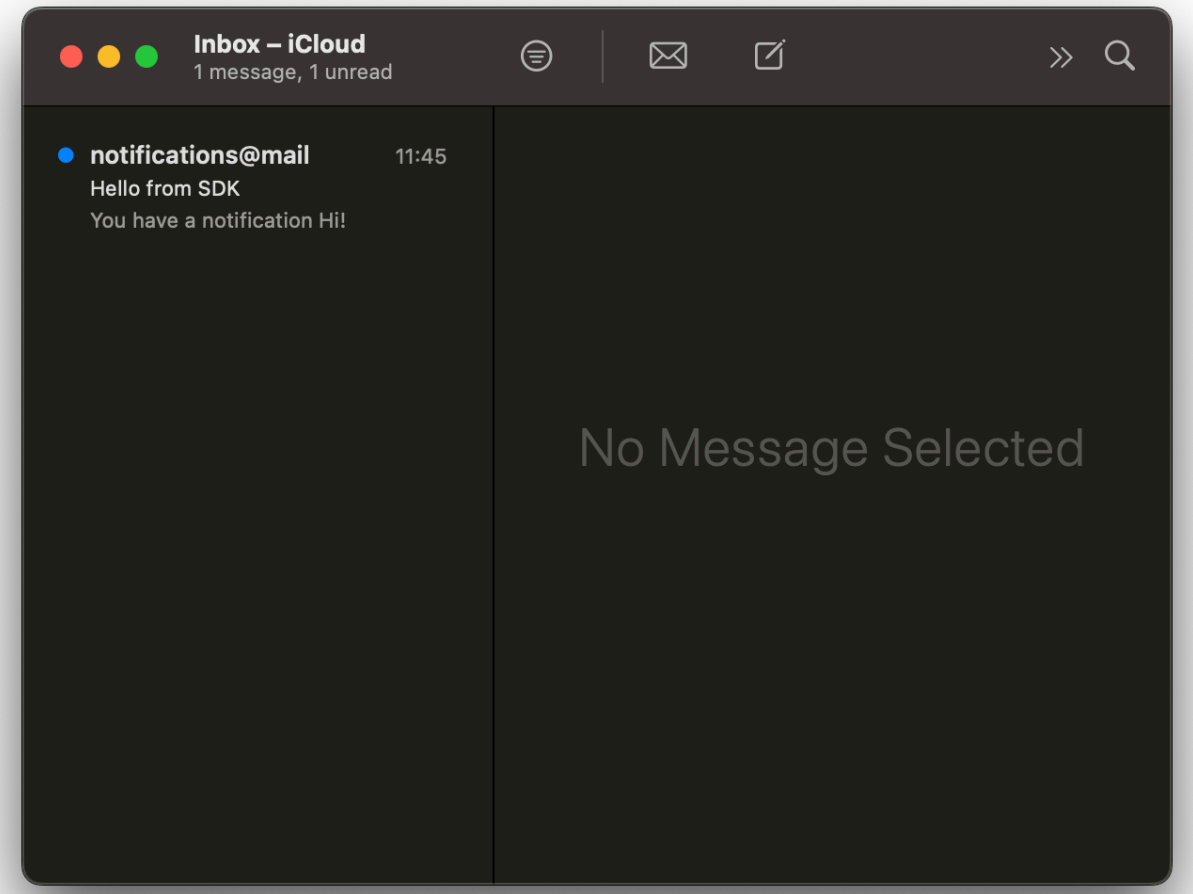
Публикация сообщения в группу уведомлений

Теперь, когда группа уведомлений создана и в ней настроен шаблон уведомления, мы готовы к отправке сообщения:

```
notifications.publish(
    notification_group_name=my_group.name,
    payload={"friend": "SDK", "message": "Hello!"},
)
```

> ⓘ Примечание
>
> Убедитесь, что в параметр `payload` переданы значения для всех переменных, определенных выше, иначе сообщение не будет получено. Если вы не указали ни одной переменной, вы должны передать пустой словарь (`payload={}`).

Откройте приложение электронной почты и убедитесь, что уведомление получено:



Деактивация и активация группы и шаблона уведомлений

Если необходимо остановить все уведомления для группы уведомлений или, другими словами, отключить уведомления для группы, можно воспользоваться следующим способом:

```
notifications.deactivate_notification_group(notification_group_id=my_group.notification_group_id)
```

Начиная с этого момента, уведомления не будут отправляться получателям, настроенным во всех шаблонах.

Чтобы снова включить уведомления, просто выполните следующие действия:

```
notifications.activate_notification_group(notification_group_id=my_group.notification_group_id)
```

Начиная с этого момента все уведомления будут работать как и раньше. Обратите внимание, что при создании все группы уведомлений считаются активными.

Кроме того, можно деактивировать и активировать только один из конкретных шаблонов уведомлений

```
template = notifications.list_group_templates(notification_group_id=my_group.notification_group_id)[0]
notifications.deactivate_template(my_group.notification_group_id, template["id"])
notifications.activate_template(my_group.notification_group_id, template["id"])
```

> ⓘ Примечание
>
> Отключение уведомлений для всей группы, по сути, равносильно отключению уведомлений для каждого отдельного шаблона в этой группе. Другими словами, можно отключить всю группу, а затем выборочно включить уведомления для определенных шаблонов при необходимости.

Удаление группы уведомлений

Если группа уведомлений больше не нужна, ее можно удалить:

```
notifications.remove_notification_group(notification_group_id=my_group.notification_group_id)
```

Однако убедитесь, что вы не удаляете связанную группу уведомлений **сразу** после отправки сообщения, иначе вы можете его пропустить. Например, выполните приведенный ниже код:

```
notifications.publish(
    notification_group_name=my_group.name,
    payload={"friend": "SDK", "message": "Hi!"},
)
notifications.remove_notification_group(notification_group_id=my_group.notification_group_id)
```

не приведет ни к каким уведомлениям.

Полный пример

Полный пример кода приведен ниже:

```
import time

from platform_sdk import Notifications

notifications = Notifications()

my_group = notifications.create_notification_group(name="my_group")

notifications.add_notification_template_email(
    notification_group_id=my_group.notification_group_id,
    to=["<your_email>"],
    subject_template="Hello from ${friend}",
    message_template="${friend} sends you a message: ${message}",
    name="my_receiver1",
)

notifications.publish(
    notification_group_name=my_group.name,
    payload={"friend": "SDK", "message": "Hi!"},
)
```

```
# do not delete notification group immediately after sending a message!
time.sleep(10)
notifications.remove_notification_group(notification_group_id=my_group.notification_group_id)
```

Протоколы доставки

Список поддерживаемых протоколов доставки и соответствующие методы настройки шаблонов уведомлений приведены ниже:

Электронная почта	`add_notification_template_email`	Добавляет шаблон уведомления с протоколом доставки по электронной почте.
WebSocket	`add_notification_template_popup`	Добавляет шаблон уведомления с протоколом доставки через всплывающие окна.

Переменные окружения

В этом руководстве вы узнаете о переменных окружения SDK, необходимых для доступа к платформе.

Обзор

Установка пакета `platform_sdk` не предоставляет вам доступа к платформе по умолчанию. Вам необходимо задать минимальный набор переменных окружения с учетными данными пользователя на платформе.

Минимальный набор переменных окружения

При локальном использовании `platform_sdk` (в том числе для запуска приложений) необходимо указать как минимум 3 переменные окружения:

`AUTH_ENDPOINT`	Доменное имя или IP-адрес платформы, например, `test.Platform.ai` или `101.91.6.23`.
`AUTH_USERNAME`	Ваше имя пользователя.
`AUTH_CLIENT_SECRET`	Ваш пароль.

При запуске приложения на платформе эти учетные данные заполняются автоматически, поэтому нет необходимости указывать их вручную.

Другие переменные

Для дополнительных настроек можно указать и другие переменные окружения. Список поддерживаемых переменных окружения приведен ниже.

> ⓘ Примечание
>
> Обратите внимание, что этот список может быть неполным.

Как установить переменные окружения из входных параметров

В некоторых случаях требуется изменить переменную окружения для конкретного приложения. Например, приложение может выполнить один тяжелый запрос в начале, а затем перейти к легким запросам. В этом случае может потребоваться увеличить интервал таймаута для тяжелых запросов приложения. SDK предоставляет возможность указать переменные окружения для определенного приложения при одном условии - все указанные переменные окружения должны быть отмечены как разрешенные для указания из входных параметров приложения.

> ⓘ Примечание
>
> О том, как добавить пользовательские переменные в приложение, читайте в разделе Как добавить пользовательские переменные окружения в белый список ниже.
>
> Предположим, что `MYENV1` и `MYENV2` разрешено задавать из входных параметров приложения.

Для этого задайте входной параметр приложения `platform_sdk_ENV_VARS_INPUT_DICT` как JSON словарь из имен переменных окружения и их новых значений. Например, если пользователь установил:

```
"SDK_ENV_VARS_INPUT_DICT": {"MYENV1": "10", "MYENV2": "30"}
```

тогда `10` и `30` будут переданы как значения переменных окружения для `MYENV1` и `MYENV2`. Обратите внимание, что если переменной нет в белом списке, то она будет просто проигнорирована. В настоящее время SDK позволяет указывать следующие переменные окружения из `SDK_ENV_VARS_INPUT_DICT`:

```
['EVAL_DELIMITER', 'KEDRO_LOGGING_LEVEL', 'MAX_EVALUTOR_COOLDOWN_DICT_SIZE', 'MAX_EXF
```

Как добавить пользовательские переменные окружения в белый список

Если переменная окружения должна быть настроена через входные параметры приложения, то ее следует добавить в специальный класс с белым списком переменных окружения, который должен иметь в качестве метакласса специальный класс из библиотеки SDK. Например, если белый список находится по следующему пути в некоторой библиотеке: `library_name/constants.py`, то добавьте в код библиотеки следующий класс:

```
from platform_sdk import ConfigurableEnvConstants

class MyEnvVars(metaclass=ConfigurableEnvConstants):
    MY_ENV = 4
    MY_ENV_2 = 6
```

После этого переменные из этого класса можно использовать в других местах кода как обычные атрибуты класса. Переменные окружения `MY_ENV` и `MY_ENV_2` будут добавлены в белый список и будут загружаться из одноименных переменных окружения. Обратите внимание, что все буквы в имени переменной должны быть заглавными.

Журналирование

При помощи данного руководства вы получите общую информацию о журналировании, а также получите практический опыт, работая с событиями SDK, фильтрацией событий и клиентами категорий событий.

Предварительные условия

- Учетная запись на Платформе.
- SDK установлен на вашем компьютере.
- Указаны ref:*минимальные настройки <ev-min-setup>* переменных среды.

Обзор

Журналирование представляет собой журнал событий, предназначенный для записи и отображения информации об изменениях сигналов и связанных с ними действиях. Например, событие будет сгенерировано в случае предоставления неправильных данных сигнала в качестве ввода в приложение. Таким образом, пользователи могут получить подробную информацию о каждом событии и исследовать возникшие проблемы с сигналами.

> ⓘ Примечание
>
> Чтобы узнать больше о журналировании, обратитесь к документации платформы. Чтобы открыть документацию платформы, войдите в систему на Платформе, нажмите на значок пользователя в левом нижнем углу страницы и выберите **Документация платформы**.

Начнем с краткого введения основных сущностей журналирования.

Основные сущности

Категория события – это классификацию события, которая пользователям разграничивать и группировать события. Категории могут быть созданы или обновлены только через API.

Событие – это изменение сигналов и их состояния. Каждое событие включает следующие атрибуты:

- Приоритет
- public_id сигнала, связанного с событием
- Сообщение
- Категория

Фильтр событий используется для группировки событий, связанных с конкретными сигналами, на основе различных критериев, таких как приоритет, категория, дата и т. д.

Ниже представлен краткий обзор основных SDK клиентов журналирования.

Работа с сущностями журналирования

Категории событий

Предоставляет метод списка для категорий событий. Предоставленные категории могут быть созданы или изменены только через API.

```
from platform_sdk import EventCategoryClient

client = EventCategoryClient()
categories = client.list_categories()
```

События

Событие – это ключевая сущность для функциональности журналирования. Существует 2 способа создания события: - Запись нового событие напрямую - Запись событие как значение в сигнал

Вот пример записи события:

```
from platform_sdk import EventClient, EventSeverity

client = EventClient()
event1 = Event(signal_public_id=pid, severity=EventSeverity.MEDIUM, message="test", category="category2")
client.default_category = "<default_category>"
# for event2 default category will be applied
event2 = Event(signal_public_id=pid, severity=EventSeverity.MEDIUM, message="test")
client.record([event1, event2])
```

Если событие нужно записать как значение сигнала, следует использовать другой метод:

```
from platform_sdk import EventClient, EventSeverity, JournalSignalValue, EventSignalType

client = EventClient()
filter_pid, pid = journal_filter_with_signal
event_values = [
    JournalSignalValue(
        category="category2", message="test", severity=EventSeverity.MEDIUM, value=0.0, timestamp=datetime.dat
    ),
    JournalSignalValue(
        category="category2", message="test", severity=EventSeverity.MEDIUM, value=1.0, timestamp=datetime.dat
    ),
]
client.write_to_journal_signal(public_id=pid, signal_type=EventSignalType.JOURNALED_ROW, journal_values=event_
```

Фильтры событий

Фильтр событий используется для группировки событий, связанных с конкретными сигналами. События могут быть сгруппированы по приоритету, категориям, сигналам и определенной дате. Кроме того, фильтры могут содержать различные комбинации таких критериев. Работа с фильтрами событий показана ниже:

```python
from platform_sdk import EventClient, EventSeverity, EventFilterClient

journal_filter = EventFilterClient()

signal_pid = "<signal-pid>"
filter_pid = "<filter-pid>"
journal_filter.create(
    public_id=filter_pid,
    name="<name>",
    signal_public_ids=[signal_pid],
    categories=["<category>"],
    severities=[<EventSeverity>],
    threshold_dt=datetime.datetime.now(pytz.UTC),
)
filter = journal_filter.get(filter_pid)
event = Event(signal_public_id=signal_pid, severity=EventSeverity.MEDIUM, message="test", category="<category
events.record([event])
events = events.list_events(filter_public_id=filter["public_id"], threshold_dt=None)

# if filter is no longer required
journal_filter.delete(filter_pid)
```

Логирование

В этом руководстве вы узнаете, как настроить и использовать логирование в SDK.

Предварительные условия

- Локально установленная библиотека platform SDK.
- Минимальная настройка переменных окружения.
- Базовое понимание стандартного логирования в python.

Обзор

Для логирования в SDK используется библиотека `loguru`, которая поддерживает 6 уровней логирования:

- `CRITICAL`
- `ERROR`
- `WARNING`
- `INFO`
- `DEBUG`
- `TRACE` (регистрация всех запросов к платформе)

По умолчанию SDK использует уровень логирования `INFO` и сохраняет журналы в каталоге `/tmp/`. Вы можете установить любой уровень логирования для всех модулей или указать другой уровень логирования для отдельного модуля.

SDK поддерживает три способа конфигурации уровня логирования:

1. Переменная окружения LOGGER_LEVEL: через переменную окружения `LOGGER_LEVEL`
2. Входной параметр LOGGER_LEVEL: через входные параметры ML-приложения (полезно при использовании существующего приложения)

3. Из кода: внутри кода (полезно, если необходимо установить различный уровень логирования для конкретного модуля)

Логирование на платформе

При создании Docker контейнера с приложением необходимо установить следующую переменную окружения:

```
export LOGGER_JSON=True
```

В результате лог будет иметь требуемый необходимый для платформы JSON формат.

Можно ли изменить логирование в коде?

Да, вы можете полностью управлять логированием в своем коде, для этого достаточно импортировать объект `logger` из библиотеки `loguru` и добавить новые обработчики:

```
from loguru import logger

logger.remove()  # remove all existing handlers
logger.add(sys.stderr, level=log_level)
```

Переменная окружения LOGGER_LEVEL

Уровень логирования устанавливается при первом вызове SDK и определяется переменной окружения `LOGGER_LEVEL`. Установить эту переменную можно следующим образом:

```
export LOGGER_LEVEL=DEBUG
```

Входной параметр LOGGER_LEVEL

Если у вас нет доступа к переменным окружения или вы запускаете существующее ML-приложение, вы можете задать уровень логгера, используя дополнительный входной параметр `LOGGER_LEVEL`:

```
{
  "fieldType": "TEXT",
  "name": "LOGGER_LEVEL",
  "label": "LOGGER_LEVEL",
  "isRequired": true,
  "description": "Logging level for the SDK. Could be one of: CRITICAL, ERROR, WARNING, INFO, DEBUG",
  "defaultValue": "INFO"
}
```

Вы можете легко добавить этот параметр к существующей схеме входных параметров. Вам не нужно поддерживать этот параметр внутри вашего приложения, SDK будет обрабатывать его автоматически. (Если вы не знаете, что такое входные параметры приложения, пожалуйста, просмотрите руководство по созданию первого приложения).

Из кода

Если вы хотите установить другой уровень логирования для своего кода, то вам необходимо установить соответствующий уровень логирования *до* импорта классов и модулей из SDK:

```
import os

os["LOGGER_LEVEL"] = "DEBUG"

from platform_sdk import SignalsOps
...
```

Хартбит

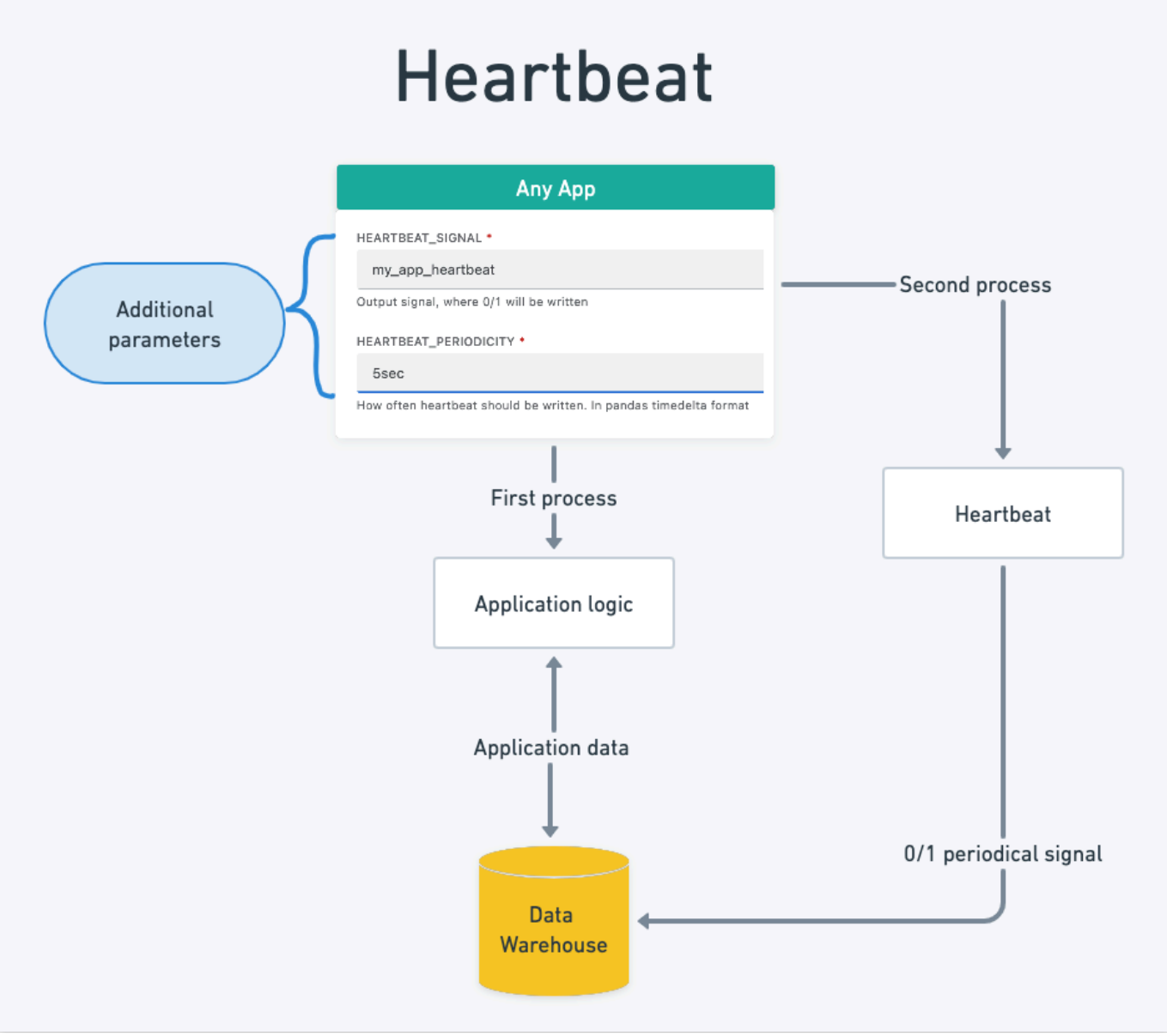
SDK предоставляет инструмент для мониторинга хартбита приложения. Хартбит является индикатором работоспособности приложения. Он представляет собой `ROW` сигнал, который непрерывно принимает значения 0 и 1 одно за другим. Если значения обновляются с фиксированной частотой без разрыва цепочки 0/1, значит, приложение работает.

Существует два способа использования мониторинга хартбита:

1. Стандартный хартбит
2. Гибкий хартбит

Стандартный хартбит

Принцип работы стандартного хартбита приведен на диаграмме ниже:



SDK может запускать хартбит в отдельном процессе при запуске приложения, для чего необходимо добавить два следующих входных параметра:

```
{
    "fields": [
        {
          "fieldType": "TEXT",
          "name": "HEARTBEAT_SIGNAL",
          "label": "HEARTBEAT_SIGNAL",
          "isRequired": true,
          "description": "Output signal, where 0/1 will be written"
        },
        {
          "fieldType": "TEXT",
          "name": "HEARTBEAT_PERIODICITY",
          "label": "HEARTBEAT_PERIODICITY",
          "isRequired": true,
          "description": "How often heartbeat should be written. In pandas timedelta format"
        }
    ]
}
```

Гибкий хартбит

Если вы хотите сделать хартбит более интеллектуальным, вы можете использовать его вручную в основном цикле приложения. SDK предоставляет класс `Heartbeat`, который может быть использован внутри пользовательского приложения для отправки сигнала пульса в конце каждого логического цикла. Например:

```
from platform_sdk import (
    ParameterizedPipelineStep,
    SignalStreaming,
    Heartbeat,
    SignalWriterClient,
    SignalsOps
)


class MyApp(ParameterizedPipelineStep):

    def __init__(signal_streaming: SignalStreaming, writer: SignalWriterClient, signals_ops: SignalsOps):
        self._signal_streaming = signal_streaming
        self._writer = writer
        self._signals_ops = signals_ops

    def _start(self, self, task_id: int, job_params: JobParamsSchema, input_params: InputParamsSchema):
        self._heartbeat = Heartbeat(output_signal, self._writer, self._signals_ops)

        for snapshot in self._signal_streaming.subscribe_to_raw_updates(
                signal_public_ids=input_params.public_ids, history_size=input_params.batch_size, batch_interva
            ):
            self.do_main_logic()
            self._heartbeat.write()
```

Период тишины хартбита

Чтобы хартбит отправлял данные только через определенное количество итераций после запуска, можно использовать „heartbeat_startup_silence_period“. Просто установите требуемое количество итераций в данный параметр:

```
from platform_sdk import (
    ParameterizedPipelineStep,
    SignalStreaming,
    Heartbeat,
    SignalWriterClient,
    SignalsOps,
    SignalsGroupOps
)


class MyApp(ParameterizedPipelineStep):

    def __init__(signal_streaming: SignalStreaming, writer: SignalWriterClient, signals_ops: SignalsOps, signa
        self._signal_streaming = signal_streaming
        self._writer = writer
        self._signals_ops = signals_ops
        self._signal_group_ops = signal_group_ops


    def _start(self, self, task_id: int, job_params: JobParamsSchema, input_params: InputParamsSchema):
        self._heartbeat = Heartbeat(output_signal, self._writer, self._signals_ops, self._signal_group_ops, he

        for snapshot in self._signal_streaming.subscribe_to_raw_updates(
                signal_public_ids=input_params.public_ids, history_size=input_params.batch_size, batch_interva
            ):
            self.do_main_logic()
            self._heartbeat.write()
```

Первые 5 „write“ вызовов хартбита не будут отправлять данные. Первый хартбит будет отправлен на 6-ом вызове „write“.

