Инструкция Go FusionLab

Введение

Go FusionLab — это модульная платформа для управления интернет-рекламой, где все настройки и контроль кампаний осуществляются через управляющие таблицы в базе данных. Такой подход обеспечивает централизованное управление, прозрачность и гибкость: пользователи могут оперативно менять параметры таргетинга, бюджеты, настройки сегментации и автоматизации без сложных технических действий. Управляющие таблицы связывают все функциональные модули системы, упрощая взаимодействие между ними.

Архитектура **Go FusionLab** построена на отдельных микросервисах, каждый из которых выполняет свои задачи: сбор данных, предиктивная аналитика, визуализация, передача событий на внешние платформы и другие. Это позволяет клиентам использовать только те модули, которые отвечают их потребностям, повышая эффективность управления рекламой. Такой подход делает систему гибкой, масштабируемой и идеально подходящей для решения задач различного уровня сложности.

Содержание

Как начать работать с системой

Для начала работы с Go FusionLab необходимо первым шагом получить доступ в базу данных postgres. Для этого на стороне Go Mobile создается учетная запись пользователя со всеми необходимыми ему правами доступа.

Также для непосредственной работы с платформой потребуется установка отдельной программы для работы с PostgreSQL. Подойдет любая программа для подобных целей, например DataGrip или pgAdmin.

Поскольку работа со многими частями платформы Go FusionLab связана с заполнением менеджерских и системных таблиц, а также постановкой задач для планировщиков и сервисов, пользователю потребуются навыки владения языком программирования SQL и понимание работы СУБД PostgreSQL.

К программному обеспечению ПК пользователя для работы с Go FusionLab требований не предъявляется. Рекомендуемая входящая/исходящая скорость соединения — от 512 кбит/с. К аппаратному обеспечению пользователя для работы с Go FusionLab требований не предъявляется.

Модуль сбора и обработки данных ("Гоу Коллекторы")

Что это?

Модуль «Гоу Коллекторы» автоматически собирает данные о рекламных кампаниях ежедневно, основываясь на параметрах, указанных в управляющих таблицах PostgreSQL. Эти таблицы задают источники данных, частоту обновлений, фильтры и ключи для сбора, обеспечивая гибкое управление процессом сбора. Собранные данные хранятся в высокопроизводительном хранилище ClickHouse, что позволяет эффективно обрабатывать большие объёмы информации и выполнять сложные аналитические запросы.

Как это работает?

Совокупность управляющих таблиц, в которых ставятся таски на сбор данных из рекламных кабинетов, а также таблиц, хранящих эти данные.

Сбор данных из Adjust

БД - `collectors_system`

Схема - `collector_adjust_pull_api`

Перед началом работы необходимо получить API token на сайте adjust.com - Account settings - API token, а затем добавить этот токен в таблицу accounts схемы collector_adjust_pull_api в PostgreSQL.

В данной схеме существуют два вида тасков: events и report.

Таск вида events ставится 1 раз для каждого аккаунта и собирает список событий в таблицу apps_events схемы collector_adjust_pull_api в ClickHouse.

Описание полей:

`name` = events,

`group_name` = default,

`account_id` = id аккаунта из таблицы accounts.

`status` не заполняется

поле `id` оставляем по умолчанию.

Остальные поля достаточно заполнить нулевыми/пустыми значениями, они будут проигнорированы.

`last_run_date` отобразит последнюю дату сбора событий.

`comment` - произвольное описание таска, заполнять не обязательно.

Таск вида report собирает статистику ежедневно.

Описание полей:

`name` = report,

`group_name` = default,

`status` не заполняем,

`id` оставляем по умолчанию.

`clickhouse_id` - выбираем сами, значение будет записано как task_id в кликхаусе.

`metrics` - метрики

`dimensions` - разбивки

`filters` - фильтры

`period` - разбивка по дням/часам, может быть строго day или hour.

`last_run_date` - начальная дата сбора, а также дата последнего сбора коллектором.

`comment` - произвольное описание таска.

Сбор данных из Apple search ads

БД - `collectors_system`

Схема - `collector_apple_search_ads`

Перед постановкой таска, нужно заполнить данные аккаунта в таблице collectors_system > collector_apple_search_ads > task_params

Описание полей:

`id` - не трогаем

`certificate` - берется из рекламного кабинета

`client_id` - берется из рекламного кабинета

`key_id` - берется из рекламного кабинета

Чтобы поставить таск, необходимо перейти в PostgreSQL collectors_system > collector_apple_search_ads > task.

Описание полей:

`id` - не трогаем

`name` - default

`group_name` - default

`status` - не трогаем

`param_id` - id из таблицы collectors_system > collector_apple_search_ads > task_params

`date_start` - дата начала сбора.

`clickhouse_id` - следующее по порядку число от последнего созданного

`org_id` - берется из рекламного кабинета

`description` - наименование организации

`created_at` - не трогаем

`updated_at` - не трогаем

Сбор данных из Appmetrica

БД - `collectors_system`

Схема - `collector_appmetrica`

Описание полей в управляющей таблице и как завести таск:

Таблица collectors_system > collector_appmetrica > task.:

`id` - айди таска, оставляем default (не заполняем)

`clickhouse_id` - с таким