Встроенные приложения

SDK платформы поставляется с несколькими мощными приложениями, которые доступны прямо из коробки. Эти приложения по умолчанию служат для мониторинга качества данных на этапах подготовки и развертывания, а также для расширения функциональности платформы.

Все встроенные приложения могут быть запущены и использованы с интерфейса Платформы.

Требования к ресурсам

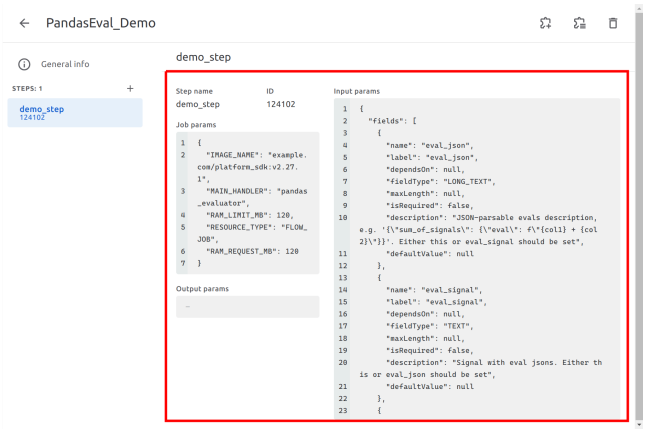
Все встроенные приложения оптимизированы для минимизации потребления оперативной памяти, независимо от количества обрабатываемых сигналов. В среднем для обработки до 100 сигналов для чтения и 10 сигналов для записи большинству приложений по умолчанию требуется всего 192 МБ оперативной памяти. Более подробную информацию о требованиях к ресурсам отдельных приложений можно получить в документации к ним.

> ⓘ Примечание
>
> Чтобы избежать излишней нагрузки на платформу, обязательно укажите необходимый объем оперативной памяти в полях `RAM_REQUEST_MB` и `RAM_LIMIT_MB` в схеме параметров задания.

Пример использования

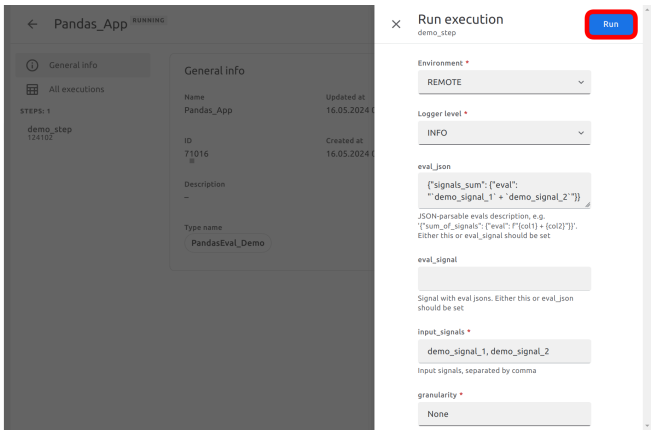
Данный небольшой пример демонстрирует как запустить встроенное приложение на Платформе. Это приложение будет настроено в соответствии с руководством PandasEvaluator, чтобы использовать выражение для добавления значений нескольких сигналов. Таким образом, два сигнала типа *ROW* со специальными значениями созданы для этой демонстрации (`demo_signal_1` и `demo_signal_2`).

Сначала необходимо создать новый *шаблон приложения* на Платформе. Далее необходимо создать новый **шаг** в созданном шаблоне и предоставить `параметры процесса выполнения` и `входные параметры` как показано на скриншоте:



После настройки шаблона, создайте новое *приложение* с этим шаблоном на Платформе. Затем необходимо запустить и настроить шаг согласно инструкциям. Данное *eval_json* выражение используется для сложения значений сигналов: `{"signals_sum": {"eval": "`demo_signal_1` + `demo_signal_2`"}}` .

Далее щелкните по кнопке **Запустить**, чтобы запустить настроенный шаг:



Следуйте той же логике для настройки и использования других встроенных приложений. Однако, обратите внимание, что встроенные приложения могут иметь другие требования, такие как создание групп уведомлений или использование артефактов.

Доступные приложения

Все доступные приложения по умолчанию перечислены ниже:

Data player

В этом руководстве вы познакомитесь с основами работы приложения data player, а также с конфигурацией данного приложения из пользовательского интерфейса платформы.

Необходимые условия

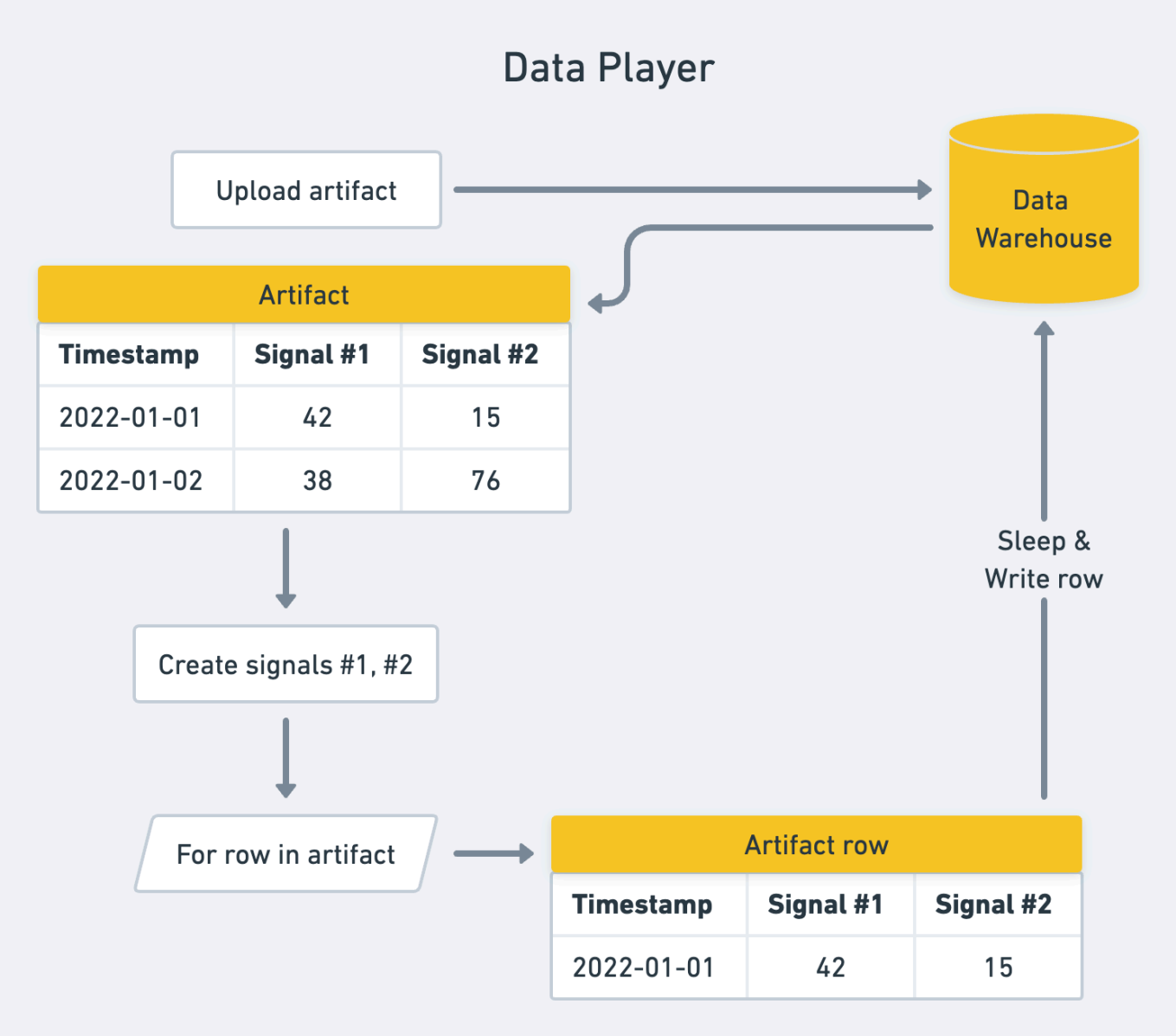
- Аккаунт на платформе Тайга.
- Знакомство с понятиями «приложения», «сигналы» и «артефакты». Прочитать соответствующие разделы документации платформы. Для доступа к документации платформы войдите в систему, нажмите на значок пользователя в левом нижнем углу страницы и выберите пункт «Документация по платформе».
- Настроенный реестр Docker с образом `platform-sdk:latest`. Если вы не знакомы с реестрами Docker, мы рекомендуем начать с этого руководства.

Обзор

Data player - это приложение для создания и заполнения сигналов данными, хранящимися в файле. Запущенный data player перебирает строки данных в файле и многократно записывает значения в указанные источники вывода. Это эмулирует поведение реальных данных, поступающих из внешних источников, например, датчиков, что позволяет тестировать модели и приложения в производственной среде.

Все плееры имеют одинаковый интерфейс, разница лишь в формате потребляемого файла. Существует два плеера данных: CSV Data player и Parquet Data player.

Диаграмма приложения показана ниже:



Требования к ресурсам

Если размер артефакта данных не превышает 10 Мбайт, то для записи до 100 сигналов требуется всего 192 Мбайт оперативной памяти. Если необходимо записывать более 100 сигналов одновременно или записывать данные пакетно (т.е. более одной строки за раз), то просто удвойте требуемое количество оперативной памяти. При работе с большими артефактами данных требуемое количество оперативной памяти можно рассчитать следующим образом:

- Для CSV-плеера: 192 МБ плюс 3-кратный размер артефакта. Например, для артефакта размером 100 МБ потребуется 192 МБ + 3 * 100 МБ = 492 МБ оперативной памяти.
- For a Parquet player: Writing 100 signals with 100000 values takes 330 Mb. It is recommended to set limit to 386 Mb in order to reserve 20 percents of RAM. Less artifact size takes less memory, so ram usage depends on artifact and should be based on specific player input.

Запустите приложение из пользовательского интерфейса платформы.

Ниже приведены шаги для запуска приложения:

1. Загрузка артефактов в платформу: загрузка выборочных данных, содержащих 10 произвольных значений на сигнал для 2 сигналов.
2. Создать и запустить приложение: создание шаблона плеера и запуск плеера с указанными данными.
3. Проверьте результаты: убедитесь, что сигналы созданы и заполнены данными.

Загрузка артефактов в платформу

Сначала необходимо загрузить файлы данных для воспроизведения в платформу.

> ⓘ Примечание
>
> Если вы используете FlatParquetPlayer, убедитесь, что в индексе файла .parquet записаны дата и время в формате `datetime`. Для FlatCsvPlayer файл .csv должен содержать дату и время в ISO формате в первом столбце.

1. В зависимости от выбранного типа плеера загрузите соответствующие файлы данных:

   в формате CSV

   `sample_1.csv` `sample_2.csv`

   или в формате Parquet

   `sample_1.parquet` `sample_2.parquet`

   Приведенные файлы содержат примеры данных для 2 сигналов:

   sample_1.csv

   ```
   timestamp,test_signal_1,test_signal_2
   2022-01-01 12:00:00,1,2
   2022-01-01 13:00:00,2,4
   2022-01-01 14:00:00,3,6
   2022-01-01 15:00:00,4,8
   2022-01-01 16:00:00,5,10
   ```

   sample_2.csv

   ```
   timestamp,test_signal_1,test_signal_2
   2022-01-02 12:00:00,6,12
   2022-01-02 13:00:00,7,14
   2022-01-02 14:00:00,8,16
   2022-01-02 15:00:00,9,18
   2022-01-02 16:00:00,10,20
   ```

   Содержимое файлов CSV и Parquet идентично.

2. Войдите в платформу и перейдите в раздел **Data** → **Artifacts**.
3. Нажмите кнопку **Загрузить** в правом верхнем углу окна.
4. В открывшемся диалоговом окне введите имя артефакта и ссылку на первый файл, затем нажмите кнопку **Создать**.

5. Дождитесь загрузки артефакта и появления небольшого окна с идентификатором артефакта. Запомните идентификатор артефакта, он пригодится вам в дальнейшем.
6. Повторите действия со вторым файлом.

Создать и запустить приложение

Следующим шагом является создание и запуск приложения.

> ⓘ Примечание
>
> Подробные инструкции по регистрации и запуску приложений на платформе приведены в документации платформы. Чтобы открыть документацию платформы, войдите в платформу, нажмите значок пользователя в левом нижнем углу страницы и выберите **Документация по платформе**.

1. Создайте новый шаблон приложения, дайте ему осмысленное имя и запомните его для дальнейшего использования.
2. Создайте новый шаг с произвольным именем. Оставьте значение по умолчанию в поле **Страница** и укажите следующие два параметра. Нажмите кнопку **Сохранить** под каждым параметром.

   `Job params`:

   ```
   {
       "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
       "MAIN_HANDLER": "parquet_data_player",
       "RESOURCE_TYPE": "STEP_JOB",
       "RAM_REQUEST_MB": 386,
       "RAM_LIMIT_MB": 386
   }
   ```

   > ⓘ Примечание
   >
   > Если вы используете CSV-плеер, установите для `MAIN_HANDLER` значение `csv_data_player`.

   `Input params`:

   ```
   {
     "fields": [
       {
         "fieldType": "TEXT",
         "name": "artifact_ids",
         "label": "artifact_ids",
         "isRequired": true
       },
       {
         "fieldType": "TEXT",
         "name": "columns_to_write",
         "label": "columns_to_write",
         "isRequired": false
       },
       {
         "fieldType": "TEXT",
         "name": "signals_to_write",
   ```

```
      "label": "signals_to_write",
      "isRequired": false
    },
    {
      "fieldType": "TEXT",
      "name": "signals_group",
      "label": "signals_group",
      "isRequired": false
    },
    {
      "fieldType": "NUMBER",
      "name": "batch_size",
      "label": "batch_size",
      "isRequired": false,
      "defaultValue": "1"
    },
    {
      "fieldType": "TEXT",
      "name": "to_sleep_between_writes",
      "label": "to_sleep_between_writes",
      "isRequired": true,
      "defaultValue": "10s"
    },
    {
      "fieldType": "TEXT",
      "name": "time_delta_between_files",
      "label": "time_delta_between_files",
      "isRequired": true,
      "defaultValue": "1min"
    },
    {
      "fieldType": "TEXT",
      "name": "is_fill_na",
      "label": "is_fill_na",
      "isRequired": true,
      "defaultValue": "true"
    },
    {
      "fieldType": "TEXT",
      "name": "is_inf",
      "label": "is_inf",
      "isRequired": true,
      "defaultValue": "false"
    },
    {
      "fieldType": "TEXT",
      "name": "write_each_in_now_timestamp",
      "label": "write_each_in_now_timestamp",
      "isRequired": true,
      "defaultValue": "false"
    },
    {
      "fieldType": "TEXT",
      "name": "start_datetime",
      "label": "start_datetime",
      "isRequired": false,
      "defaultValue": null
    }
  ]
}
```


`LOGGER_LEVEL` type: string

Уровень логирования. Более подробная информация приведена в руководстве по логированию.

`artifact_ids` type: string

	Список идентификаторов артефактов с загруженными файлами, разделенный запятыми.
`columns_to_write`	`type: string` `optional` Список столбцов для выбора из файла, разделенный запятыми. Если None или пустая строка, то используются все столбцы.
`signals_to_write`	`type: string` `optional` Список публичных идентификаторов сигналов, разделенный запятыми. Если не None, создается отображение между выбранными именами столбцов и публичными идентификаторами. В противном случае используются выбранные имена столбцов.
`signals_group`	`type: string` `optional` Группа для закрепления всех выходных сигналов.
`batch_size`	`type: number` Количество строк для одновременной записи в одну партию.
`to_sleep_between_writes`	`type: string` Задержка между записью двух последующих строк или пачек в сигналы в формате pandas.Timedelta.
`time_delta_between_files`	`type: string` Задержка между временными метками последней строки одного файла и первой строки следующего в формате pandas.Timedelta. Это влияет только на временные метки и не добавляет задержку между файлами.
`is_fill_na`	`type: boolean` Если значение True, то значения `NaN` заполняются нулями. В противном случае все значения `NaN` пропускаются.
`is_inf`	`type: boolean` Если значение True, то все файлы будут воспроизводиться бесконечно.
`write_each_in_now_timestamp`	`type: boolean` Если значение True, то данные записываются с той же временной меткой, что и момент выполнения

	действия записи. В противном случае все временные метки сдвигаются к начальному времени (или к now, если начальное время равно None) с сохранением исходной временной разницы между строками.
`start_datetime`	`type: string` Если дата начала в формате ISO 8601 отсутствует или раньше текущей даты, то запуск происходит мгновенно. В противном случае перед запуском необходимо дождаться этой даты.

> ⓘ Примечание
>
> Плеер может записывать данные только с равномерной частотой, определяемой параметром `to_sleep_between_writes`. Пользователь имеет возможность сохранить временную дельту, как в исходных данных, установив `write_each_in_now_timestamp` в False, но временные метки все равно будут смещены так, чтобы первая из них была равна `start_datetime` (или now, если `start_datetime` - None). Если вам нужно, чтобы плеер записывал данные в прошлое, просто установите `write_each_in_now_timestamp` в False и `start_datetime` в желаемое время в прошлом.

3. Создайте новое приложение из шаблона, которое только что создано и запустите его со следующих параметров:

`Environment`	`REMOTE`
`LOGGER_LEVEL`	`INFO`
`Artifact_ids`	Идентификаторы артефактов файлов, загруженных выше, разделенные запятыми, например `101,102`.
`Columns_to_write`	`test_signal_1,test_signal_2`
`Signals_to_write`	`player_test_signal_1,player_test_signal_2`
`Batch_size`	1
`To_sleep_between_writes`	`10s`
`Time_delta_between_files`	`1min`
`Is_fill_na`	`true`
`Is_inf`	`false`

`Write_each_in_now_timestamp`	`false`
`Start_datetime`	Оставить пустым.

4. Дождитесь изменения статуса на `ЗАВЕРШЕНО`, что означает успешное выполнение приложения. Если статус приложения `СБОЙ`, то можно зайти внутрь шага и изучить логи.

Проверьте результаты

Если приложение завершилось, как ожидалось, то вы должны обнаружить 2 новых сигнала: `test-player_test_signal_1` и `test-player_test_signal_2`.

- Убедитесь, что оба сигнала содержат те же данные, что и в загруженных артефактах.
- Убедитесь, что между каждой парой записей был промежуток в 10 секунд.

Поздравляю! Вы только что освоили основы работы с плеером данных!

Дополнительно

Для плеера данных существует несколько дополнительных скрытых параметров, которые могут оказаться полезными:

1. `output_prefix` - префикс для всех выходных сигналов

```
{
  "fieldType": "TEXT",
  "name": "output_prefix",
  "label": "output_prefix",
  "isRequired": false,
  "defaultValue": ""
}
```

2. `chunk_size` - размер отдельного чанка для памяти. Данные из parquet или csv загружаются в память чанк за чанком. Чем меньше размер чанка - тем меньше потребление памяти игроком. Но будьте внимательны, `chunk_size` должен быть выше, чем `batch_size`

```
{
  "fieldType": "NUMBER",
  "name": "chunk_size",
  "label": "chunk_size",
  "isRequired": false,
  "defaultValue": 64000
}
```

Expectations

В данном руководстве рассматриваются основы применения приложения Expectations, а также конфигурацию данного приложения в пользовательском интерфейсе платформы.

Необходимые условия

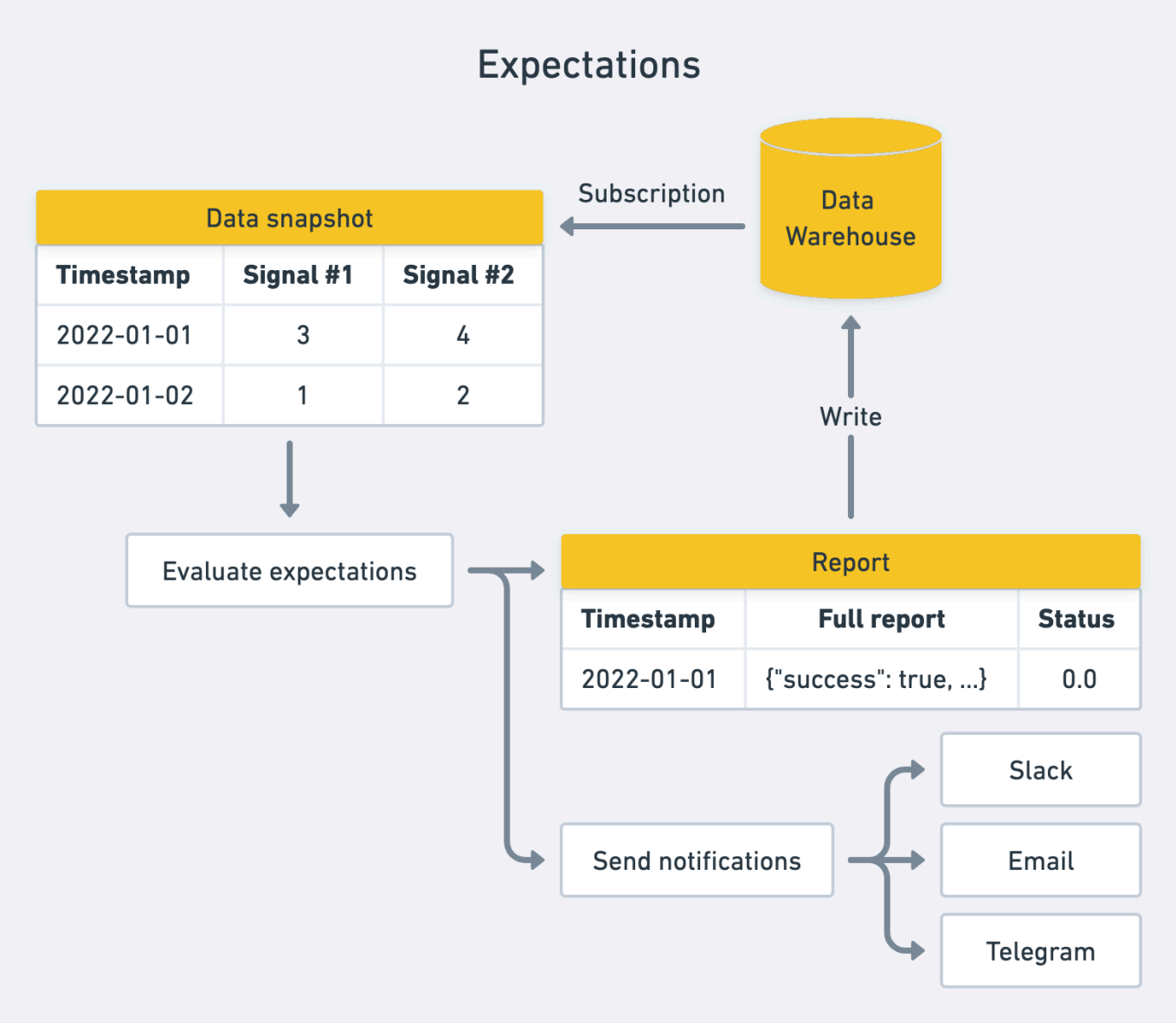
- Аккаунт на платформе Тайга.
- Знакомство с понятиями приложений, сигналов и уведомлений. Прочитайте соответствующие разделы документации платформы. Чтобы получить доступ к документации платформы, войдите в платформу, нажмите на значок пользователя в левом нижнем углу страницы и выберите «Документация платформы».
- Знакомство с основными концепциями Great Expectations. Если вы никогда раньше не использовали Great Expectations, мы настоятельно рекомендуем ознакомиться с разделом данного руководства Great Expectations quickstart, прежде чем двигаться дальше.
- Настроенный реестр Docker с образом `platform-sdk:latest`. Если вы не знакомы с реестрами Docker, мы рекомендуем начать с этого руководства.

Обзор

Expectations - это приложение-обертка по умолчанию для функциональности Great Expectations. Оно позволяет пользователям рассчитывать заданные ожидания в режиме реального времени на поступающих данных или сразу на большом батче исторических данных.

Если ожидания не выполняются, приложение отправляет оповещения по электронной почте, в Slack или в виде всплывающего окна на платформе.

Диаграмма приложения показана ниже:



Требования к ресурсам

Для мониторинга до 100 сигналов требуется всего 192 Мбайт оперативной памяти. Если необходимо отслеживать более 100 сигналов одновременно, достаточно увеличить объем оперативной памяти в два раза или запустить второй экземпляр приложения.

Запустите приложение из пользовательского интерфейса платформы.

Ниже приведен пример настройки и запуска приложения:

1. Создание сигналов: Создание двух сигналов с выборочными данными.
2. (Необязательно) Создать группу уведомлений: Создание и настройка группы уведомлений для получения сообщений электронной почты от приложения.
3. Создать и запустить приложение: Создание шаблона ожиданий и запуск приложения.
4. Проверьте результаты: Проверка того, что приложение пишет статусы и отчеты для указанных ожиданий.

В данном примере мы запускаем приложение ожиданий в реальном времени, оценивающее три ожидания для созданных сигналов:

- Ожидать, что первый сигнал будет иметь большинство своих значений в заданном диапазоне.
- Ожидать, что среднее значение первого сигнала будет находиться в более узком заданном диапазоне.
- Ожидать, что второй сигнал будет иметь не менее X% уникальных значений.

Создание сигналов

Прежде всего, нам необходимо создать 2 сигнала с выборочными данными (по 10 значений в каждом).

1. Загрузите файл `signals.csv`.

   Содержимое этого файла можно воспроизвести, выполнив приведенный ниже сценарий на языке Python:

   ```
   timestamp,ge_signal_1,ge_signal_2
   2022-01-31 12:00:00,7,42
   2022-01-31 13:00:00,4,42
   2022-01-31 14:00:00,7,42
   2022-01-31 15:00:00,5,43
   2022-01-31 16:00:00,4,42
   2022-01-31 17:00:00,1,42
   2022-01-31 18:00:00,2,42
   2022-01-31 19:00:00,3,42
   2022-01-31 20:00:00,4,42
   2022-01-31 21:00:00,8,42
   ```

   > ⓘ Примечание
   >
   > Как видно, все значения первого сигнала (`ge_signal_1`) попадают в диапазон *[1; 8]* со средним значением, равным *4,5*. Второй сигнал (`ge_signal_2`) имеет 1 значение без дубликатов (что составляет 10% от 10 общих значений).

2. Войдите в платформу и перейдите в раздел **Данные** → **Коннекторы**.
3. Нажмите на кнопку **Импортировать данные сигнала** в правом верхнем углу страницы.
4. В открывшемся диалоговом окне загрузите файл, установите флажок **Пользовательский формат даты** и задайте **Формат даты** равным `yyyy-MM-dd HH:mm:ss`. Установите разделитель в виде запятой (`,`) и нажмите **Импортировать**.

Создать группу уведомлений

Вы можете создать группу уведомлений либо на Платформе, либо из кода. Если вы решили пойти первым путем, обратитесь к документации Платформы. В противном случае обратитесь к Руководству по уведомлениям.

Создать и запустить приложение

Следующим шагом является создание и запуск приложения.

> ⓘ Примечание
>
> Подробные инструкции по регистрации и запуску приложений на платформе приведены в документации платформы. Чтобы открыть документацию платформы, войдите в платформу, нажмите значок пользователя в левом нижнем углу страницы и выберите **Документация по платформе**.

1. Создайте новый шаблон приложения, дайте ему осмысленное имя и запомните его для дальнейшего использования.
2. Создайте новый шаг с произвольным именем. Оставьте значение по умолчанию в поле **Страница** и укажите следующие два параметра. Нажмите кнопку **Сохранить** под каждым параметром.

   `Job params`:

   ```
   {
       "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
       "MAIN_HANDLER": "great_expectations_evaluate",
       "RESOURCE_TYPE": "FLOW_JOB",
       "RAM_REQUEST_MB": 192,
       "RAM_LIMIT_MB": 192
   }
   ```

   `Input params`:

   ```
   {
     "fields": [
       {
         "fieldType": "LONG_TEXT",
         "name": "expectations",
         "label": "expectations",
         "isRequired": true,
         "description": "JSON-parsable list of expectations, e.g. '[{\"expectation_type\": \"expect_colum
       },
       {
         "fieldType": "TEXT",
         "name": "aggregation_window",
         "label": "aggregation_window",
         "isRequired": true,
         "defaultValue": "1 min"
       },
       {
         "fieldType": "TEXT",
         "name": "status_signal",
         "label": "status_signal",
         "isRequired": true
       },
       {
         "fieldType": "TEXT",
         "name": "report_signal",
         "label": "report_signal",
         "isRequired": true
       },
       {
         "fieldType": "TEXT",
         "name": "notification_group_name",
         "label": "notification_group_name",
         "isRequired": false,
         "description": "Existing notification group id, e.g. 'my-group'"
       },
       {
         "fieldType": "NUMBER",
         "name": "max_patience",
         "label": "max_patience",
   ```

```
      "isRequired": false,
      "defaultValue": 10,
      "description": "Max failed reports in a row to throw error status"
    },
    {
      "fieldType": "TEXT",
      "name": "batch_interval",
      "label": "batch_interval",
      "isRequired": false,
      "defaultValue": "10 s",
      "description": "Timedelta between expectation reports in pandas.Timedelta format"
    },
    {
      "fieldType": "TEXT",
      "name": "heartbeat_internal_signal",
      "label": "heartbeat_internal_signal",
      "isRequired": false,
      "description": "Signal for writing internal smart heartbeat of app"
    },
    {
      "fieldType": "TEXT",
      "name": "heartbeat_startup_silence_period",
      "label": "heartbeat_startup_silence_period",
      "isRequired": false,
      "description": "How many iterations to wait until starting to send heartbeat"
    },
    {
      "fieldType": "TEXT",
      "name": "journaling_on",
      "label": "journaling_on",
      "isRequired": false,
      "description": "Turns on event writing",
       "defaultValue": "False"
    },
    {
      "fieldType": "TEXT",
      "name": "event_category",
      "label": "event_category",
      "isRequired": false,
      "description": "Default event category for events"
    }
  ]
}
```


`LOGGER_LEVEL` — `type: string`

Уровень логирования. Более подробную информацию можно найти в руководстве Логирование.

`expectations` — `type: string`

Список ожиданий в формате Great Expectations. Обратите внимание, что каждое значение `kwargs.column` должно быть равно публичному идентификатору существующего сигнала. Вы также можете указать имя группы сигналов как `kwargs.group` вместо `kwargs.column`, чтобы применить ожидание к каждому сигналу в группе. Если указаны оба значения, то `kwargs.column` игнорируется в пользу `kwargs.group`.

`aggregation_window` — `type: string`

	Значение в формате `pandas.Timedelta` размера истории subscription, например, `12 H`, `5 min`, `3 seconds`.
`status_signal`	`type: string` Публичный идентификатор сигнала `ROW` для получения статуса для общего отчета. Отчет считается успешным только в том случае, если все ожидания успешны. Обратите внимание, что можно также указать сигналы статуса для каждого отдельного ожидания, что более подробно рассматривается в разделе руководства настройка отчета.
`report_signal`	`type: string` Публичный идентификатор сигнала `BLOB` для получения полного отчета об ожиданиях.
`notification_group_name`	`type: string` `optional` Запятые имена существующих групп уведомлений, например, `my-group,my-group2`.
`max_patience`	`type: number` `optional` Максимальное количество неудачных отчетов в строке, при котором выдается статус ошибки. Отчет считается неудачным, если хотя бы одно из ожиданий оказалось неудачным.
`batch_interval`	`type: string` `optional` Временная интервал между проверками на наличие новых данных в формате pandas.Timedelta.
`heartbeat_internal_signal`	`type: string` Дополнительный сигнал для внутреннего интеллектуального подтверждения приложения. Если указано, приложение будет записывать пульс в конце каждой итерации цикла.
`journaling_on`	|type-bool| `optional` Позволяет записывать события, если ожидания не были выполнены. По умолчанию установлено в FALSE. Только ожидания со статусами WARNING и ERROR будут генерироваться как события (Warining – представляет события с уровнем критичности Medium, Error - для событий с уровнем Critical).

`event_category` `type: string` `optional`

Категория событий при остановке сигнала.

> ⓘ Примечание
>
> Чтобы избежать спама, ожидания рассылают уведомления только при изменении состояния потока данных, например:
>
> - *Failed expectations: [„signal_1_expect_column_values_to_be_between“]*
> - *No failed expectations*
> - *Failed expectations: [„signal_2_expect_column_values_to_be_between“]*
>
> и никогда не отправляет одни и те же сообщения подряд.

3. Создайте новое приложение на основе только что созданного шаблона и запустите его со следующими параметрами:

   Заполните поле `expectations` приведенной ниже конфигурацией:

```
[
    {
        "meta": {
            "status_signal": "ge_signal_1_values_between_status",
            "report_rules": {
                "1": {
                    "text": "ge_signal_1: all values are in expected range",
                    "type": "status"
                },
                "0": {
                    "text": "ge_signal_1: value range violation",
                    "type": "warning"
                },
                "-1": {
                    "text": "ge_signal_1: signal repeatedly violates the range",
                    "type": "error"
                },
                "-2": {
                    "text": "Application crashed or terminated",
                    "type": "status"
                }
            }
        },
        "expectation_type": "expect_column_values_to_be_between",
        "kwargs": {
            "column": "ge_signal_1",
            "min_value": 0,
            "max_value": 10,
            "mostly": 0.95
        }
    },
    {
        "expectation_type": "expect_column_mean_to_be_between",
        "kwargs": {
            "column": "ge_signal_1",
            "min_value": 4,
            "max_value": 6
        }
    },
    {
        "expectation_type": "expect_column_values_to_be_unique",
        "meta": {
            "status_signal": "ge_signal_2_values_between_status"
        },
        "kwargs": {
            "column": "ge_signal_2",
```

```
            "mostly": 0.05
        }
    }
]
```

> ⓘ Примечание
>
> Пожалуйста, прочитайте раздел данного руководства expectation settings details для получения дополнительной информации о том как корректно конфигурировать ожидания.

Заполните все оставшиеся поля следующими значениями:

`Environment`	`REMOTE`
`LOGGER_LEVEL`	`INFO`
`aggregation_window`	`1D`
`status_signal`	`ge_status`
`report_signal`	`ge_report`
`notification_group_name`	Имя группы уведомлений, созданной выше.
`max_patience`	`10`
`batch_interval`	`10 s`
`heartbeat_internal_signal`	`expectations_heartbeat`

4. Дождитесь, пока приложение изменит свой статус на `ВЫПОЛНЯЕТСЯ`, затем перейдите на страницу выполнения шага и обновляйте ее до тех пор, пока не увидите, что эта строка повторяется хотя бы один раз:

```
... ...
... INFO     Wrote status codes ...
... INFO     Wrote report to report signal 'ge_report'
```

После этого вы можете безопасно завершить этот шаг.

Проверьте результаты

Чтобы убедиться в том, что работа приложения завершилась должным образом, выполните следующие действия:

- Проверьте свою электронную почту на наличие сообщения с текстом «No failed expectations».

- Перейдите на страницу **Factory setup** → **Signals** и найдите сигнал `ge_status`. Убедитесь, что он имеет хотя бы одно значение `1` (статус успеха) и что его последнее значение равно `-2` (так как приложение деактивировано).
- Аналогично проверьте, что оба сигнала состояния для отдельных ожиданий, `ge_signal_1_values_between_status` и `ge_signal_2_values_between_status`, имеют одинаковые значения.
- Кроме того, найдите сигнал `ge_report` и убедитесь, что он содержит все перечисленные ниже сообщения:

```
"No failed expectations"
"ge_signal_1: all values are in expected range"
"Signal 'ge_signal_1' meets the expectation 'expect_column_mean_to_be_between'"
"Signal 'ge_signal_2' meets the expectation 'expect_column_values_to_be_unique'"
```

Поздравляю! Вы только что освоили основы ожиданий!

Особенности задания ожиданий

Установка ожиданий производится с довольно большим количеством настроек и может быть сложным на первый взгляд. Ниже, вы найдете несколько советов и дополнительной информации о том, как правильно настраивать ожидания.

Для начала приведем пример задания ожиданий:

```
[
    {
        "meta": {
            "status_signal": "ge_signal_1_values_between_status",
            "report_rules": {
                "1": {
                    "text": "ge_signal_1: all values are in expected range",
                    "type": "status"
                },
                "0": {
                    "text": "ge_signal_1: value range violation",
                    "type": "warning"
                },
                "-1": {
                    "text": "ge_signal_1: signal repeatedly violates the range",
                    "type": "error"
                },
                "-2": {
                    "text": "Application crashed or terminated",
                    "type": "status"
                }
            }
        },
        "expectation_type": "expect_column_values_to_be_between",
        "kwargs": {
            "column": "ge_signal_1",
            "min_value": 0,
            "max_value": 10,
            "mostly": 0.95
        }
    },
    {
        "expectation_type": "expect_column_mean_to_be_between",
        "kwargs": {
            "column": "ge_signal_1",
            "min_value": 4,
            "max_value": 6
        }
    },
```

```
    {
        "expectation_type": "expect_column_values_to_be_unique",
        "meta": {
            "status_signal": "ge_signal_2_values_between_status"
        },
        "kwargs": {
            "column": "ge_signal_2",
            "mostly": 0.05
        }
    }
]
```

1. Если вы не можете разобраться с этой конфигурацией, пожалуйста, прочитайте раздел данного руководства Формирование ожиданий для получения дополнительной информации.
2. Обратите внимание, что некоторые ожидания имеют свойства `meta`, задающие сигналы состояния для отдельных ожиданий и пользовательских сообщений отчета. Более подробно эти свойства рассматриваются в разделе руководства Настройка отчета. Пока же важно знать, что в дополнение к сигналу состояния для всего отчета добавляются два сигнала состояния: `ge_signal_1_values_between_status` и `ge_signal_2_values_between_status`, которые соответствуют первому и последнему ожиданиям, соответственно.
3. Здесь мы ожидаем, что 95% значений первого сигнала будет находиться в диапазоне *[0; 10]*, а его среднее значение - в диапазоне *[4; 6]* (заметим, что для одного и того же сигнала можно оценить несколько ожиданий). Мы также ожидаем, что второй сигнал будет иметь не менее 5% уникальных значений. Как вы видели выше, все эти ожидания верны.
4. Если вы хотите задать ожидания в качестве дефолтного значения входного параметра, вам нужно экранировать двойные кавычки внутри выражения. Для примера выше, вам необходимо записать следующее:

```
{
  "fieldType": "LONG_TEXT",
  "name": "expectations",
  "label": "expectations",
  "isRequired": true,
  "description": "JSON-parsable list of expectations",
  "defaultValue": "[{\"meta\": {\"status_signal\": \"ge_signal_1_values_between_status\", \"report_rules\'
}
```

   Проще всего данное выражение сгенерировать с помощью кода на python:

```
import json

my_config = [{"expectation_type": "expect_column_values_to_be_unique",
              "meta": {
                  "status_signal": "ge_signal_2_values_between_status"
              },
              "kwargs": {
                  "column": "ge_signal_2",
                  "mostly": 0.05
              }}]

print(json.dumps(my_config).replace('"', '\\"'))
```

   Код представленный выше выведет ожидание с экранированными двойными кавычками.

5. Внутри каждого ожидания вам нужно описать несколько параметров:

- `meta` - здесь вы можете задать все параметры для уведомлений: о каких статусах следует уведомлять, каков их уровень важности, какие сообщения должны быть отправлены в уведомления. Больше информации в Настройке отчетов.
- `expectation_type` - здесь вам нужно задать название ожидания.
- `kwargs` - здесь вам нужно задать все параметры специфичные для конкретного ожидания. Параметры зависят от конкретного ожидания. Полный список параметров можно найти в описании ожидания. Часто, в качестве параметров передается `column`, содержащий название сигнала, `min_value` и `max_value`, содержащие границы значений.

6. Внутри параметра `meta` вы можете указать `status_signal`. Это предоставляет способ задать индивидуальный статус сигнал для выбранного ожидания. Заметрим, что указывать параметр `status_signal` внутри `meta` не обязятельно. Главное отличие от общего параметра `status_signal`, который задается во входных параметрах приложения состоит в том, что `status_signal` из `meta` отвечает только за числовой статус конкретного ожидания. В то время как общий в `status_signal` записывается самый опасный статус среди всех ожиданий.

Интеграция с уведомлениями

Как уже упоминалось, данное приложение может отправлять уведомления когда статус ожиданий изменяется. Вы можете гибко настроить какие именно уведомления вы хотите получать.

1. Для конфигурации уведомлений, вам нужно указать параметр `notification_groups` во входных параметрах приложения. После этого приложение будет отправлять уведомления во все сконфигурированные шаблоны уведомлений для выбранной группы нотификаций.
2. Уведомления отправляются только для тех статусов, которые указаны в поле `meta`. Например, для ожидания заданного ниже:

```
{
    "meta": {
        "report_rules": {
            "1": {"text": "my_signal: all values are in expected range", "type": "status"},
            "0": {"text": "my_signal: value range violation", "type": "warning"},
            "-1": {"text": "my_signal: signal repeatedly violates the range","type": "error"},
            "-2": {"text": "Application crashed or terminated", "type": "status"}
        },
        "additional_keys": {"key1": "val1", "key2": "val2"},
        "duplicate_message_timeout": "10min"
    },
    "expectation_type": "expect_column_values_to_be_between",
    "kwargs": {
        "column": "my_signal",
        "min_value": 0,
        "max_value": 10
    }
}
```

   уведомление будет отправлено если статус ожидания поменяется на 1, 0, -1 or -2.

   Для ожидания ниже:

```
{
    "meta": {
        "report_rules": {
```

```
            "-1": {"text": "my_signal: signal repeatedly violates the range","type": "error"},
        }
    },
    "expectation_type": "expect_column_values_to_be_between",
    "kwargs": {
        "column": "my_signal",
        "min_value": 0,
        "max_value": 10
    }
}
```

приложение отошлет уведомление только в случае ошибки (статус -1).

3. Уведомомления отправляются каждый раз когда статус *изменяется*, другими словами, если ожидание получит несколько статусов с ошибкой подряд, вы получите уведомление лишь на первый раз.
4. Текст уведомления будет взят из поля `text`, либо сгенерирован автоматически в случае если это поле не указано.

Уведомомления в случае сбоя приложения

Если вы хотите, чтобы приложение отправляло уведомление в случае сбоя, вы можете добавить параметр. Дополнительную информацию можно найти в разделе Уведомление о сбое приложения.

Приостановить работу приложения в зависимости от значений в специальном сигнале

Если вы хотите, чтобы приложение приостанавливало свои проверка на основании значений специального сигнала, вы можете добавить соответствующий параметр. Дополнительную информацию можно найти в разделе Приостановка уведомлений по значению в сигнале.

Передать дополнительные пары ключ-значение в текст уведомления

Если вы хотите, чтобы приложение отправляло дополнительные пары ключ-значение в текст уведомления, вы можете добавить параметр `additional_keys`. Дополнительную информацию можно найти в разделе Отправка дополнительных пар ключ-значение.

Отсылать одинаковые сообщения после истечения таймаута

По умолчанию приложение отправляет только уникальные уведомления. Однако вы можете использовать параметр `duplicate_message_timeout`, чтобы указать таймаут для идентичных сообщений. Учтите, что вместо задания индивидуальных таймаутов с помощью `duplicate_message_timeouts` вы можете указать их в `meta` параметре. Больше информации в Документации приложения с нотификациями.

Формирование ожиданий

В данном разделе описаны основы работы с библиотекой Great Expectations и создания конфигураций ожиданий для ваших данных. Для более детального ознакомления просим пользователей обратиться к официальной документации. Там же вы найдете подробное описание всех доступных ожиданий и соответствующих параметров.

Этот раздел представляет собой быстрый практический курс по изучению функциональности Great Expectations. Для его выполнения вам потребуется локально установленные библиотеки Pandas и Great Expectations.

В комплект поставки Great Expectations входит более сотни готовых к использованию ожиданий и еще больше ожиданий сообщества. Выполните следующий фрагмент, чтобы получить список всех доступных ожиданий:

```
import great_expectations as ge
import pandas as pd

ge.from_pandas(pd.DataFrame()).list_available_expectation_types()
```

Теперь создадим пример данных. Обратите внимание, что для оценки ожиданий необходимо также преобразовать Pandas DataFrame в Great Expectations DataFrame:

```
import great_expectations as ge
import pandas as pd

df = pd.DataFrame(
    data={
        "sensor": [10, 22, 56, 34, 91],
        "heartbeat": [1, 2, 3, 4, 5],
    },
    index=pd.date_range(
        start="2022-01-01",
        periods=5,
        freq="1s",
    )
)
ge_df = ge.from_pandas(df)

print(df)
```

Выполнение приведенного выше кода дает следующий результат:

```
                     sensor  heartbeat
2022-01-01 00:00:00      10          1
2022-01-01 00:00:01      22          2
2022-01-01 00:00:02      56          3
2022-01-01 00:00:03      34          4
2022-01-01 00:00:04      91          5
```

Все конфигурации ожиданий представляют собой словари с двумя ключами: `expectation_type` и `kwargs`. Если `expectation_type` - это просто уникальный идентификатор ожидания, то `kwargs` - это словарь с одним обязательным свойством `column`, указывающим столбец, для которого оценивается ожидание, и несколькими аргументами, характерными для данного типа ожидания.

Представьте, что мы ожидаем, что все значения `сенсора` будут находиться в диапазоне [0; 100]. Для проверки этого необходимо ожидание `expect_column_values_to_be_between`. Конфигурация должна выглядеть следующим образом:

```
expectation = {
    "expectation_type": "expect_column_values_to_be_between",
    "kwargs": {
        "column": "sensor",
        "min_value": 0,
```

```
        "max_value": 100,
    }
}
```

Обратите внимание на два специфических аргумента `min_value` и `max_value`, определяющих границы диапазона. Теперь подтвердим это ожидание, вызвав метод `validate` DataFrame:

```
ge_df.validate(
    expectation_suite={
        "expectation_suite_name": "test",
        "expectations": [expectation],
    },
)
```

> ⓘ Примечание
>
> Параметр `expectation_suite` может быть задан произвольной строкой. В приложении этот параметр не используется.

Выполнение приведенного выше кода дает следующий результат:

```
{
    "success": true,
    "meta": {
        ...
    }
}
```

Обратите внимание, что флаг верхнего уровня `success` равен `true`, что означает, что все ожидания оправдались. Информация, представленная внутри `meta`, дает более подробное представление о том, какие ожидания только что были оценены, какие из них оказались успешными и неудачными, и почему.

Аналогично, в одном вызове `validate` можно оценить несколько ожиданий. Например, проверим, что значения `heartbeat` уникальны и имеют возрастающий порядок. Для этого потребуются ожидания `expect_column_values_to_be_unique` и `expect_column_values_to_be_increasing`, соответственно:

```
expectations = [
    {
        "expectation_type": "expect_column_values_to_be_unique",
        "kwargs": {
            "column": "heartbeat"
        }
    },
    {
        "expectation_type": "expect_column_values_to_be_increasing",
        "kwargs": {
            "column": "heartbeat"
        }
    },
]
ge_df.validate(
    expectation_suite={
        "expectation_suite_name": "test",
        "expectations": expectations,
    },
)
```

Обратите внимание, что в отличие от предыдущего типа ожидания, эти типы ожидания не требуют дополнительных аргументов, кроме имени столбца. Убедитесь, что флаг верхнего уровня `success` равен `true`, и при необходимости проверьте метаданные. Теперь вы можете экспериментировать с различными типами ожиданий и конфигурациями. Если остались какие-то непонятные моменты, обратитесь к официальной документации.

Единственное, что необходимо сделать перед запуском приложения, - это убедиться, что все свойства `column`, которые вы определили для всех своих ожиданий, являются Public ID существующих сигналов. Теперь можно следовать инструкциям Создать и запустить приложение и использовать свои конфигурации.

Для настройки отчетов приложения можно также добавить к ожиданиям некоторые метаданные, что более подробно описано в следующем разделе.

Настройка отчетов

В этом разделе описано, как настроить сообщения отчета и добавить сигналы состояния для отдельных ожиданий. Обе эти опции можно контролировать с помощью свойств в метаданных ожиданий. Все свойства являются необязательными.

Если добавить свойство `status_signal` в мету ожидания, то приложение создаст новый сигнал ROW и будет записывать в него обновления состояния. Например:

```
{
    "meta": {
        "status_signal": "status_for_my_signal"
    },
    "expectation_type": "expect_column_values_to_be_between",
    "kwargs": {
        "column": "my_signal",
        "min_value": 0,
        "max_value": 10
    }
}
```

Ниже приведен список возможных значений статуса ожидания и их значения:

`1`	Нормально. Ожидание успешно.
`0`	Предупреждение. Ожидание провалено один или несколько раз, но менее `max_patience` раз подряд.
`-1`	Ошибка. Ожидание не выполняется более `max_patience` раз подряд.
`-2`	Отключено. Приложение либо только что запущено, либо остановлено, либо не получало никаких данных.

Вы также можете предоставить пользовательские сообщения для каждого из кодов состояния в списке. Для этого добавьте свойство `report_rules` в мета-объект ожидания:

```
{
    "meta": {
        "report_rules": {
            "1": {"text": "my_signal: all values are in expected range", "type": "status"},
```

```
            "0": {"text": "my_signal: value range violation", "type": "warning"},
            "-1": {"text": "my_signal: signal repeatedly violates the range","type": "error"},
            "-2": {"text": "Application crashed or terminated", "type": "status"}
        }
    },
    "expectation_type": "expect_column_values_to_be_between",
    "kwargs": {
        "column": "my_signal",
        "min_value": 0,
        "max_value": 10
    }
}
```

Убедитесь, что ключи `report_rules` являются строками, а значения - либо строками, либо словарями в формате виджета журнала. Вы можете опустить некоторые ключи, если вам не нужны сообщения для определенных значений статуса (например, для случаев успеха). Если вам не нужно свойство status signal, вы можете пропустить его, и сообщения отчета все равно будут доступны.

Пользовательские ожидания SDK

В дополнение к основному набору ожиданий, предлагаемому библиотекой great_expectations, библиотека platform SDK также предлагает ряд уникальных пользовательских ожиданий. Ниже приведен полный список этих пользовательских ожиданий с краткими описаниями.

expect_column_last_value_to_be_between

Это ожидание проверяет, находится ли последнее не-NaN-значение столбца между `min_value` и `max_value`. Параметры:

- `column` - имя столбца для применения ожиданий
- `min_value` - минимальное значение для сравнения. Если не задано, то сравнение будет производиться только с `max_value`.
- `max_value` - максимальное значение для сравнения. Если не задано, то сравнение будет производиться только с `min_value`.
- `strict_min` - выполнять ли строгое сравнение с `min_value`, по умолчанию `False`.
- `strict_max` - выполнять ли строгое сравнение с `max_value`, по умолчанию `False`.

Примеры

1. Если `strict_min` и `strict_max` не указаны, то условием будет `min_value <= val <= max_value`:

   DataFrame

   ```
                       heartbeat
   2022-01-01 00:00:00          0
   2022-01-01 00:00:01          1
   2022-01-01 00:00:02          0
   2022-01-01 00:00:03          1
   2022-01-01 00:00:04          0
   ```

   Expectation

```
{'expectation_type': 'expect_column_last_value_to_be_between',
    'kwargs': {
        'column': 'heartbeat',
        'min_value': 0,
        'max_value': 0
    }
}
```

Output

```
{"success": True, "result": {"observed_value": 0}}
```

2. При желании можно указать только один из порогов:

DataFrame

```
                    heartbeat
2022-01-01 00:00:00      0
2022-01-01 00:00:01      1
2022-01-01 00:00:02      0
2022-01-01 00:00:03      1
2022-01-01 00:00:04      0
```

Expectation

```
{'expectation_type': 'expect_column_last_value_to_be_between',
    'kwargs': {
        'column': 'heartbeat',
        'min_value': 0,
        'strict_min': True
    }
}
```

Output

```
{"success": False, "result": {"observed_value": 0}}
```

3. Если последнее значение `NaN`, то ожидание использует последнее значение, отличное от `NaN`.

DataFrame

```
                    heartbeat
2022-01-01 00:00:00      0
2022-01-01 00:00:01      1
2022-01-01 00:00:02      0
2022-01-01 00:00:03      1
2022-01-01 00:00:04    NaN
```

Expectation

```
{'expectation_type': 'expect_column_last_value_to_be_between',
    'kwargs': {
        'column': 'heartbeat',
        'min_value': 0,
        'max_value': 1
    }
}
```

Output

```
{"success": True, "result": {"observed_value": 1}}
```

Примеры ожиданий

В данном разделе описываются примеры некоторых ожиданий, которые могут быть полезны для конфигурации приложения без обращения к официальной документации. Все параметры внутри `<>` должны быть заполнены пользователем перед запуском приложения:

- значения сигнала лежат в определенном промежутке (0 < value < 100)

```
{
    'expectation_type': 'expect_column_values_to_be_between',
    'kwargs': {
        'column': '<signal_public_id>',
        'min_value': 0,
        'max_value': 100,
        'strict_min': true,
        'strict_max': true
    }
}
```

- значения сигнала не равны определенным значениям (1 или 2)

```
{
    'expectation_type': 'expect_column_values_to_not_be_in_set',
    'kwargs': {
        'column': '<signal_public_id>',
        'value_set': [1, 2]
    }
}
```

- значения сигнала равны определенным значениям (1 или 2)

```
{
    'expectation_type': 'expect_column_values_to_be_in_set',
    'kwargs': {
        'column': '<signal_public_id>',
        'value_set': [1, 2]
    }
}
```

- значения сигнала принадлежат определенному типу (целое число)

```
{
    'expectation_type': 'expect_column_values_to_be_of_type',
    'kwargs': {
        'column': '<signal_public_id>',
        'type_': 'int'
    }
}
```

- значения сигнала возрастают

```
{
    'expectation_type': 'expect_column_values_to_be_increasing',
    'kwargs': {
        'column': '<signal_public_id>'
    }
}
```

- значения сигнала убывают

```
{
    'expectation_type': 'expect_column_values_to_be_decreasing',
    'kwargs': {
        'column': '<signal_public_id>'
    }
}
```

Больше примеров ожиданий и информации о формате и конфигурации можно найти в Галерее ожиданий.

Смотрите также

- Приложение с уведомлениями
- PandasEvaluator

Data periodicity

В этом руководстве вы изучаете основы приложения периодичности данных, а также как запустить это приложение из пользовательского интерфейса платформы.

Предварительные условия

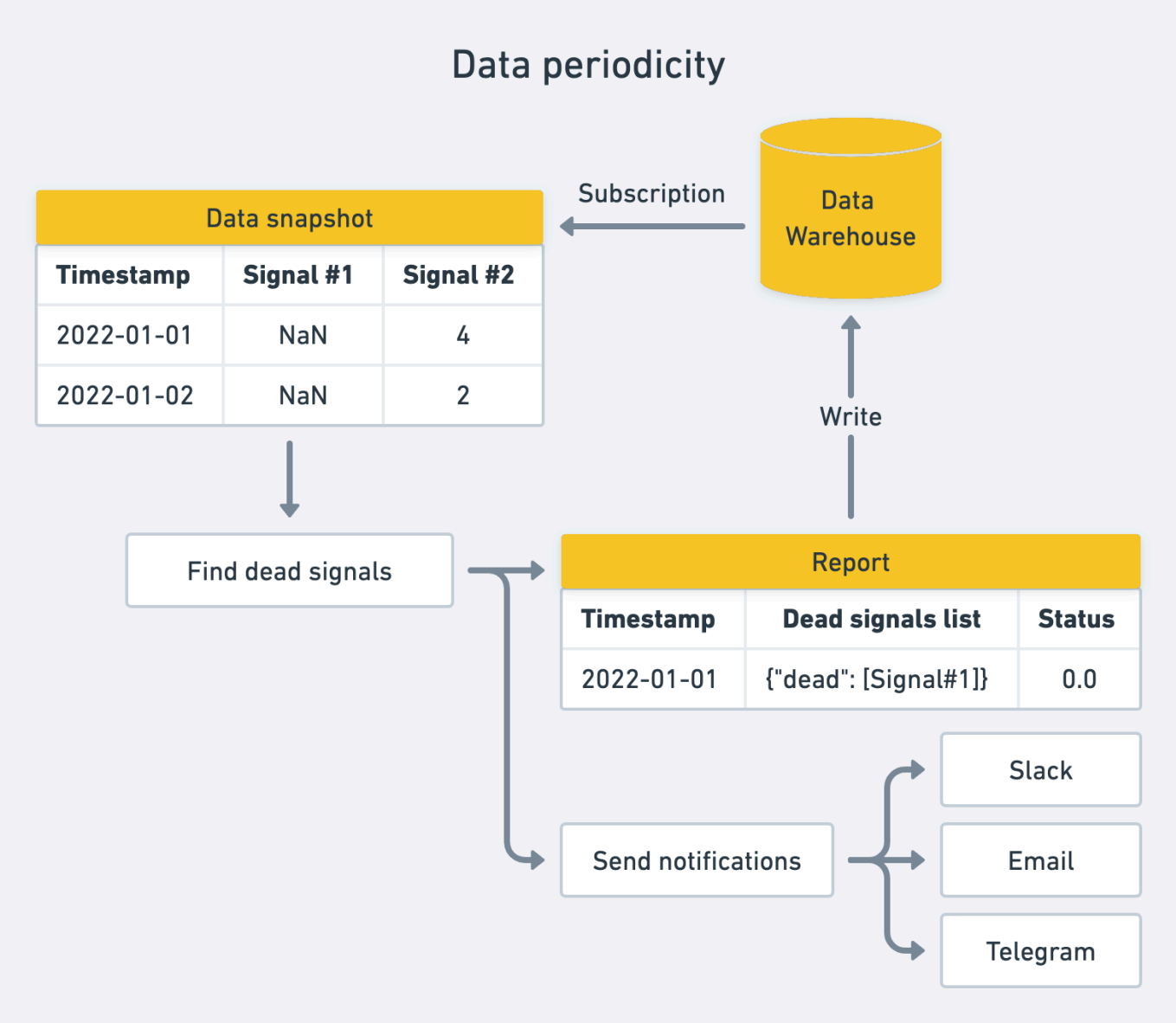
- Аккаунт на платформе Тайга.
- Знакомство с понятиями приложений, сигналов и уведомлений. Прочитайте соответствующие разделы документации платформы. Чтобы получить доступ к документации платформы, войдите в платформу, нажмите на значок пользователя в левом нижнем углу страницы и выберите «Документация платформы».
- Настроенный реестр Docker с образом `platform-sdk:latest`. Если вы не знакомы с реестрами Docker, мы рекомендуем начать с этого руководства.

Обзор

Периодичность данных-это приложение для последовательно мониторинга данных в режиме реального времени. Если новые данные не зарегистрированы на дольше, чем указанный период, приложение отправит предупреждения по электронной почте, телеграмме или Slack, или записать флаги ошибок на указанные сигналы.

Это может пригодиться, особенно когда какой-либо источник данных (например, датчик) используется в качестве входного сигнала для приложения, работающего на платформе, когда любые существенные задержки в потоке входных данных могут привести к сбою приложения.

Диаграмма приложения показана ниже:



Для работы периодичность данных требует список сигналов для мониторинга и желаемых частот, которые могут варьироваться для различных сигналов. Приложение контролирует эти сигналы и регулярно предоставляет обновления о статусе их периодичности. Вывод этого приложения включает в себя сигнал состояния, который представляет общий статус всех сигналов (или отдельные сигналы состояния для каждого контролируемого сигнала, в зависимости от конфигурации). Кроме того, генерируется сигнал отчета, содержащий подробную информацию о любых сигналах поступивших с задержкой. Пользователь также может настроить группу уведомлений для целей оповещения.

Приложение может создать следующие статусы на основе ситуации:

1 – NORMAL: Это состояние указывает на то, что все работает правильно и сигналы принимаются так часто, как необходимо, или даже чаще. 0 – WARNING: Это состояние означает, что один или несколько сигналов испытывают задержки. -2 — DEACTIVATED: этот статус указывает, что приложение либо только что запустилось, либо остановлено, либо еще не получило никаких данных.

Требования к ресурсам

Для периодичности данных требуется всего 90 МБ ОЗУ для мониторинга до 100 сигналов. Рекомендуется поставить лимит в 110 МБ, чтобы зарезервировать 20 процентов ОЗУ. Если

вам необходимо отслеживать более 100 сигналов одновременно, просто увеличьте вдвое объем необходимой оперативной памяти или запустите второй экземпляр приложения.

Запустите приложение из пользовательского интерфейса платформы.

Ниже приведен пример настройки и запуска приложения:

1. Создать сигнал: создаёт пустой сигнал наблюдения для повторной проверки на наличие новых значений.
2. Создать группу уведомлений: создать и настроить группу уведомлений для получения оповещений из приложения.
3. Создать и запустить приложение: создайте шаблон периодичности данных и запустите приложение.
4. Проверьте результаты: убедитесь, что приложение дождалось добавления новых значений к сигналу наблюдения, записало статус оповещения `0.0` в указанный сигнал статуса и отправило вам оповещение по электронной почте.

Создать сигнал

Во-первых, нам нужно создать пустой сигнал наблюдения.

1. Войдите на платформу и перейдите в раздел **Заводские настройки** → **Сигналы**.
2. Нажмите кнопку **Создать сигнал** в правом верхнем углу окна.
3. В открытии диалога введите сигнал публичный идентификатор `data_periodicity_observe` и установите селектор **Тип данных** на **ROW (число)**. Оставьте все другие поля по умолчанию и нажмите **Создать**.

Создать группу уведомлений

Вы можете создать группу уведомлений либо на Платформе, либо из кода. Если вы решили пойти первым путем, обратитесь к документации Платформы. В противном случае обратитесь к Руководству по уведомлениям.

Создать и запустить приложение

Следующим шагом является создание и запуск приложения.

> ⓘ Примечание
>
> Подробные инструкции по регистрации и запуску приложений на платформе приведены в документации платформы. Чтобы открыть документацию платформы, войдите в платформу, нажмите значок пользователя в левом нижнем углу страницы и выберите **Документация по платформе**.

1. Создайте новый шаблон приложения, дайте ему осмысленное имя и запомните его для дальнейшего использования.

2. Создайте новый шаг с произвольным именем. Оставьте значение по умолчанию в поле **Страница** и укажите следующие два параметра. Нажмите кнопку **Сохранить** под каждым параметром.

`Job params`:

```
{
    "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
    "MAIN_HANDLER": "data_periodicity",
    "RESOURCE_TYPE": "FLOW_JOB",
    "RAM_REQUEST_MB": 110,
    "RAM_LIMIT_MB": 110
}
```

`Input params`:

```
{
  "fields": [
    {
      "fieldType": "LONG_TEXT",
      "name": "track_signals",
      "label": "track_signals",
      "isRequired": false,
      "description": "Comma-separated list of signal public ids to be tracked, e.g. 'track_1,track_2'"
    },
    {
      "fieldType": "TEXT",
      "name": "track_signals_group",
      "label": "track_signals_group",
      "isRequired": false,
      "description": "Group of track signals. Could be set instead of track_signals field. Cannot be u
    },
    {
      "fieldType": "TEXT",
      "name": "periodicity",
      "label": "periodicity",
      "isRequired": true,
      "defaultValue": "10 sec",
      "description": "Either a single pandas.Timedelta-parsable desired data periodicity value (e.g. '
    },
    {
      "fieldType": "TEXT",
      "name": "status_signals",
      "label": "status_signals",
      "isRequired": true,
      "description": "Either a single ROW signal public id (e.g. 'status_1') or comma-separated list o
    },
    {
      "fieldType": "TEXT",
      "name": "report_signal",
      "label": "report_signal",
      "isRequired": true,
      "description": "Public id of BLOB signal to receive dead signals"
    },
    {
      "fieldType": "TEXT",
      "name": "notification_group_name",
      "label": "notification_group_name",
      "isRequired": false,
      "description": "Existing notification group id, e.g. 'my-group'"
    },
     {
      "fieldType": "TEXT",
      "name": "heartbeat_internal_signal",
      "label": "heartbeat_internal_signal",
      "isRequired": false,
      "description": "Signal for writing internal smart heartbeat of app"
    },
    {
      "fieldType": "TEXT",
```

```
      "name": "heartbeat_startup_silence_period",
      "label": "heartbeat_startup_silence_period",
      "isRequired": false,
      "description": "How many iterations to wait until starting to send heartbeat"
    },
     {
      "fieldType": "TEXT",
      "name": "batch_interval",
      "label": "batch_interval",
      "isRequired": false,
      "description": "Timedelta interval between the iterations in pandas.Timedelta format",
      "defaultValue": "1 s"
    }
  ]
}
```

`LOGGER_LEVEL`

type: string

Дополнительный уровень журнала приложений. Дополнительную информацию см. в руководстве Логирование.

`track_signals`

type: string optional

Разделенный запятыми список пользовательских идентификаторов сигналов, которые необходимо отслеживать на наличие обновлений.

`track_signals_group`

type: string optional

Группа сигналов трека. Может быть установлен вместо поля Track_signals. Нельзя использовать с несколькими состояниями.

`periodicity`

type: string

Либо единственное значение периодичности данных для `pandas.timedelta` (например, `12 H`, `5 min`, `3 seconds`), либо список значений, разделенных запятой.

`status_signals`

type: string

Либо один пользовательский идентификатор, либо разделенный запятыми список пользовательских идентификаторов сигналов состояния для получения статуса обновления track_signals: `0.0`, если хотя бы один не обновлен, и `1.0` в противном случае. Несколько пользовательских идентификаторов нельзя использовать с параметром track_signals_group.

`report_signal`

type: string

	Публичный идентификатор `blob` - сигнала для получения списка неактивных сигналов в `track_signals` (если есть).
`notification_group_name`	`type: string` `optional` Запятые имена существующих групп уведомлений, например, `my-group,my-group2`.
`heartbeat_internal_signal`	`type: string` `optional` Дополнительный сигнал для внутреннего умного хартбита приложения. Если указано, приложение будет записывать хартбит в конце каждой итерации цикла.
``batch_interval ``	`type: string` `optional` Время между проверками на поступление новых данных в формате pandas.Timedelta

> **Предупреждение**
>
> Обратите внимание, что параметры `status_signals` и `periodicity` оба должны иметь либо 1 значение (один сигнал состояния и периодичность для всех отслеживаемых сигналов), либо одинаковое количество записей, что и `track_signals` (разделенный запятыми). Последний вариант может использоваться для задания нескольких различных сигналов или даже групп сигналов для отслеживания.
>
> Например, эта комбинация сигналов для отслеживания - `["track_1", "track_2", "track_3"]` - и status_signals - `["status_1", "status_1", "status_2"]` - валидны. При такой настройке сигнал состояния `status_1` будет получать статус как `track_1`, так и `track_2`, тогда как `status_2` будет получать только статус `track_3`.

> ⓘ Примечание
>
> Во избежание спама функция периодичности данных отправляет оповещения только при изменении состояния потока данных, например:
>
> - *Dead signals: [„a“, „b“, „c“]*
> - *No dead signals*
> - *Dead signals: [„a“]*
>
> и никогда не отправляет одни и те же сообщения подряд.
>
> Однако будьте осторожны, добавляя оповещения, когда значение «периодичности» слишком мало (например, менее 10 секунд), если вы точно не знаете ожидаемую периодичность данных, потому что это все равно может очень сильно спамить ваш почтовый ящик.

3. Создайте новое приложение из только что созданного шаблона и запустите его со следующими параметрами:

`Environment`	`REMOTE`
`LOGGER_LEVEL`	`INFO`
`track_signals`	`data_periodicity_observe`
`periodicity`	`10 s`
`status_signals`	`data_periodicity_alert`
`report_signal`	`data_periodicity_dead_signals`
`notification_group_name`	Имя созданной группы уведомлений выше.

4. Подождите, пока приложение не изменит свой статус на «ВЫПОЛНЯЕТСЯ», затем перейдите на страницу выполнения шага и обновляйте ее, пока не увидите эти строки:

```
... ...
... INFO     Wrote status code '0.0' to signal 'data_periodicity_alert'
... INFO     Wrote dead signals list ['data_periodicity_observe'] to signal 'data_periodicity_dead_signals
```

   После этого вы можете безопасно завершить этот шаг.

Проверьте результаты

Последний шаг - убедиться, что приложение завершилось, как и ожидалось. Для этого:

- Откройте свой адрес электронной почты и убедитесь, что вы получили новое электронное письмо со следующим текстом: «Dead signals: [„data_periodicity_observe“]».
- Перейти к **Настройки фабрики** → **Сигналы** и поиск сигнала `data_periodicity_alert`. Открыть страницу сигнала и убедитесь, что у него есть как

минимум одно значение, равное `0.0` (статус оповещения).
- Точно так же найдите сигнал `data_periodicity_dead_signals`. Откройте страницу сигнала и убедитесь, что у него есть хотя бы одно значение, равное `{"Dead_signals": ['data_periodicity_observe']}` (потому что этот сигнал не имел обновлений).

Поздравляем! Вы только что освоили основы периодичности данных!

Уведомомления в случае сбоя приложения

Если вы хотите, чтобы приложение отправляло уведомление в случае сбоя, вы можете добавить параметр. Дополнительную информацию можно найти в разделе Уведомление о сбое приложения.

Приостановить работу приложения в зависимости от значений в специальном сигнале

Если вы хотите, чтобы приложение приостанавливало свои проверка на основании значений специального сигнала, вы можете добавить соответствующий параметр. Дополнительную информацию можно найти в разделе Приостановка уведомлений по значению в сигнале.

Передать дополнительные пары ключ-значение в текст уведомления

Если вы хотите, чтобы приложение отправляло дополнительные пары ключ-значение в текст уведомления, вы можете добавить параметр `additional_keys`. Дополнительную информацию можно найти в разделе Отправка дополнительных пар ключ-значение.

Отсылать одинаковые сообщения после истечения таймаута

По умолчанию приложение отправляет только уникальные уведомления. Однако вы можете использовать параметры `duplicate_message_timeouts` или `duplicate_message_timeout`, чтобы указать таймаут для идентичных сообщений. Больше информации в Документации приложения с нотификациями.

PandasEvaluator

В данном руководстве вы узнаете как работать с дефолтным приложением PandasEvaluator, а также как настроить и запустить это приложение из пользовательского интерфейса Платформы.

Необходимые условия

- Аккаунт на Платформе.
- Знакомство с понятиями приложений, сигналов и уведомлений. Прочитайте соответствующие разделы документации платформы. Чтобы получить доступ к документации платформы, войдите в платформу, нажмите на значок пользователя в левом нижнем углу страницы и выберите «Документация платформы».
- Настроенный реестр Docker с образом `platform-sdk:latest`. Если вы не знакомы с реестрами Docker, мы рекомендуем начать с этого руководства.

Обзор

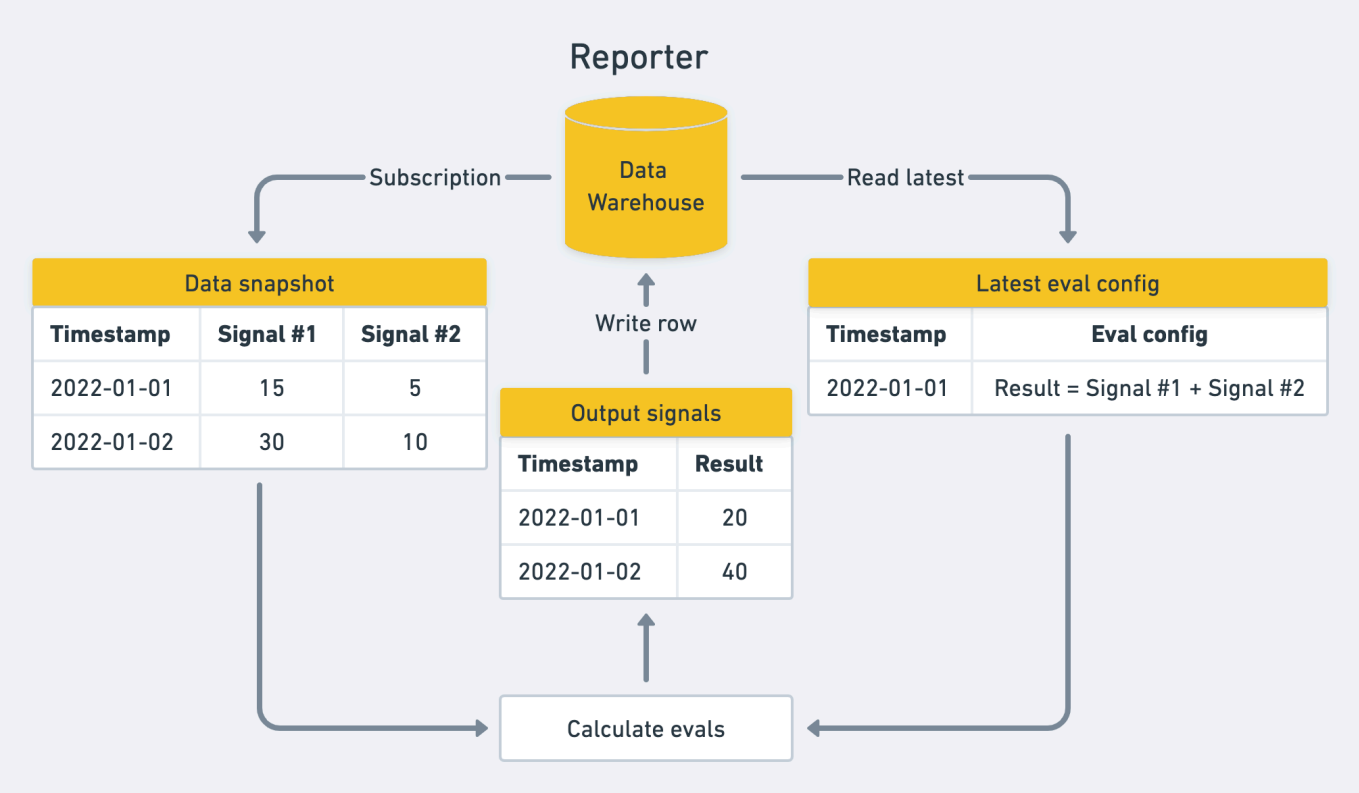
Приложение PandasEvaluator использует библиотеку Pandas для операций, связанных с обработкой и анализом данных. Главная цель такого приложения - облегчить работу пользователей с табличными данными.

PandasEvaluator может исполнять пользовательские формулы с использованием `pandas.eval()`. Приложение можно использовать для создания новых виртуальных датчиков и расчета пользовательских метрик.

Формулы расчета могут быть заданы один раз при запуске приложения (с использованием параметра `eval_json`) или динамически загружаться из BLOB-сигнала (указанного в параметре `eval_signal`). Можно использовать только `eval_json` либо `eval_signal` – использовать оба параметра невозможно.

Для настройки оповещений необходимо указать поле `alerts`. По крайней мере, один из трех ранее упомянутых параметров должен быть указан.

Ниже приведена диаграмма приложения:



Требования к ресурсам

PandasEvaluator требует только 100 МБ ОЗУ для чтения и записи до 100 сигналов. Рекомендуется установить лимит в 120 Мб, чтобы резервировать 20 процентов ОЗУ. Если вам необходимо работать с более чем 100 сигналами одновременно, просто удвойте требуемый объем оперативной памяти или запустите второй экземпляр приложения.

Настройка приложения на Платформе

Этот раздел описывает, как создать новый шаблон приложения и как настроить новое приложение для интеграции с PandasEvaluator. Подробные инструкции по регистрации

приложений приведены в документации Платформы.

Создание нового шаблона

Перед созданием нового приложения необходимо создать новый шаблон на Платформе. Зайдите в меню **Шаблоны приложения** и выполните следующие шаги:

1. Создайте новый шаблон приложения и запомните его для последующего использования.
2. Создайте новый шаг внутри вашего созданного шаблона
3. Укажите следующие *входные* параметры и *параметры процесса выполнения*

Параметры процесса выполнения

```
{
    "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
    "MAIN_HANDLER": "pandas_evaluator",
    "RESOURCE_TYPE": "FLOW_JOB",
    "RAM_REQUEST_MB": 120,
    "RAM_LIMIT_MB": 120
}
```

Входные параметры

```
{
  "fields": [
    {
      "fieldType": "LONG_TEXT",
      "name": "eval_json",
      "label": "eval_json",
      "isRequired": false,
      "description": "JSON-parsable evals description, e.g. '{\"sum_of_signals\": {\"eval\": f\"{col1}
    },
    {
      "fieldType": "TEXT",
      "name": "eval_signal",
      "label": "eval_signal",
      "isRequired": false,
      "defaultValue": null,
      "description": "Signal with eval jsons. Either this or eval_json should be set"
    },
    {
      "fieldType": "TEXT",
      "name": "input_signals",
      "label": "input_signals",
      "isRequired": true,
      "defaultValue": "",
      "description": "Input signals, separated by comma"
    },
    {
      "fieldType": "TEXT",
      "name": "granularity",
      "label": "granularity",
      "isRequired": true,
      "defaultValue": "None",
      "description": "pandas.Timedelta-parsable data step size. Use None for raw reading"
    },
    {
      "fieldType": "TEXT",
      "name": "batch_size",
      "label": "batch_size",
      "isRequired": true,
      "defaultValue": "1min",
      "description": "pandas.Timedelta-parsable data batch size"
    },
    {
      "fieldType": "TEXT",
```

```
      "name": "use_regular_reading",
      "label": "use_regular_reading",
      "isRequired": true,
      "defaultValue": "False",
      "description": "Whether to use regular reading or subscription"
    },
    {
      "fieldType": "TEXT",
      "name": "heartbeat_internal_signal",
      "label": "heartbeat_internal_signal",
      "isRequired": false,
      "description": "Signal for writing internal smart heartbeat of app"
    },
    {
      "fieldType": "TEXT",
      "name": "heartbeat_startup_silence_period",
      "label": "heartbeat_startup_silence_period",
      "isRequired": false,
      "description": "How many iterations to wait until starting to send heartbeat"
    }
  ]
}
```

При настройке параметров процесса выполнения учтите упомянутые выше требования к ресурсам. После указания каждого параметра нажмите кнопку Сохранить.

Создание приложения

После настройки шаблона необходимо создать новое приложение на Платформе. Используйте соответствующую кнопку внутри созданного шаблона или пройдите в **Приложения>Создать приложение** (убедитесь, что выбран ваш созданный шаблон).

После создание приложения, нажмите на кнопку *Запустить шаг*, которая находится в разделе *Шаги*. Настройте приложение согласно вашим требованиям:

`LOGGER_LEVEL`	`type: string` Уровень логгирования. Подробнее смотрите в Руководстве по логгированию.
`eval_json`	`type: string` `optional` Формулы расчета в формате JSON, например: `{"first_output_signal": {"eval": "`first_eval`"}}`. Вы можете использовать это поле, если хотите установить формулы один раз на весь период работы. Задайте значение либо для этого поля, либо для `eval_signal` – использовать оба параметра невозможно.
`alerts`	`type: string` `optional` Формулы оценки оповещений в формате JSON, например: `[{"condition": "a>10","notification_group":"<group>", "message":"condition triggered"}]`. Это поле определяет список оповещений, которые должны быть запущены на основе входных данных.

`eval_signal`	type: string optional Если вы хотите динамически изменять формулы во время выполнения приложения, вы можете указать сигнал BLOB. В этом случае приложение будет проверять обновление формул на каждой итерации. Задайте значение либо для этого поля, либо для `eval_json` – использовать оба параметра невозможно.
`input_signals`	type: string Входные сигналы, разделенные запятой.
`granularity`	type: string optional Размер шага данных в формате pandas.Timedelta. Используйте None для сырого чтения.
`batch_size`	type: string Размер пакета в формате pandas.Timedelta.
`use_regular_reading`	type: string optional Использовать ли обычное чтение или подписку.
`new_data_only`	type: string optional Если установлено значение `true`, оцениваются только новые данные и пропускаются значения с временными штампами из предыдущих итераций. В противном случае оцениваются все данные в снимке. Значение по умолчанию - *true*.
`heartbeat_internal_signal`	type: string Дополнительный сигнал для внутреннего умного хартбита приложения. Если указан, приложение будет записывать хартбит в конце каждой итерации цикла.

Формулы evaluation

Приложение ожидает формулы evaluation в следующем формате:

```
{
    "first_output_signal": {"eval": "`signal1` + `signal2`"},
    "second_output_signal": {"eval": "`signal3`.std()"}
}
```

> **Предупреждение**
>
> Необходимо обернуть бектиками (` ) указанное в eval значение. Обратите внимание, что необходимо обернуть только значение, а не все выражение целиком (напр., `"eval": "`<signal_name>`.std()"`).

Например:

```
{
"S11_q_ni-Value": {
    "eval": "(`S15-Value` - `S13-Value`) / (`K16-Value` - `K31-Value`) * 100"
},
 "feature-something-low": {
    "eval": "`feature-something-else`"
},
"my-virtual-sensor-min": {
    "eval": "`feature1-TechLo` + (`feature2-TechHi` - `feature1-TechLo`) * 0.05"
},
```

Начиная с SDK 2.7.0, вы можете объединять свои формулы, комбинируя последовательность выражений, разделенных точкой с запятой (;), чтобы выполнить сложные выражения, которые в противном случае вызвали бы ошибку. Например, вот как вы можете вычислить стандартное отклонение для суммы двух столбцов a и b: `c = (a + b); c.std().`

> **Примечание**
>
> Вы можете установить другой разделитель по своему выбору, изменив переменную среды `EVAL_DELIMITER` на требуемое значение.

С SDK 2.15.0 у вас есть возможность удалить плато (т.е. повторяющиеся значения, появляющиеся в строке) из данных перед выполнением формул. Плато может появиться в ваших данных по разным причинам, включая внутренний механизм заполнения вперед подписки, реализованный в SDK. Вы можете выбрать сигнал-ссылку для удаления плато, предоставив дополнительный аргумент ключевого слова `remove_plateau` в ваши формулы:

```
{
    "virtual-sensor-no-plateau-sum": {
        "eval": "`feature1-TechHi` + `feature1-TechLo`",
        "remove_plateau": "feature1-TechHi"
    },
    "virtual-sensor-no-plateau-mean": {
        "eval": "(`feature1-TechHi` + `feature1-TechLo`) / 2",
        "remove_plateau": "feature1-TechLo"
    },
}
```

> **Примечание**
>
> Важно включить сигнал, указанный в параметре `remove_plateau`, в параметр `input_signals`. Это гарантирует, что сигнал учитывается при обработке в приложении. Пожалуйста, дважды проверьте и убедитесь, что сигнал, указанный в параметре `remove_plateau`, также присутствует в параметре `input_signals`.

Обратите внимание, что вы не можете выбрать более одного сигнала удаления плато в одной формуле (запятые в значениях запрещены), но вы можете использовать разные сигналы удаления плато для разных формул. Чтобы лучше понять идею, рассмотрим пример с следующей конфигурацией формул:

```
{
    "C": {
        "eval": "(`A` + `B`)"
        "remove_plateau": "B"
    }
}
```

и данные:

```
                     A   B
2022-01-01 12:00:00  10  20
2022-01-01 13:00:00  11  21
2022-01-01 14:00:00  12  21
2022-01-01 15:00:00  13  23
```

В этом случае перед выполнением eval `(`A` + `B`)` снимок будет отфильтрован от плато по сигналу `B` (выбранному в качестве сигнала `remove_plateau`). Таким образом, результат будет следующим:

```
                     C
2022-01-01 12:00:00  30
2022-01-01 13:00:00  32
2022-01-01 15:00:00  36
```

Обратите внимание, что если вместо этого выбрать `A` в качестве сигнала `remove_plateau`, результат будет на одну строку длиннее, так как у `A` нет плато:

```
                     C
2022-01-01 12:00:00  30
2022-01-01 13:00:00  32
2022-01-01 14:00:00  34
2022-01-01 15:00:00  36
```

Оповещения

Параметр оповещений предоставляет возможность анализировать снимок входной подписки на основе условия `pd.eval()`. Основная структура входного параметра `alerts` выглядит следующим образом:

```
[
    {
        "condition": "a>10",
        "notification_group": "<group>",
        "message": "condition_triggered"
    }
]
```

Где:

- *condition`* - `pd.eval` – условие ДОЛЖНО возвращать логическое значение в качестве результата, поэтому оно должно быть настроено правильно. В противном случае будет

вызвано исключение ValueError.

- `notification_group` - уведомляет группу в случае, если условие имеет значение `true`.
- `message` - сообщение для оповещения. Обратите внимание, что сообщение может использовать шаблон Jinja, который может содержать входные параметры для замены. По умолчанию мы подставляем снимок ввода как параметр `df`. Если необходимо, чтобы никакой входной параметр не был настроен, тогда сообщение может представлять собой обычную строку (string).

Например, если пользователь хочет получать оповещение, когда кто-то написал сообщение в будущее, оповещения могут быть настроены следующим образом:

```
[
    {
        "condition": "index>datetime.datetime.now(datetime.timezone.utc)",
        "notification_group": "<group>",
        "message": "Someone wrote value to the future to one of signals {{df.columns}}  "
    }
]
```

Или следующим образом:

```
[
    {
        "condition": "index>datetime.datetime.now(datetime.timezone.utc)",
        "notification_group": "<group>",
        "message": "Someone wrote value to the future to one of signals: my_signal_1, my_signal_2"
    }
]
```

Расширенное форматирование сообщений

Если пользователю нужно, чтобы поле `message` содержало больше информации о состоянии оповещения, следует использовать параметр `message_params`. **message_params** - это словарь, который содержит имя переменной в качестве ключа и условие eval, которое предоставляет значение для `message` с использованием Jinja2. Если условие eval возвращает серию в качестве результата и оно должно быть использовано для фильтрации входа dataframe, то входной dataframe должен использоваться в качестве значения переменной для подстановки. Например, если пользователь хочет знать, какие значения были точно записаны в будущее в сообщении, оповещения могут выглядеть следующим образом:

```
[
    {
        "condition": "index>datetime.datetime.now(datetime.timezone.utc)",
        "notification_group": "<group>",
        "message": "Someone wrote value from future to one of signals: {{future_df.columns}}, values are {{fut
        "message_params": {"future_df": "index>datetime.datetime.now(datetime.timezone.utc)"}
    }
]
```

Чтобы предотвратить спам одного и того же сообщения, параметр `cooldown_sec` может использоваться для оповещения. Пример может выглядеть следующим образом:

```
[
    {
        "condition": "index>datetime.datetime.now(datetime.timezone.utc)",
        "notification_group": "<group>",
```

```
        "message": "Someone wrote value from future to one of signals: {{future_df.columns}}, values are {{fu
        "message_params": {"future_df": "index>datetime.datetime.now(datetime.timezone.utc)"},
        "cooldown_sec": 10
    }
]
```

В этом случае одно и то же сообщение не сможет быть отправлено в течение 10 секунд после отправки первого сообщения. Если `cooldown_sec` установлен в null, то приложение вообще не будет отправлять повторяющиеся сообщения. Вместо этого оно будет отправлять каждое конкретное сообщение в конкретную группу только один раз. Пожалуйста, используйте этот параметр осторожно, чтобы избежать ситуаций, когда пользователи пропускают важные уведомления.

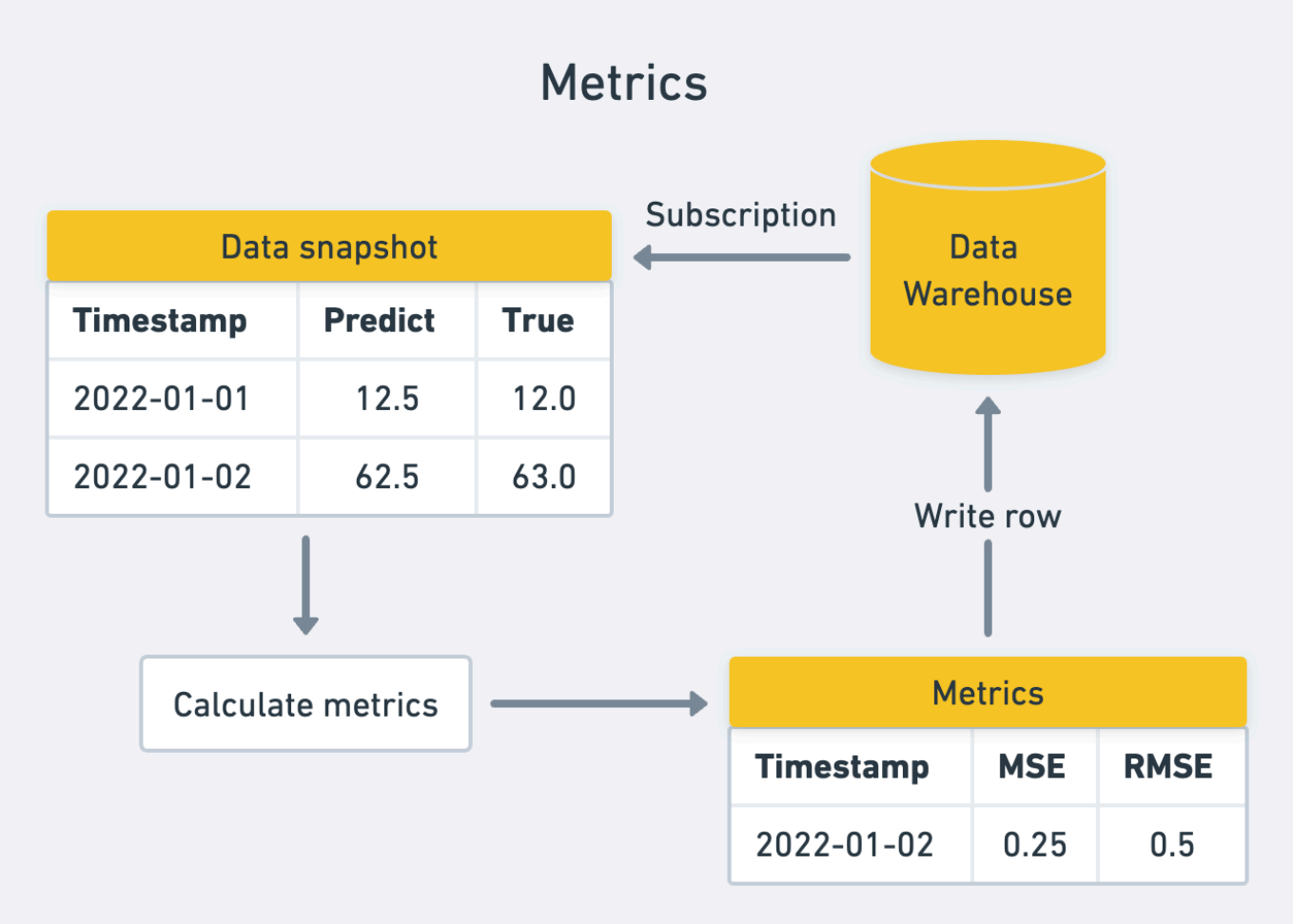
Metrics

В этом руководстве вы узнаете основы для работы с приложением Метрик, а также как правильно настроить данное приложение из пользовательского интерфейса платформы.

Обзор

Метрики - это приложение для оценки моделей машинного обучения непосредственно на платформе. Метрики работают с парами сигналов предсказания модели и истинными значениями в одном цикле для исторических данных или в цикле для стриминговых данных в режиме реального времени. Имеются как метрики классификации, так и метрики регрессии, доступные из коробки.

Диаграмма приложения показана ниже:



Для использования приложения метрик вам необходимо предоставить словарь конфигурации, который описывает желаемые метрики. Каждое описание метрики должно включать сигнал наблюдения, сигнал предсказания, название метрики и любые дополнительные аргументы. Вот пример конфигурации:

```
{
  "metrics": [
    {
      "prediction_signal": "sample_y_pred",
      "observation_signal": "sample_y_true",
      "metric_name": "MAE",
      "aggregation_window": "1 H"
    },
    {
      "prediction_signal": "sample_y_pred",
      "observation_signal": "sample_y_true",
      "metric_name": "MAPE",
      "aggregation_window": "1 H",
      "prediction_future_shift": "30min",
      "kwargs": {
        "multioutput": "uniform_average"
      }
    }
  ]
}
```

В настоящее время приложение метрик поддерживает следующие метрики, все из которых принимают соответствующие дополнительные аргументы на основе документации scikit-learn:

1. «MAE» - mean_absolute_error
2. «MAPE» - mean_absolute_percentage_error
3. «RMSE» - mean_absolute_error(squared=False)
4. «R2» - r2_score
5. «recall» - recall_score
6. «precision» - precision_score
7. «F1» - f1_score
8. «acceptance_rate» - acceptance_rate_score (y_true: pd.Series, y_pred: pd.Series, eps: float = 0), где `eps` представляет собой допустимый уровень приемлемости.

Убрать плато

С версии SDK 2.17.0 у вас есть возможность убирать плато (т.е. повторяющиеся значения, появляющиеся подряд) из данных перед расчетом метрики. Плато может появиться в ваших данных по разным причинам, включая внутренний механизм заполнения вперед подписки, реализованный в SDK. Вы можете выбрать опорный сигнал для удаления плато, предоставив дополнительный ключевой аргумент `remove_plateau` в описании вашей метрики:

```
{
  "metrics": [
    {
      "prediction_signal": "sample_y_pred",
      "observation_signal": "sample_y_true",
      "metric_name": "MAPE",
      "aggregation_window": "1 H",
      "remove_plateau": "signal_for_removing_plateau",
      "kwargs": {
        "multioutput": "uniform_average"
```

```
      }
    }
  ]
}
```

Дополнительные параметры метрик

Вы можете передавать дополнительные параметры каждой указанной метрике в виде пар ключ-значение в разделе `kwargs`. Эти параметры могут быть двух типов: конкретные параметры метрики, которые принимает scikit-learn (например, вы можете передать `{"average": "macro"}` для метрики `precision`), или один из дополнительных универсальных параметров ниже:

- `ignore_nan: bool`: если `True`, исключить все пары, где хотя бы одно из значений `y_true`, `y_pred` равно `NaN`. По умолчанию False.
- `y_true_fill: Literal["ffill", "bfill"]`: если указано, заполнить значения `NaN` в `y_true` с использованием стратегии forward-fill (`ffill`) или backward-fill (`bfill`). Если None, заполнение не применяется. По умолчанию None.
- `y_pred_fill: Literal["ffill", "bfill"]`: если указано, заполнить значения `NaN` в `y_pred` с использованием стратегии forward-fill (`ffill`) или backward-fill (`bfill`). Если None, заполнение не применяется. По умолчанию None.

Метрики классификации (recall, precision и F1) также принимают дополнительный аргумент

- `ignore_class`: указывает номер класса, который должен быть проигнорирован при вычислении метрики. По умолчанию None.

Предварительные требования

- Учетная запись на платформе.
- Знакомство с концепциями приложений, сигналов и артефактов. Прочитайте соответствующие разделы документации платформы. Для доступа к документации платформы войдите в систему, нажмите на значок пользователя в левом нижнем углу страницы и выберите «Документация по платформе».
- Настроенный реестр Docker с доступным образом `platform-sdk:latest`. Если вы не знакомы с реестрами Docker, прочтите, например, это руководство.

Требования к ресурсам

Для мониторинга до 100 сигналов метрики требуют всего 192 МБ оперативной памяти. Если вам нужно отслеживать более 100 сигналов одновременно, просто удвойте требуемый объем оперативной памяти или запустите второй экземпляр приложения.

Запустите приложение из пользовательского интерфейса

Вот пример использования для запуска приложения:

1. Создайте сигналы с тестовыми данными: создайте 2 сигнала, представляющих предсказания модели и истину. Оба сигнала имеют всего 2 значения каждый и идентичны, симулируя ситуацию с идеальной моделью.
2. Создание и запуск приложения: создайте шаблон для исторических метрик и запустите приложение.
3. Проверьте результаты: убедитесь, что приложение правильно рассчитало выбранную метрику MSE и она равна `0.0`.

Создайте сигналы с тестовыми данными

Сначала нам нужно создать сигналы с тестовыми значениями истинных данных и предсказаний модели для оценки метрик. Это можно сделать быстро с использованием функции импорта данных сигнала на платформе.

1. Скачайте файл `signals.csv`.

   В указанном файле содержатся следующие данные:

   ```
   timestamp,sample_y_true,sample_y_pred
   2022-01-01T12:00:00,0.0,0.0
   2022-01-01T13:00:00,1.0,1.0
   ```

   > ⓘ Примечание
   >
   > Как видите, это всего лишь тестовый образец идеальной модели.

2. Войдите в систему и перейдите в раздел **Сигналы**
3. Нажмите кнопку **Импорт данных сигнала** в правом верхнем углу страницы.
4. В открывшемся диалоговом окне загрузите файл, установите флажок **Пользовательский формат даты** и установите **Формат даты** в `yyyy-MM-dd HH:mm:ss`. Установите разделитель на запятую (`,`) и нажмите **Импортировать**.

Создание и запуск приложения

Следующим шагом является создание и запуск приложения. одробные инструкции по регистрации и запуску приложений предоставлены в документации платформы.

1. Создайте новый шаблон приложения, дайте ему значимое имя и запомните для будущего использования.
2. Создайте новый шаг с произвольным именем.
3. Укажите следующие параметры внутри созданного шага:

   ``Параметры процесса выполнения ``:

   ```
   {
       "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
       "MAIN_HANDLER": "metrics_evaluate",
       "RESOURCE_TYPE": "FLOW_JOB",
       "RAM_REQUEST_MB": 192,
   ```

```
    "RAM_LIMIT_MB": 192
}
```


Входные параметры:


```
{
    "fields": [
        {
            "fieldType": "LONG_TEXT",
            "name": "metric_description",
            "label": "metric_description",
            "isRequired": true,
            "description": "JSON-parsable metric description, e.g. '{\"metrics\": [{\"prediction_signa
        },
        {
            "fieldType": "TEXT",
            "name": "data_step_size",
            "label": "data_step_size",
            "isRequired": true,
            "defaultValue": "0s",
            "description": "pandas.Timedelta-parsable data step size"
        },
        {
            "fieldType": "TEXT",
            "name": "output_signal_prefix",
            "label": "output_signal_prefix",
            "isRequired": false,
            "defaultValue": "",
            "description": "Output signal prefix"
        },
        {
            "fieldType": "TEXT",
            "name": "heartbeat_internal_signal",
            "label": "heartbeat_internal_signal",
            "isRequired": false,
            "description": "Signal for writing internal smart heartbeat of app"
        },
        {
          "fieldType": "TEXT",
          "name": "heartbeat_startup_silence_period",
          "label": "heartbeat_startup_silence_period",
          "isRequired": false,
          "description": "How many iterations to wait until starting to send heartbeat"
        }
    ]
}
```


`LOGGER_LEVEL`	Уровень логирования. Подробнее см. в руководстве Логирование.
`metric_description`	Строка, содержащая список словарей с описаниями метрик в формате JSON. Подробнее см. ниже.
`data_step_size`	Размер шага данных, в формате `pandas.Timedelta`, например, `12 H`, `5 min`, `3 seconds`, или `None` для получения данных без предобработки.
`output_signal_prefix`	Необязательный префикс для выходных сигналов.
`heartbeat_internal_signal`	Необязательный сигнал для внутреннего умного хартбита приложения. При указании приложение

	будет записывать хартбит в конце каждой итерации цикла.

Параметр `metric_description` представляет собой набор параметров для списка выбранных метрик, включая:

`prediction_signal`	Публичный идентификатор сигнала с предсказаниями модели.
`observation_signal`	Публичный идентификатор сигнала с истинными значениями.
`metric_name`	Метрика, которую необходимо рассчитать. Должна быть одной из доступных метрик: `MAE`, `MAPE`, `RMSE`, `R2`, `recall`, `precision`, `F1`, `acceptance_rate`.
`aggregation_window`	Определяет временной интервал для агрегации метрик в формате `pandas.Timedelta`. Он указывает продолжительность, за которую вычисляются метрики, например, `12 H`, `5 min` или `3 seconds`.
`prediction_future_shift`	Задержка между временными метками прогноза модели и временными метками истинных данных, в формате `pandas.Timedelta`, например, `12 H`, `5 min`, `3 seconds`.
`output_name`	Необязательное имя выходного сигнала с метрическими значениями. Если используется *output_signal_prefix* – то добавляется префикс к имени сигнала.
`remove_plateau`	Ссылочный сигнал для удаления плато. Дополнительную информацию о данном параметре можно найти в разделе Убрать плато.
`kwargs`	Необязательный словарь ключевых аргументов, включая: `ignore_nan`, `y_true_fill` и `y_pred_fill`. См. Дополнительные параметры метрик для получения дополнительной информации.

4. Создайте новое приложение на основе только что созданного шаблона и запустите его с указанными параметрами:

`Environment`	`REMOTE`
`LOGGER_LEVEL`	`INFO`
`metric_description`	`{"metrics": [{"prediction_signal": "sample_y_pred", "observation_signal": "sample_y_true",`

	`"metric_name": "MAE", "aggregation_window": "1 H"}]}`
`Data_step_size`	`1 H`
`Output_signal_prefix`	`test-`

5. Дождитесь, пока статус приложения не изменится на `ВЫПОЛНЯЕТСЯ`, затем перейдите на страницу выполнения шага и обновите ее до тех пор, пока не увидите сообщение журнала `INFO No new data`. После этого вы можете безопасно завершить шаг.

> ⓘ Примечание
>
> Пользователи могут настроить регулярность запусков для приложения метрик, используя параметр `batch_interval`. Если этот параметр не указан, будет использован параметр `data_step_size`. Если оба параметра указаны, будет применено минимальное значение из двух указанных параметров.

Проверьте результаты

Если приложение завершится как ожидается, вы должны обнаружить новый сигнал `test-sample_y_true-sample_y_pred-3600-MAE`. Убедитесь, что у него есть 1 значение, равное `0.0`, поскольку у сигналов предсказания и истины одинаковые значения.

Поздравляем! Вы только что освоили основы метрик!

Исторический шаг

Исторический шаг рассчитывает метрики для всех значений в желаемом периоде дат и записывает их все в одном пакете.

Для запуска исторического шага вам понадобятся другие `Job params`

``Параметры процесса выполнения ``:

```
{
    "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
    "MAIN_HANDLER": "metrics_history",
    "RESOURCE_TYPE": "STEP_JOB",
    "RAM_REQUEST_MB": 192,
    "RAM_LIMIT_MB": 192
}
```

И два дополнительных входных параметра:

`min_dt`	Необязательная начальная дата и время. Если пусто, используется первое доступное дата и время.
`max_dt`	Необязательная конечная дата и время. Если пусто, используется последнее доступное дата и время.

Пример использования

Ниже демонстрируется, как настроить приложение для запуска метрик каждые 5 минут и с исторический периодом в 2 часа.

Чтобы достичь желаемых результатов, вам нужно будет правильно настроить параметр aggregation window, настроить параметры процесса выполнения и указать дополнительные параметры ввода.

Начните с конфигурации словаря. Установите параметр aggregation window на 5 минут, как показано ниже:

```
{
  "metrics": [
    {
      "prediction_signal": "sample_x_pred",
      "observation_signal": "sample_x_true",
      "metric_name": "MAE",
      "aggregation_window": "5 min"
    }
  ]

}
```

Предоставьте остальную часть конфигурации, если это требуется для вашего случая. Кроме того, убедитесь, что требуемые требуемые сигналы созданы на Платформе.

Измените ранее созданный шаг в шаблоне приложения или создайте новый шаг со следующими параметрами процесса выполнения:

```
{
    "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
    "MAIN_HANDLER": "metrics_history",
    "RESOURCE_TYPE": "STEP_JOB",
    "RAM_REQUEST_MB": 192,
    "RAM_LIMIT_MB": 192
}
```

Измените параметры ввода, чтобы включить исторический шаг, как показано ниже:

```
{
    "fields": [
        {
            "fieldType": "LONG_TEXT",
            "name": "metric_description",
            "label": "metric_description",
            "isRequired": true,
            "description": "JSON-parsable metric description, e.g. '{\"metrics\": [{\"prediction_signal\": \"
        },
        {
            "fieldType": "TEXT",
            "name": "data_step_size",
            "label": "data_step_size",
            "isRequired": true,
            "defaultValue": "0s",
            "description": "pandas.Timedelta-parsable data step size"
        },
        {
            "fieldType": "TEXT",
            "name": "output_signal_prefix",
            "label": "output_signal_prefix",
            "isRequired": false,
            "defaultValue": "",
            "description": "Output signal prefix"
        },
```

```
        {
            "fieldType": "TEXT",
            "name": "heartbeat_internal_signal",
            "label": "heartbeat_internal_signal",
            "isRequired": false,
            "description": "Signal for writing internal smart heartbeat of app"
        },
        {
          "fieldType": "TEXT",
          "name": "heartbeat_startup_silence_period",
          "label": "heartbeat_startup_silence_period",
          "isRequired": false,
          "description": "How many iterations to wait until starting to send heartbeat"
        },
        {
          "fieldType": "TEXT",
          "name": "min_dt",
          "label": "min_dt",
          "isRequired": false,
          "description": "Optional start datetime for historical data"
        },
        {
          "fieldType": "TEXT",
          "name": "max_dt",
          "label": "max_dt",
          "isRequired": false,
          "description": "Optional end datetime for historical data"
        }
    ]
}
```

Откройте ваше приложение на Платформе и запустите шаг с требуемой конфигурацией:

```
{
  "REMOTE": true,
  "LOGGER_LEVEL": "INFO",
  "metric_description": "{\"metrics\": [{\"prediction_signal\": \"sample_y_pred\", \"observation_signal\": \"s
  "data_step_size": "5 min",
  "output_signal_prefix": "test-",
  "min_dt": "2022-01-01T12:00:00",  // Start time
  "max_dt": "2022-01-01T14:00:00"   // End time (2 hours later)
}
```

Таким образом, эта конфигурация обеспечит расчет метрик каждые 5 минут за указанный 2-часовой исторический период.

Watchdog

В этом разделе мы запустим приложение Watchdog на платформе.

> Совет
>
> Вы можете использовать виджет мониторинга в реальном времени для визуализации результатов, созданных приложением Watchdog. Для получения подробной информации о настройке виджета обратитесь к разделу *Мониторинг в реальном времени* в документации платформы.

Необходимые условия

- Учетная запись на платформе.

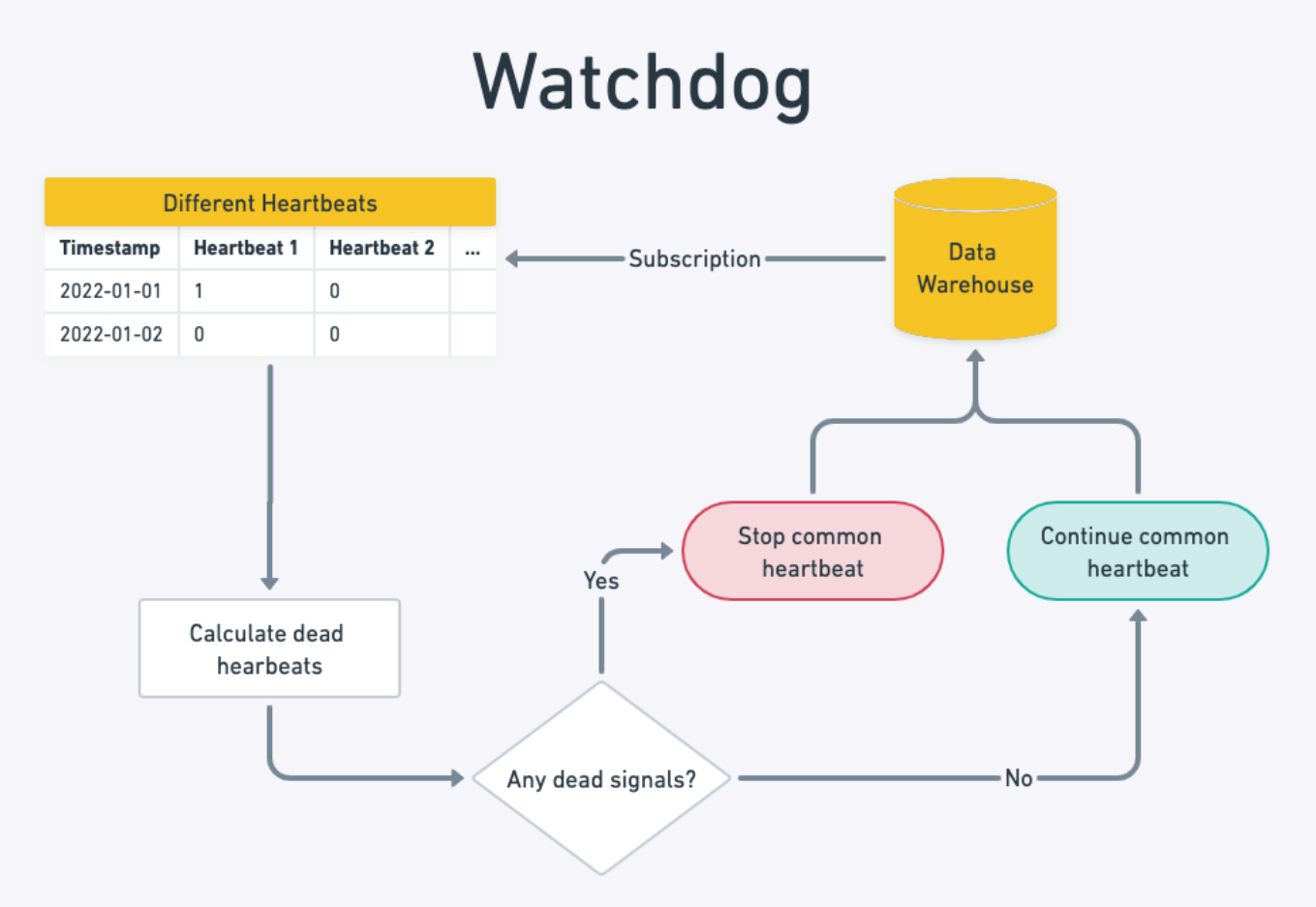
- Знакомство с концепциями приложений и сигналов. Прочитайте соответствующие разделы документации по платформе. Для доступа к документации платформы войдите в систему, нажмите на значок пользователя в левом нижнем углу страницы и выберите «Документация платформы».
- Знакомство с основными концепциями хартбитов.
- Настроенный реестр Docker с доступным образом `platform-sdk:latest`. Если вы не знакомы с реестрами Docker, прочтите, например, это руководство.

Обзор

Приложение Watchdog отвечает за генерацию системного сигнала хартбита и консолидацию информации от всех указанных в параметрах ввода сигналов хартбитов. Этот сервис регулярно проверяет все указанные сигналы хартбитов в параметрах ввода, и если он не обнаруживает новых данных, то перестает генерировать общий выходной сигнал хартбита.

Приложение Watchdog получает несколько общедоступных идентификаторов сигналов хартбитов и их соответствующих временных интервалов ожидания в параметрах ввода. После запуска приложение начинает генерацию обобщенного сигнала харбита. При работе приложения оно проверяет каждый сигнал хартбита, и если обнаруживает, что какой-то из них больше не генерирует данные, оно немедленно прекращает генерацию обобщенного сигнала хартбита.

На схеме ниже показана диаграмма приложения:



Запустите приложение из пользовательского интерфейса

В этом разделе описано, как создать и запустить приложение.

> ⓘ Примечание
>
> Подробные инструкции по регистрации и запуску приложений предоставлены в документации по платформе. Чтобы открыть документацию платформы, авторизуйтесь на платформе, нажмите на значок пользователя в левом нижнем углу страницы и выберите **Документация платформы**.

В этом разделе мы запустим приложение Watchdog на платформе.

1. Создайте новый шаблон приложения, дайте ему содержательное имя и запомните его для последующего использования.
2. Создайте новый шаг с произвольным именем. Оставьте значение по умолчанию в поле **Страница** и укажите следующие два параметра. Нажмите кнопку **Сохранить** под каждым параметром.

   ``Job params`:

   ```
   {
       "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
       "MAIN_HANDLER": "watchdog",
       "RESOURCE_TYPE": "FLOW_JOB",
       "RAM_REQUEST_MB": 110,
       "RAM_LIMIT_MB": 110
   }
   ```

   `Input params`:

   ```
   {
     "fields": [
       {
         "fieldType": "TEXT",
         "name": "heartbeats",
         "label": "heartbeats",
         "isRequired": true,
         "description": "List of heartbeats signals for aggregation separated by comma"
       },
       {
         "fieldType": "TEXT",
         "name": "heartbeat_output_signal",
         "label": "heartbeat_output_signal",
         "isRequired": true,
         "description": "The output signal for aggregated heartbeat"
       },
       {
         "fieldType": "TEXT",
         "name": "heartbeat_output_periodicity",
         "label": "heartbeat_output_periodicity",
         "isRequired": true,
         "defaultValue": "10sec",
         "description": "pandas.Timedelta-parsable aggregated heartbeat periodicity"
       },
       {
         "fieldType": "TEXT",
         "name": "app_hangs_timeout",
         "label": "app_hangs_timeout",
         "isRequired": true,
         "description": "pandas.Timedelta-parsable periodicity of the apps heartbeats separated by comma"
       },
   ```

```
    {
      "fieldType": "TEXT",
      "name": "heartbeat_internal_signal",
      "label": "heartbeat_internal_signal",
      "isRequired": false,
      "defaultValue": null,
      "description": "Optional signal for the heartbeat of watchdog application itself"
    },
    {
      "fieldType": "TEXT",
      "name": "heartbeat_startup_silence_period",
      "label": "heartbeat_startup_silence_period",
      "isRequired": false,
      "description": "How many iterations to wait until starting to send heartbeat"
    },
    {
      "fieldType": "TEXT",
      "name": "notification_group_name",
      "label": "notification_group_name",
      "isRequired": false,
      "description": "Existing notification group id, e.g. 'my-group'"
    },
     {
      "fieldType": "TEXT",
      "name": "batch_interval",
      "label": "batch_interval",
      "isRequired": false,
      "description": "Timedelta interval between the iterations in pandas.Timedelta format",
      "defaultValue": "1 s"
    },
    {
      "fieldType": "TEXT",
      "name": "journaling_on",
      "label": "journaling_on",
      "isRequired": false,
      "description": "Turns on event writing",
       "defaultValue": "False"
    },
    {
      "fieldType": "TEXT",
      "name": "event_category",
      "label": "event_category",
      "isRequired": false,
      "description": "Default category for events"
    },
    {
      "fieldType": "TEXT",
      "name": "event_message",
      "label": "event_message",
      "isRequired": false,
      "description": "Message that is written to recorded events if journaling is turned on"
    }
  ]
}
```

> **Предупреждение**
>
> Обратите внимание, что наблюдатель иногда может обнаруживать минимальные задержки сигнала, вызванные различными условиями работы, такими как проблемы с сетью или высокая загрузка ЦП. Для устранения этого мы рекомендуем корректировать ваши ожидания от хартбитов, особенно при работе с таймаутами около 5 секунд или более частыми интервалами. Например, если вы предполагаете обновление сигнала каждые 2 секунды, мы рекомендуем установить немного более низкий порог, например, `app_hangs_timeout=3sec`. Эта корректировка поможет предотвратить ненужное срабатывание наблюдателя.

Использование прогнозов модели в приложении Watchdog

Можно использовать приложение Watchdog для отслеживания того, что модель делает прогнозы с необходимой периодичностью. Для отслеживания этого добавьте сигнал прогноза модели в `heartbeats` и определите все другие необходимые параметры. В этом случае, если приложение с прогнозом модели зависло, это будет показано в выходном сигнале.

Уведомление, о том, что приложение упало

Если вы хотите, чтобы приложение отправляло уведомление в случае сбоя, вы можете добавить специальные параметры. Дополнительную информацию можно найти в Документации по приложению „Оповещение“.

Приостановка приложения на основе значений сигнала

Если вы хотите, чтобы приложение приостанавливало свои проверки на основе значений конкретных сигналов, вы можете добавить соответствующий параметр. Дополнительную информацию можно найти в Документации по Приложениям с уведомлениями.

Передача дополнительных пар ключ-значение для уведомлений

Если вы хотите включить дополнительные пары ключ-значение в свое уведомление, вы можете добавить соответствующий параметр `additional_keys`. Более подробная информация в Документации по Приложениям с уведомлениями

Отсылать одинаковые сообщения после истечения таймаута

По умолчанию приложение отправляет только уникальные уведомления. Однако вы можете использовать параметры `duplicate_message_timeouts` или `duplicate_message_timeout`, чтобы указать таймаут для идентичных сообщений. Больше информации в Документации приложения с нотификациями.

Resource Requirements

Watchdog requires only 90 MB of RAM to read from and write to up to 100 signals. It is recommended to set limit to 110 Mb in order to reserve 20 percents of RAM. If you need to read or write to more than 100 signals at once, simply double the required RAM or launch a second instance of the application.

Signal Notificator

В этом руководстве вы узнаете основы стандартного приложения Signal Notificator.

Обзор

Приложение Signal Notificator позволяет отслеживать определенные сигналы и получать уведомления, когда эти сигналы получают новые значения. Это приложение совместимо как с сигналами типа BLOB, так и с ROW, и предоставляет уведомления в следующем формате:

```
{"message": "{timestamp}: {value} (from {signal_public_id})"}
```

Входные параметры

Приложение ожидает следующий список входных параметров

`LOGGER_LEVEL`	Уровень логирования. Подробнее см. в руководстве Логирование.
`public_ids`	Публичные идентификаторы сигналов для мониторинга, разделенные запятыми. Здесь вы можете включить как сигналы типа ROW, так и BLOB.
`notification_group_name`	Имена существующих групп уведомлений, разделенные запятыми, куда отправлять оповещения, например, `my-group,my-group2`.
`check_for_new_data_interval`	(Необязательно) Указывает, как часто проверять наличие новых данных, в формате временных интервалов pandas, например, `10s`. Значение по умолчанию - 1 секунда.

Job parameters

У приложения есть следующие параметры:

```
{
    "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
    "MAIN_HANDLER": "signal_notificator",
    "RESOURCE_TYPE": "FLOW_JOB",
    "RAM_REQUEST_MB": 110,
    "RAM_LIMIT_MB": 110
}
```

Запуск приложения из пользовательского интерфейса

В этой части вы узнаете, как пошагово создать и запустить приложение Signal Notificator на платформе.

1. Создание сигналов: создание сигналов.
2. Создание группы уведомлений: создание необходимых групп уведомлений.
3. Создание и запуск приложения: запуск приложения.
4. Проверьте результаты: убедитесь, что у вас есть уведомление.

Создание сигналов

> ⓘ Примечание
>
> Если вы запускаете это приложение для мониторинга существующих сигналов, вы можете пропустить этот шаг.

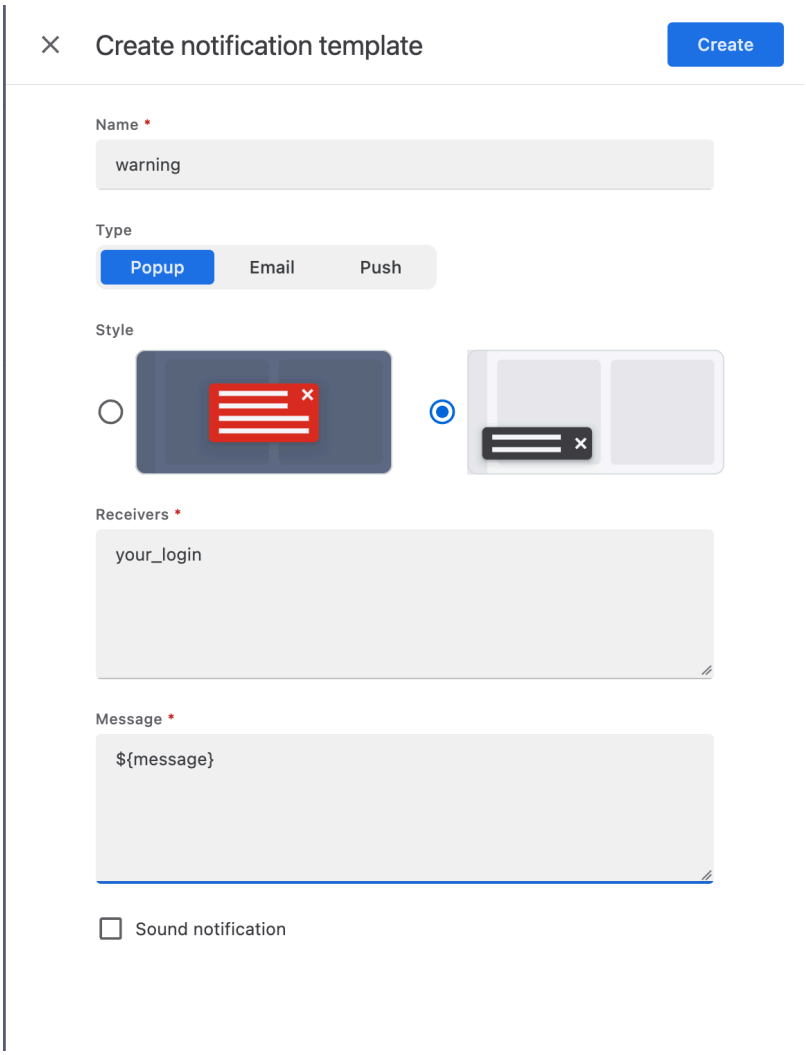
На этом этапе мы создадим пару простых сигналов исключительно для иллюстративных целей.

1. Войдите в систему и перейдите в раздел **Сигналы**.
2. Щелкните кнопку **Создать сигнал** в правом верхнем углу окна.
3. В открывшемся диалоговом окне введите общедоступный идентификатор сигнала `signal_notificator_row` и установите селектор **Тип данных** на **ROW (Число)**. Оставьте все остальные поля по умолчанию и нажмите **Создать**.

Повторите те же шаги для сигнала `signal_notificator_blob`, но с **Тип данных BLOB (Text)**

Создание группы уведомлений

1. Войдите в систему и перейдите в раздел **Уведомления** .
2. Щелкните кнопку **Создать группу** в правом верхнем углу окна.
3. В открывшемся диалоговом окне введите имя группы уведомлений `my-notifications` и нажмите **Создать**.
4. На странице группы уведомлений добавьте всплывающий шаблон для вашего пользователя, заполните параметры, как показано на картинке ниже, и нажмите **Создать**.



Создание и запуск приложения

Следующим шагом является создание и запуск приложения.

> **Примечание**
>
> Подробные инструкции по регистрации и запуску приложений предоставлены в документации платформы. Чтобы открыть документацию платформы, авторизуйтесь на платформе, нажмите на значок пользователя в левом нижнем углу страницы и выберите **Документация платформы**.

1. Создайте новый шаблон приложения, дайте ему понятное имя и запомните его для последующего использования.
2. Создайте новый шаг с произвольным именем. Оставьте значение по умолчанию в поле **Страница** и укажите следующие два параметра. Нажмите кнопку **Сохранить** под каждым параметром.

   `Job params`:

   ```
   {
       "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
       "MAIN_HANDLER": "signal_notificator",
       "RESOURCE_TYPE": "FLOW_JOB",
       "RAM_REQUEST_MB": 110,
   ```

```
    "RAM_LIMIT_MB": 110
}
```

`Input params`:

```
{
  "fields": [
    {
      "fieldType": "TEXT",
      "name": "public_ids",
      "label": "public_ids",
      "isRequired": true,
      "description": "Public IDs of the signals to monitor, separated by commas"
    },
    {
      "fieldType": "TEXT",
      "name": "check_for_new_data_interval",
      "label": "check_for_new_data_interval",
      "isRequired": true,
      "description": "Specifies how often to check for new data, in pandas timedelta format, i.e.``10s
      "defaultValue": "1s"
    },
    {
      "fieldType": "TEXT",
      "name": "notification_group_name",
      "label": "notification_group_name",
      "isRequired": false,
      "description": "Existing notification group id, e.g. 'my-group'"
    }
  ]
}
```

3. Создайте новое приложение на основе только что созданного шаблона и запустите его с следующими параметрами:

`Environment`	`REMOTE`
`LOGGER_LEVEL`	`INFO`
`notification_group_name`	`my-notifications`
`public_ids`	`signal_notificator_row,signal_notificator_blob`

4. Дождитесь, пока приложение не изменит свой статус на `ВЫПОЛНЯЕТСЯ`.

Проверьте результаты

Попробуйте записать что-то в одном из созданных выше сигналов. Для этого

1. Войдите в систему и перейдите в раздел **Сигналы**.
2. Найдите один из созданных выше сигналов и нажмите **Добавить значение**. Добавьте любое значение.

Вы должны увидеть всплывающее уведомление с добавленным значением. Теперь вы можете безопасно завершить приложение.

Уведомление в случае сбоя приложения

Если вы хотите, чтобы приложение отправляло уведомление в случае сбоя, вы можете добавить специальные параметры. Более подробную информацию можно найти в Alert App doc.

Передача дополнительных пар ключ-значение в уведомления

Если вы хотите включить дополнительные пары ключ-значение в уведомление, вы можете добавить соответствующий параметр `additional_keys`. Более подробную информацию см. в Alert App doc

Latency measurer

В этом руководстве вы узнаете основы по работе с приложением для измерения задержки.

Обзор

Приложение для измерения задержки предоставляет функциональность для измерения полной продолжительности передачи данных через коннекторы загрузки и отправки. Если это время превышает определенный предел, инструмент может отправить вам уведомление. Вы предоставляете инструменту инструкции о том, какие сигналы использовать для отправки и приема данных. При активации инструмент генерирует импульс сигнала в указанные сигналы записи, одновременно инициируя чтение сигналов, назначенных для чтения. Таким образом, он определяет общее время продолжительности пути данных. После этого он записывает это время в специальный набор сигналов, предназначенных для хранения временных различий.

Кроме того, пользователи имеют возможность устанавливать правила уведомлений, включая пороговые значения и назначенные группы уведомлений. Эти правила облегчают доставку уведомлений в случае превышения расчетной разницы времени предопределенного порога. Кроме того, пользователь может указать специальные группы уведомлений, которые должны быть деактивированы, если порог превышен.

> ⓘ Примечание
>
> Измеритель задержки ожидает, что экземпляр платформы работает с SCADA. И ожидает, что коннекторы будут настроены так, чтобы данные из `signals_for_writing` были отправлены в SCADA через коннектор отправки, а `signals_for_reading` получали данные из SCADA через коннектор загрузки.

Входные параметры

Приложение ожидает следующий список входных параметров

`LOGGER_LEVEL`	Уровень логирования. Подробнее см. в руководстве Логирование.
`signals_for_writing`	Список общедоступных идентификаторов сигналов, разделенных запятой, куда приложение записывает импульс

	(временную метку записи в формате Unix).
`signals_for_reading`	Список общедоступных идентификаторов сигналов, разделенных запятой, которые следует читать.
`signals_time_difference`	Список общедоступных идентификаторов сигналов для записи временной разницы.
`frequency`	Как часто отправлять импульс.
`notification_rules`	Необязательный список правил уведомлений. Если не установлен, приложение не будет отправлять уведомлений.

Параметр `notification_rules` должен быть предоставлен в виде списка словарей, каждый из которых имеет следующий формат

```
class NotificationRule(BaseModel):
"""Schema for notifications description."""

threshold: float
"""Threshold in seconds."""
text: Optional[str] = Field(default="The latency is above threshold")
"""Text of notification."""
notification_groups: List[str]
"""Comma-separated list of notification group names."""
notification_groups_to_mute: Optional[List[str]]
"""Comma-separated list of notification group names to mute if this rule threshold is broken."""
```

Так, например, параметр `notification_rules` может быть установлен следующим образом:

```
[
    {"threshold": 20, "text": "Warning! The data is late.", "notification_groups": "support,ds"},
    {"threshold": 60, "text": "Error! The data is super late.", "notification_groups": "support,ds,factory"}
]
```

Job parameters

У приложения есть следующие параметры:

```
{
    "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
    "MAIN_HANDLER": "latency_measurer",
    "RESOURCE_TYPE": "FLOW_JOB",
    "RAM_REQUEST_MB": 192,
    "RAM_LIMIT_MB": 192
}
```

Запустите приложение из пользовательского интерфейса

В этой части вы узнаете, как пошагово создать и запустить измеритель задержки на платформе.

1. Создание сигналов: создайте пустой набор `signals_for_writing` и `signals_for_reading`.
2. Настройка коннекторов: создайте и настройте коннекторы отправки и загрузки.

3. Создание группы уведомлений: создайте необходимые группы уведомлений.
4. Создание и запуск приложения: запустите приложение.
5. Проверьте результаты: убедитесь, что у вас есть уведомление.

Создание сигналов

Мы создадим один сигнал для записи и один для чтения.

1. Войдите в платформу и перейдите в раздел **Сигналы**.
2. Нажмите кнопку **Создать сигнал** в верхнем правом углу окна.
3. В открывшемся диалоговом окне введите открытый идентификатор сигнала `latency_signal_for_writing` и установите селектор **Тип данных** в **ROW (Число)**. Оставьте все остальные поля по умолчанию и нажмите **Создать**.

Повторите те же шаги для сигнала `latency_signal_for_reading`.

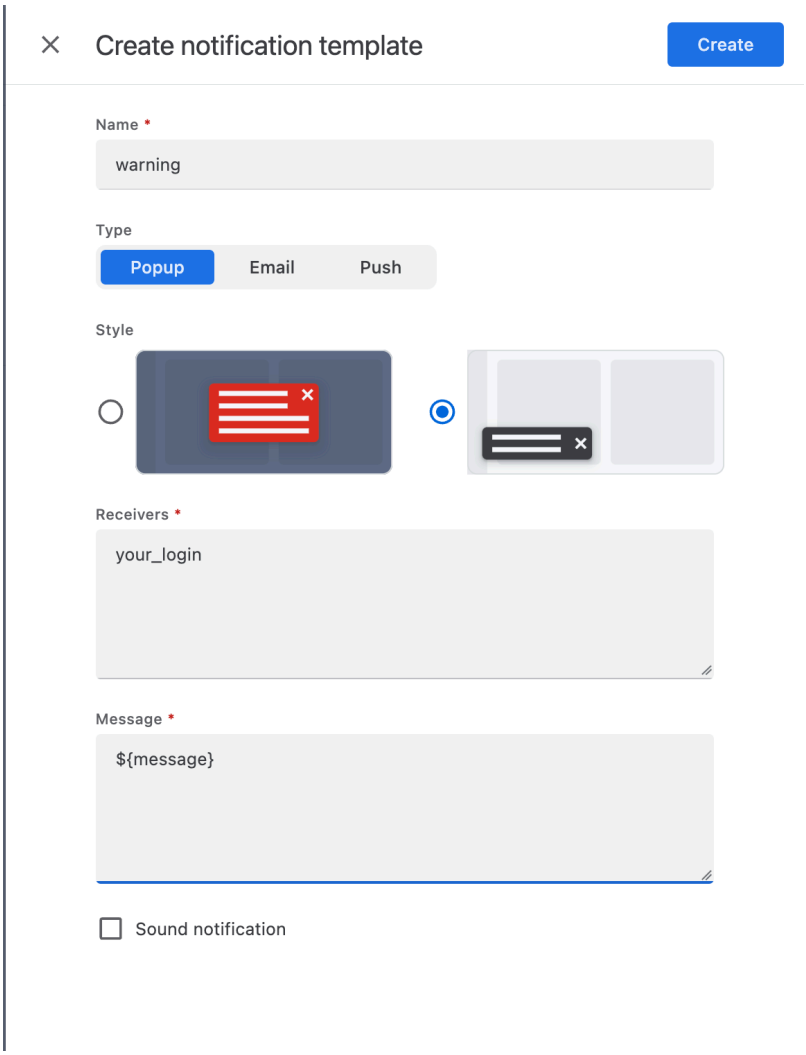
Настройка коннекторов

Теперь вам нужно настроить коннектор для записи (sink connector) для `latency_signal_for_writing` и коннектор для чтения (source connector) для `latency_signal_for_reading`. Или просто добавьте к существующим коннекторам, если они уже настроены, и вы запускаете приложение в экземпляре с SCADA.

Подробную инструкцию по настройке коннекторов вы найдете в документации к платформе в разделе  **Коннекторы**.

Создание группы уведомлений

1. Войдите в платформу и перейдите в раздел **Уведомления**  .
2. Нажмите кнопку **Создать группу** в верхнем правом углу окна.
3. В открывшемся диалоговом окне введите имя группы уведомлений `latency-notifications` и нажмите **Создать**.
4. На странице группы уведомлений добавьте шаблон всплывающего окна для вашего пользователя, заполните параметры, как показано на картинке ниже, и нажмите **Создать**.



Создание и запуск приложения

Следующим шагом является создание и запуск приложения.

> **Примечание**
>
> Подробные инструкции по регистрации и запуску приложений приведены в документации к платформе. Чтобы открыть документацию к платформе, авторизуйтесь на платформе, нажмите на иконку пользователя в левом нижнем углу страницы и выберите **Документация по платформе**.

1. Создайте новый шаблон приложения, дайте ему содержательное имя и запомните его для последующего использования.
2. Создайте новый шаг с произвольным именем. Оставьте значение по умолчанию в поле **Страница** и укажите следующие два параметра. Нажмите кнопку **Сохранить** под каждым параметром.

   `Job params`:

   ```
   {
       "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
       "MAIN_HANDLER": "latency_measurer",
       "RESOURCE_TYPE": "FLOW_JOB",
       "RAM_REQUEST_MB": 192,
   ```

```
    "RAM_LIMIT_MB": 192
}
```

Input params:

```
{
  "fields": [
    {
      "fieldType": "TEXT",
      "name": "signals_for_writing",
      "label": "signals_for_writing",
      "isRequired": true,
      "description": "Comma-separated list of signal public ids where app writes a pulse (the writing
    },
    {
      "fieldType": "TEXT",
      "name": "signals_for_reading",
      "label": "signals_for_reading",
      "isRequired": true,
      "description": "Comma-separated list of signal public ids to be red"
    },
    {
      "fieldType": "TEXT",
      "name": "signals_time_difference",
      "label": "signals_time_difference",
      "isRequired": true,
      "description": "Comma-separated list of signal public ids to write a time difference"
    },
    {
      "fieldType": "TEXT",
      "name": "frequency",
      "label": "frequency",
      "isRequired": true,
      "defaultValue": "1 s",
      "description": "pandas.Timedelta-parsable frequency, indicating how often to send a pulse"
    },
    {
      "fieldType": "TEXT",
      "name": "notification_rules",
      "label": "notification_rules",
      "isRequired": false,
      "defaultValue": "[{\"threshold\": 5, \"text\": \"Warning! The data is late\", \"notification_gro
      "description": "Optional list of notification rules. If not set, application won't be sending an
    }
  ]
}
```

3. Создайте новое приложение на основе только что созданного шаблона и запустите его с следующими параметрами:

`Environment`	`REMOTE`
`LOGGER_LEVEL`	`INFO`
`signals_for_writing`	`latency_signal_for_writing`
`signals_for_reading`	`latency_signal_for_reading`
`signals_time_difference`	`latency_signal_td`
`frequency`	`1 s`
`notification_rules`	`[{"threshold": 5, "text": "Внимание! Данные задерживаются",`

```
"группы_уведомлений": "latency-notifications"}]
```

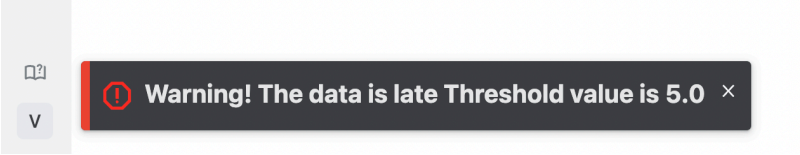
4. Дождитесь, пока статус приложения не изменится на `ВЫПОЛНЯЕТСЯ`, затем перейдите на страницу выполнения шага и обновите ее, пока вы не увидите эти строки:

   ```
   ... ...
   ... | INFO     | Published message 'Warning! The data is late Threshold value is 5.0' ...
   ```

   Затем вы можете безопасно завершить шаг.

Проверьте результаты

Если у вас нет SCADA на dev стенде, вы увидите уведомление с указанным текстом:



Уведомлять, если приложение упало с ошибкой

Если вы хотите, чтобы приложение отправляло уведомление в случае сбоя, вы можете добавить специальные параметры. Дополнительную информацию можно найти в Документации по приложению для оповещения.

Передача дополнительных пар ключ-значение в уведомления

Если вы хотите добавить дополнительные пары ключ-значение в ваше уведомление, вы можете добавить соответствующий параметр `additional_keys`. Дополнительную информацию см. в Документации по приложению для оповещения.

Jupyter Notebook Reports

В этом руководстве вы узнаете основы использования приложения для отчетов в Jupyter Notebook, а также как правильно настроить данное приложения из пользовательского интерфейса платформы.

Обзор

Отчет в формате Jupyter notebook - это приложение для выполнения блокнотов Jupyter на платформе. Приложение позволяет пользователю выполнять блокнот внутри контейнера Docker и анализировать результаты отчета: значения сигналов или блокнот Jupyter в виде HTML- или PDF-страницы.

Стандартный рабочий процесс с отчетом в формате Jupyter notebook следующий:

1. Пользователь создает образ Docker с установленным SDK и помещает туда все необходимые блокноты Jupyter.

2. Пользователь запускает приложение с точкой входа `jupyter_report` и необходимыми параметрами ввода.
3. Для запуска генерации отчета пользователь должен отправить сообщение с заданной структурой в управляющий сигнал BLOB:

```
{
    "min_dt": "2023-01-12 22:07:13",  # str in ISO format, min dt of data needed for report
    "max_dt": "2023-03-06 12:10:38",  # str in ISO format, max dt of data needed for report
    "report_id": 1,  # int, number of report to execute
    "report_artifact_name": "report_2023-01-12_22:07:13_2023-03-06_12:10:38_8jks7fkj2.html"  # str, with
}
```

4. Приложение `jupyter_report` запускает блокнот, используя входные параметры `report_paths` и `report_id` из сообщения BLOB. Оно выбирает блокнот с индексом `report_id` из путей, указанных во входном параметре `report_paths`, где индексация начинается с 1.
5. После выполнения блокнота приложение сохраняет отчет в артефакт, помеченный как `report_artifact_name`.

Ниже вы найдете пошаговое руководство по запуску отчета в формате Jupyter notebook.

Необходимые условия

- Учетная запись на платформе.
- Знание концепций приложений, сигналов и артефактов. Прочтите соответствующие разделы документации по платформе. Чтобы получить доступ к документации по платформе, войдите в платформу, нажмите на иконку пользователя в левом нижнем углу страницы и выберите «Документация по платформе».
- Настроенный реестр Docker с доступным образом `platform-sdk:latest`. Если вы не знакомы с реестрами Docker, прочтите, например, это руководство.
- Знакомство с концепциями Jupyter Notebooks

Требования к ресурсам

Невозможно установить стандартизированный набор требований к ресурсам для отчетов в формате Jupyter Notebook, поскольку требования будут варьироваться в зависимости от размера обрабатываемых данных. Поэтому рекомендуется оценить использование ОЗУ отчета перед развертыванием его в производственной среде. Предлагается начать с 256 МБ ОЗУ и увеличивать его в случае ошибки нехватки памяти (OOM) во время выполнения отчета.

Настройте приложение из пользовательского интерфейса

Вот пример использования для запуска приложения:

1. Добавьте блокноты Jupyter в образ: Добавьте файлы отчетов Jupyter `.ipynb` в образ SDK, см. эту команду для получения дополнительной информации. Блокноты Jupyter ищутся по пути к файлу, поэтому они должны быть доступны внутри образа Docker с использованием команды ADD в файле Dockerfile или добавлены с использованием Docker

CLI. После добавления блокнотов в образ результаты необходимо загрузить в реестр Docker.
2. Создание и запуск приложения: Создайте шаблон приложения для отчетов Jupyter. Шаблон приложения должен ссылаться на образ из предыдущего шага. К шаблону можно добавить произвольное количество параметров `TEXT`. Эти параметры будут переданы и заменены с использованием jinja. Это предоставляет дополнительные возможности конфигурации для отчетов в формате блокнота Jupyter.
3. Запустите приложение для отчета

Добавьте блокноты Jupyter в образ

Сначала вам нужно создать блокнот Jupyter и протестировать его локально. Если перед выполнением блокнота требуется дополнительная конфигурация, создайте параметры шаблона, используя синтаксис jinja. Прочтите эти параметры внутри блокнота и переделайте логику кода для использования этих параметров. Например, следующая ячейка может быть добавлена в блокнот:

```
pid = '{{ test_signal }}'
```

Если параметр `test_signal` будет добавлен в шаблон и настроен пользователем с `'user_signal'`, то этот блок будет преобразован в

```
pid = 'user_signal'
```

перед выполнением ячейки.

> ⓘ Примечание
>
> Обратите внимание, что тип значения параметра зависит от указанного в схеме `input_params` значения `fieldType`. Следовательно, вы должны обрабатывать параметры внутри блокнота соответствующим образом. Например, параметры типа `TEXT` следует заключать в кавычки, в то время как параметр типа `NUMBER` может быть передан как есть.

```
my_text_param: float = float('{{ text_param }}')
my_number_param: float = {{ number_param }}
```

Здесь можно использовать не только параметры, переданные через `input_params`. Вы также можете использовать `min_dt` и `max_dt`, переданные через управляющий сигнал BLOB. Они передаются в виде строки в формате ISO.

```
from datetime import datetime

min_dt = datetime.fromisoformat('{{ min_dt }}')
max_dt = datetime.fromisoformat('{{ max_dt }}')
```

Вы можете прочитать пример блокнота с рекомендациями в примере блокнота отчета Jupyter. Или вы можете загрузить `пример блокнота jupyter_report_example.ipynb`.

Создание и запуск приложения

Следующим шагом является создание и запуск отчета в формате блокнота Jupyter с параметром test_signal из приведенного выше примера.

> **Примечание**
>
> Подробные инструкции по регистрации и запуску приложений предоставлены в документации по платформе. Чтобы открыть документацию по платформе, авторизуйтесь на платформе, нажмите на иконку пользователя в левом нижнем углу страницы и выберите **Документация по платформе**.

1. Создайте новый шаблон приложения, дайте ему значимое имя и запомните его для последующего использования.
2. Создайте новый шаг с произвольным именем. Оставьте значение по умолчанию в поле **Страница** и укажите следующие два параметра. Нажмите кнопку **Сохранить** под каждым параметром.

   `Job params`:

   ```
   {
       "IMAGE_NAME": "cr.yandex/crpdovt88qa2r62fpmge/platform-sdk:latest",
       "MAIN_HANDLER": "jupyter_report",
       "RESOURCE_TYPE": "STEP_JOB",
       "RAM_REQUEST_MB": 1024,
       "RAM_LIMIT_MB": 1024
   }
   ```

   `Input params`:

   ```
   {
     "fields": [
       {
         "fieldType": "TEXT",
         "name": "report_paths",
         "label": "report_paths",
         "isRequired": true
       },
       {
         "fieldType": "TEXT",
         "name": "report_artifact_names",
         "label": "report_artifact_names",
         "isRequired": false
       },
       {
         "fieldType": "TEXT",
         "name": "trigger",
         "label": "trigger",
         "isRequired": false
       },
       {
         "fieldType": "TEXT",
         "name": "exclude_input",
         "label": "exclude_input",
         "isRequired": false,
         "defaultValue": false
       },
       {
         "fieldType": "TEXT",
         "name": "test_signal",
         "label": "test_signal",
         "isRequired": true
   ```

```
    }
  ]
}
```

`LOGGER_LEVEL`	Необязательный уровень логгирования. См. подробнее в руководстве Логирование.
`report_paths`	Абсолютные пути к блокнотам Jupyter, разделенные запятой, например, `/opt/notebook_1.ipynb,/opt/notebook_2.ipynb`.
`trigger`	Имя сигнала BLOB-триггера.
`exclude_input`	Если блокнот отчета не должен содержать исходный код, то этот параметр должен быть установлен в True. По умолчанию параметр установлен в False.
`test_signal`	Специфический параметр отчета, который должен быть подставлен в отчет прямо перед выполнением. Пользователь может определить больше подобных параметров при необходимости, замена в шаблоне будет выполнена на основе имени параметра и специального синтаксиса jinja.

3. Создайте новое приложение на основе только что созданного шаблона и запустите его с указанными параметрами:

`Environment`	`REMOTE`
`LOGGER_LEVEL`	`INFO`
`report_paths`	Путь к блокнотам Jupyter, разделенным запятой, например, `/opt/notebook_1.ipynb,/opt/notebook_2.ipynb`.
`trigger`	`my_control_signal`
`test_signal`	`jupyter_notebook_signal`

4. Запишите в `my_control_signal` следующий JSON:

```
{
    "min_dt": "2023-01-12 22:07:13",
    "max_dt": "2023-03-06 12:10:38",
    "report_id": 1,
    "report_artifact_name": "report_output.html"

}
```

   Здесь будет выполнен первый отчет из `report_paths`, так как пользователь указал `"report_id": 1`. Результат выполнения будет сохранен в формате `.html` и загружен в артефакты с меткой `report_output.html`.

5. Дождитесь появления в логах строки `Finished report execution on the platform.`, что означает успешное выполнение приложения. Если статус `СБОЙ`, вы можете перейти внутрь шага и изучить логи.

Одноразовый запуск отчета Jupyter

В некоторых случаях Jupyter Notebook может быть запущен только один раз, это может быть полезно в случаях, если приложение для отчета запланировано и не использует никакой входной сигнал в качестве триггера. Чтобы запустить одноразовый отчет Jupyter для приложения, необходимо выполнить следующие шаги: 1. Параметр `trigger` должен быть пустым. 2. Параметр `report_artifact_names` должен содержать имя для каждого отчета, указанного в параметре `report_paths`. 3. Все остальные параметры можно заполнить как показано в примере выше. После запуска, приложение запустит все отчеты из параметра `report_paths`. Приложение также будет подставлять входные параметры приложения в отчеты и называть каждый отчет в соответствии с параметром `report_artifact_names`. Порядок имен используется для поиска соответствующего имени артефакта для пути к отчету. Кроме того, каждый отчет будет загружен на платформу с расширением, указанным в имени артефакта отчета.

Проверьте результаты

Если приложение завершится как ожидается, вы должны найти `report_output.html` в разделе артефактов платформы. Как только пользователь загрузит артефакт, его можно открыть как HTML-файл в любом браузере, чтобы увидеть результат выполнения ячейки.

Поздравляю! Вы только что освоили основы отчетов в формате блокнота Jupyter!

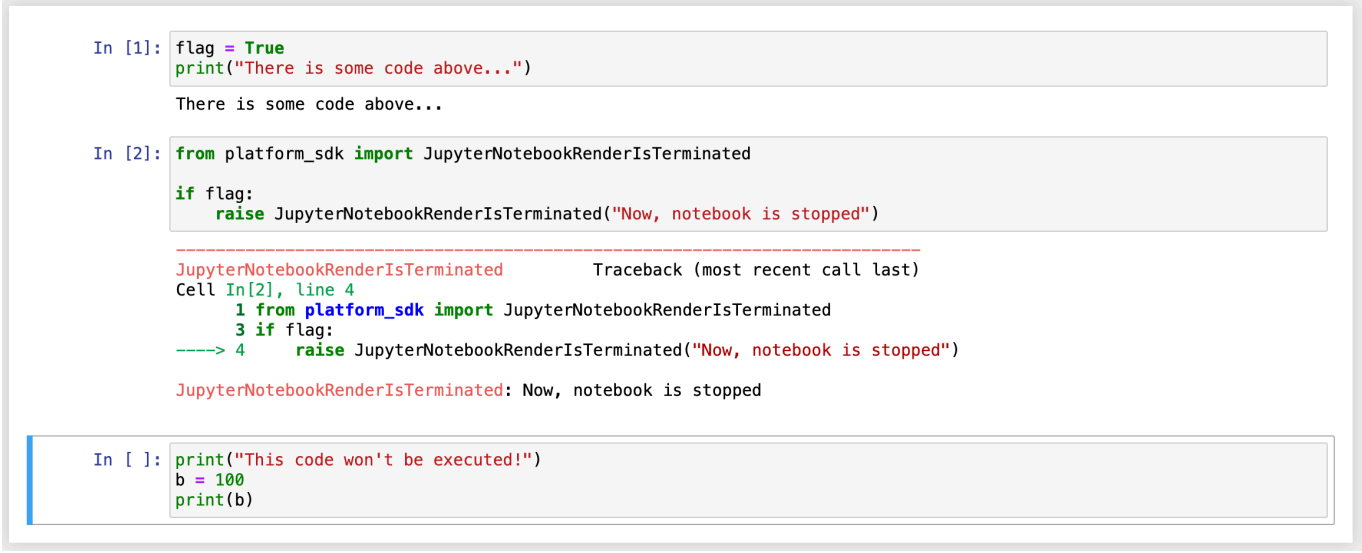
Прерывание выполнения блокнота

Если вы хотите приостановить выполнение блокнота в конкретной точке и предотвратить выполнение оставшихся ячеек, вы можете использовать исключение `JupyterNotebookStoppedError`. Для этого вызовите это исключение с пользовательским сообщением, например:

```
from platform_sdk import JupyterNotebookStoppedError

raise JupyterNotebookStoppedError("Your custom message")
```

Используя этот подход, выполнение блокнота будет принудительно прервано, и блокнот будет сохранен в текущем состоянии. Если вы хотите включить в сохраненный блокнот исходные ячейки, вы можете передать опцию `exclude_input=False`:



```
In [1]: flag = True
        print("There is some code above...")

        There is some code above...

In [2]: from platform_sdk import JupyterNotebookRenderIsTerminated

        if flag:
            raise JupyterNotebookRenderIsTerminated("Now, notebook is stopped")

        ---------------------------------------------------------------------------
        JupyterNotebookRenderIsTerminated         Traceback (most recent call last)
        Cell In[2], line 4
              1 from platform_sdk import JupyterNotebookRenderIsTerminated
              3 if flag:
        ----> 4     raise JupyterNotebookRenderIsTerminated("Now, notebook is stopped")

        JupyterNotebookRenderIsTerminated: Now, notebook is stopped

In [ ]: print("This code won't be executed!")
        b = 100
        print(b)
```

Или передать `exclude_input=True`

```
There is some code above...

---------------------------------------------------------------------------
JupyterNotebookRenderIsTerminated         Traceback (most recent call last)
Cell In[5], line 4
      2 from platform_sdk import JupyterNotebookRenderIsTerminated
      3 if flag:
----> 4     raise JupyterNotebookRenderIsTerminated("Now, notebook is stopped")

JupyterNotebookRenderIsTerminated: Now, notebook is stopped
```

Пример отчета в формате Jupyter

1. Шаблоны Jinja

В этом блокноте вы можете использовать шаблоны Jinja для передачи параметров. Для этого необходимо указать необходимые параметры в схеме `input_params` приложения и использовать их внутри этого блокнота.

Например, чтобы передать общедоступный идентификатор сигнала, вы можете добавить `pid_parameter` в схему `input_params`:

```
{
  "fields": [
    {
      "fieldType": "TEXT",
      "name": "pid_parameter",
      "label": "pid_parameter",
      "isRequired": true
    }
  ]
}
```

Затем используйте его внутри этого блокнота следующим образом:

```
[1]:
```

```
signal_pid = '{{ pid_parameter }}'
```

Обратите внимание, что тип параметра зависит от `fieldType`, указанного в схеме `input_params`. Если вы передали параметр `TEXT`, вы должны заключить его в кавычки, в то время как параметр `NUMBER` можно использовать как есть. Так что для следующей схемы:

```
{
  "fields": [
    {
      "fieldType": "NUMBER",
      "name": "threshold",
      "label": "threshold",
      "isRequired": true
    }
  ]
}
```

Вы можете использовать:

```
[2]:
```

```
threshold = {{ threshold }}
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 threshold = {{ threshold }}

NameError: name 'threshold' is not defined
```

Вы также всегда можете использовать параметры `max_dt` и `min_dt`, так как они передаются внутри BLOB в сигнале `trigger` в виде строк в формате ISO. Поэтому вы должны заключить их в кавычки:

```
[3]:
```

```
from datetime import datetime

min_dt = datetime.fromisoformat('{{ min_dt }}')
max_dt = datetime.fromisoformat('{{ max_dt }}')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[3], line 3
      1 from datetime import datetime
----> 3 min_dt = datetime.fromisoformat('{{ min_dt }}')
      4 max_dt = datetime.fromisoformat('{{ max_dt }}')

ValueError: Invalid isoformat string: '{{ min_dt }}'
```

2. Чтение данных

Почти все отчеты потребуют чтения данных. Следует помнить, что эта операция может потреблять много времени и памяти. В связи с этим мы предлагаем вам несколько простых правил, которые могут помочь вам создать оптимизированный отчет в формате Jupyter Notebook

Используйте чтение с агрегацией, если это возможно, вместо чтения необработанных данных и их повторной выборки внутри блокнота.

Например, если вам нужно агрегировать данные за 10 минут с усреднением, запросите данные с шагом в 10 минут, вместо необработанных данных. Это будет быстрее и потребует меньше памяти.

```
[4]:
```

```
from datetime import timedelta
from platform_sdk import SignalReaderClient, IntervalOperator

signal_reader = SignalReaderClient()

df = signal_reader.read_without_custom_processors(
    min_dt=min_dt,
    max_dt=max_dt,
    public_ids=[signal_pid],
    granularity=timedelta(minutes=10),
    interval_op=IntervalOperator.AVG,
)
df
```

```
2024-06-11 17:23:45.447 | WARNING  | platform_sdk.integrations.utils.deploy_config:platform_version:60 - Can no
2024-06-11 17:23:45.448 | WARNING  | platform_sdk.integrations.utils.deploy_config:platform_version:61 - 'AUTH_
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[4], line 4
      1 from datetime import timedelta
      2 from platform_sdk import SignalReaderClient, IntervalOperator
----> 4 signal_reader = SignalReaderClient()
      6 df = signal_reader.read_without_custom_processors(
      7     min_dt=min_dt,
      8     max_dt=max_dt,
   (...)
     11     interval_op=IntervalOperator.AVG,
     12 )
     13 df

File ~/work/platform_sdk/src/platform_sdk/integrations/stream_storage/signal/reader.py:82, in SignalReaderCli
     81 def __init__(self, auth: Optional[AuthService] = None, config: Optional[Configuration] = None) -> None
---> 82     super().__init__(auth, config)
     83     self._reader = SignalReaderAPI(auth=self.auth, config=self.config)
     84     self._ops = SignalsOps(auth=self.auth, config=self.config)

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/resource.py:81, in StreamStorageResource.__init__
     80 def __init__(self, auth: Optional[AuthService] = None, config: Optional[Configuration] = None) -> None
---> 81     super().__init__(auth, config)
     82     self._http_client = httpclient.HTTPClient(
     83         auth=self.auth,
     84         schema=self.config.config["integrations"]["stream-storage"]["schema"],
     85         hostname=self.config.config["integrations"]["stream-storage"]["hostname"],
     86     )
     87     self._rpc_client = rpc_client.RPCClient(
     88         auth=self.auth,
     89         schema=self.config.config["integrations"]["stream-storage"]["schema"],
     90         hostname=self.config.config["integrations"]["stream-storage"]["hostname"],
     91     )

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/resource.py:24, in AuthResource.__init__(self, a
     23 def __init__(self, auth: Optional[AuthService] = None, config: Optional[Configuration] = None):
---> 24     self._config = config or Configuration()
     25     self._auth = auth or AuthService(config=self.config)

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/config.py:16, in Configuration.__init__(self, da
```

```
     15 def __init__(self, data: Optional[dict] = None):
---> 16     self._data = data or self.load()

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/config.py:33, in Configuration.load(cls)
     26 auth_schema = os.environ.get("AUTH_SCHEMA", "https")
     27 strategy = os.environ.get("AUTH_STRATEGY", default="password")
     28 return {
     29     "integrations": {
     30         "auth": {
     31             "strategy": strategy,
     32             "schema": auth_schema,
---> 33             "hostname": os.environ["AUTH_ENDPOINT"],
     34             "realm": os.environ.get("APP_AUTH_REALM", default="platform"),
     35             "client_id": os.environ.get("AUTH_CLIENT_ID", default="ui"),
     36             "username": os.environ["AUTH_USERNAME"] if strategy == "password" else os.environ.get("AUT
     37             "password": os.environ["AUTH_CLIENT_SECRET"],
     38             "scope": os.environ.get("AUTH_SCOPE"),
     39         },
     40         "stream-storage": {
     41             "schema": os.environ.get("STREAM_STORAGE_SCHEMA", default=auth_schema),
     42             "hostname": os.environ.get("STREAM_STORAGE_ENDPOINT", default=os.environ["AUTH_ENDPOINT"]
     43         },
     44         "platform-management": {
     45             "schema": os.environ.get("PLATFORM_MANAGEMENT_SCHEMA", auth_schema),
     46             "hostname": os.environ.get(
     47                 "PLATFORM_MANAGEMENT_ENDPOINT", default=os.environ["AUTH_ENDPOINT"] + "/api/platform-m
     48             ),
     49         },
     50     },
     51 }

File /usr/lib/python3.9/os.py:679, in _Environ.__getitem__(self, key)
    676     value = self._data[self.encodekey(key)]
    677 except KeyError:
    678     # raise KeyError with the original key value
--> 679     raise KeyError(key) from None
    680 return self.decodevalue(value)

KeyError: 'AUTH_ENDPOINT'
```

В SDK доступно несколько различных методов агрегации

```
[5]:
```

```
from platform_sdk import IntervalOperator

list(IntervalOperator)
```

```
[5]:
```

```
[<IntervalOperator.MIN: 'INTERVAL_MIN'>,
 <IntervalOperator.MAX: 'INTERVAL_MAX'>,
 <IntervalOperator.AVG: 'INTERVAL_AVG'>,
 <IntervalOperator.MEDIAN: 'INTERVAL_QUANTILE'>,
 <IntervalOperator.LAST: 'INTERVAL_LAST'>]
```

Читайте все сигналы в одном запросе, если это возможно

Если вы разбиваете свой список сигналов на несколько частей, общее время расчета отчета увеличится.

Проверьте максимальные и минимальные значения в сигналах.

Если вы запрашиваете значения за пределами возможных максимальных и минимальных значений, вы получите ошибку. Поэтому рекомендуем проверить максимальные и минимальные значения сигнала заранее.

```
[6]:
```

```
from datetime import timedelta
from platform_sdk import SignalReaderClient, IntervalOperator

signal_reader = SignalReaderClient()

min_dt = max(min_dt, signal_reader.get_min_dt([signal_pid]))
max_dt = min(max_dt, signal_reader.get_max_dt([signal_pid]))

df = signal_reader.read_without_custom_processors(
    min_dt=min_dt,
    max_dt=max_dt,
    public_ids=[signal_pid],
    granularity=timedelta(hours=1),
    interval_op=IntervalOperator.AVG,
)
df
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[6], line 4
      1 from datetime import timedelta
      2 from platform_sdk import SignalReaderClient, IntervalOperator
----> 4 signal_reader = SignalReaderClient()
      6 min_dt = max(min_dt, signal_reader.get_min_dt([signal_pid]))
      7 max_dt = min(max_dt, signal_reader.get_max_dt([signal_pid]))

File ~/work/platform_sdk/src/platform_sdk/integrations/stream_storage/signal/reader.py:82, in SignalReaderCli
     81 def __init__(self, auth: Optional[AuthService] = None, config: Optional[Configuration] = None) -> None
---> 82     super().__init__(auth, config)
     83     self._reader = SignalReaderAPI(auth=self.auth, config=self.config)
     84     self._ops = SignalsOps(auth=self.auth, config=self.config)

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/resource.py:81, in StreamStorageResource.__init_
     80 def __init__(self, auth: Optional[AuthService] = None, config: Optional[Configuration] = None) -> None
---> 81     super().__init__(auth, config)
     82     self._http_client = httpclient.HTTPClient(
     83         auth=self.auth,
     84         schema=self.config.config["integrations"]["stream-storage"]["schema"],
     85         hostname=self.config.config["integrations"]["stream-storage"]["hostname"],
     86     )
     87     self._rpc_client = rpc_client.RPCClient(
     88         auth=self.auth,
     89         schema=self.config.config["integrations"]["stream-storage"]["schema"],
     90         hostname=self.config.config["integrations"]["stream-storage"]["hostname"],
     91     )

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/resource.py:24, in AuthResource.__init__(self, a
     23 def __init__(self, auth: Optional[AuthService] = None, config: Optional[Configuration] = None):
---> 24     self._config = config or Configuration()
     25     self._auth = auth or AuthService(config=self.config)

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/config.py:16, in Configuration.__init__(self, da
     15 def __init__(self, data: Optional[dict] = None):
---> 16     self._data = data or self.load()

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/config.py:33, in Configuration.load(cls)
     26 auth_schema = os.environ.get("AUTH_SCHEMA", "https")
     27 strategy = os.environ.get("AUTH_STRATEGY", default="password")
     28 return {
     29     "integrations": {
     30         "auth": {
     31             "strategy": strategy,
     32             "schema": auth_schema,
```

```
---> 33             "hostname": os.environ["AUTH_ENDPOINT"],
     34             "realm": os.environ.get("APP_AUTH_REALM", default="platform"),
     35             "client_id": os.environ.get("AUTH_CLIENT_ID", default="ui"),
     36             "username": os.environ["AUTH_USERNAME"] if strategy == "password" else os.environ.get("AUT
     37             "password": os.environ["AUTH_CLIENT_SECRET"],
     38             "scope": os.environ.get("AUTH_SCOPE"),
     39         },
     40         "stream-storage": {
     41             "schema": os.environ.get("STREAM_STORAGE_SCHEMA", default=auth_schema),
     42             "hostname": os.environ.get("STREAM_STORAGE_ENDPOINT", default=os.environ["AUTH_ENDPOINT"]
     43         },
     44         "platform-management": {
     45             "schema": os.environ.get("PLATFORM_MANAGEMENT_SCHEMA", auth_schema),
     46             "hostname": os.environ.get(
     47                 "PLATFORM_MANAGEMENT_ENDPOINT", default=os.environ["AUTH_ENDPOINT"] + "/api/platform-m
     48             ),
     49         },
     50     },
     51 }

File /usr/lib/python3.9/os.py:679, in _Environ.__getitem__(self, key)
    676     value = self._data[self.encodekey(key)]
    677 except KeyError:
    678     # raise KeyError with the original key value
--> 679     raise KeyError(key) from None
    680 return self.decodevalue(value)

KeyError: 'AUTH_ENDPOINT'
```

3. Уровень логирования

Отчет Jupyter Notebook имеет тот же уровень логирования, что и приложение jupyter_report. Поэтому низкие уровни журнала приведут к появлению множества технической информации SDK.

```
[7]:
```

```
from datetime import timedelta
from platform_sdk import SignalReaderClient, IntervalOperator

signal_reader = SignalReaderClient()

df = signal_reader.read_without_custom_processors(
    min_dt=min_dt,
    max_dt=max_dt,
    public_ids=[signal_pid],
    granularity=timedelta(hours=1),
    interval_op=IntervalOperator.AVG,
)
df
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[7], line 4
      1 from datetime import timedelta
      2 from platform_sdk import SignalReaderClient, IntervalOperator
----> 4 signal_reader = SignalReaderClient()
      6 df = signal_reader.read_without_custom_processors(
      7     min_dt=min_dt,
      8     max_dt=max_dt,
   (...)
     11     interval_op=IntervalOperator.AVG,
     12 )
     13 df

File ~/work/platform_sdk/src/platform_sdk/integrations/stream_storage/signal/reader.py:82, in SignalReaderCli
```

```
     81 def __init__(self, auth: Optional[AuthService] = None, config: Optional[Configuration] = None) -> None
---> 82     super().__init__(auth, config)
     83     self._reader = SignalReaderAPI(auth=self.auth, config=self.config)
     84     self._ops = SignalsOps(auth=self.auth, config=self.config)

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/resource.py:81, in StreamStorageResource.__init_
     80 def __init__(self, auth: Optional[AuthService] = None, config: Optional[Configuration] = None) -> None
---> 81     super().__init__(auth, config)
     82     self._http_client = httpclient.HTTPClient(
     83         auth=self.auth,
     84         schema=self.config.config["integrations"]["stream-storage"]["schema"],
     85         hostname=self.config.config["integrations"]["stream-storage"]["hostname"],
     86     )
     87     self._rpc_client = rpc_client.RPCClient(
     88         auth=self.auth,
     89         schema=self.config.config["integrations"]["stream-storage"]["schema"],
     90         hostname=self.config.config["integrations"]["stream-storage"]["hostname"],
     91     )

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/resource.py:24, in AuthResource.__init__(self, a
     23 def __init__(self, auth: Optional[AuthService] = None, config: Optional[Configuration] = None):
---> 24     self._config = config or Configuration()
     25     self._auth = auth or AuthService(config=self.config)

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/config.py:16, in Configuration.__init__(self, da
     15 def __init__(self, data: Optional[dict] = None):
---> 16     self._data = data or self.load()

File ~/work/platform_sdk/src/platform_sdk/integrations/utils/config.py:33, in Configuration.load(cls)
     26 auth_schema = os.environ.get("AUTH_SCHEMA", "https")
     27 strategy = os.environ.get("AUTH_STRATEGY", default="password")
     28 return {
     29     "integrations": {
     30         "auth": {
     31             "strategy": strategy,
     32             "schema": auth_schema,
---> 33             "hostname": os.environ["AUTH_ENDPOINT"],
     34             "realm": os.environ.get("APP_AUTH_REALM", default="platform"),
     35             "client_id": os.environ.get("AUTH_CLIENT_ID", default="ui"),
     36             "username": os.environ["AUTH_USERNAME"] if strategy == "password" else os.environ.get("AUT
     37             "password": os.environ["AUTH_CLIENT_SECRET"],
     38             "scope": os.environ.get("AUTH_SCOPE"),
     39         },
     40         "stream-storage": {
     41             "schema": os.environ.get("STREAM_STORAGE_SCHEMA", default=auth_schema),
     42             "hostname": os.environ.get("STREAM_STORAGE_ENDPOINT", default=os.environ["AUTH_ENDPOINT"]
     43         },
     44         "platform-management": {
     45             "schema": os.environ.get("PLATFORM_MANAGEMENT_SCHEMA", auth_schema),
     46             "hostname": os.environ.get(
     47                 "PLATFORM_MANAGEMENT_ENDPOINT", default=os.environ["AUTH_ENDPOINT"] + "/api/platform-m
     48             ),
     49         },
     50     },
     51 }

File /usr/lib/python3.9/os.py:679, in _Environ.__getitem__(self, key)
    676     value = self._data[self.encodekey(key)]
    677 except KeyError:
    678     # raise KeyError with the original key value
--> 679     raise KeyError(key) from None
    680 return self.decodevalue(value)

KeyError: 'AUTH_ENDPOINT'
```

Если вы не хотите видеть эту информацию в выводе, установите высокие уровни логирования (`INFO` или `WARNING`) для всего приложения.

4. Вывод ячеек

Графики

Внутри блокнота вы можете использовать различные графики и диаграммы для обмена визуальной информацией.

```
[8]:
```

```
df.hist()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 1
----> 1 df.hist()

NameError: name 'df' is not defined
```

Pandas

Если вы хотите показать часть фрейма данных, убедитесь, что вы установили правильные параметры отображения

```
[9]:
```

```
pd.set_option("display.max_columns", 100)
pd.set_option("display.max_rows", 100)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 pd.set_option("display.max_columns", 100)
      2 pd.set_option("display.max_rows", 100)

NameError: name 'pd' is not defined
```

Matplotlib

Чтобы убрать предупреждения matplotlib при импорте, установите следующие параметры перед импортом

```
[10]:
```

```
import os

os.environ["MPLCONFIGDIR"] = "/tmp/"
```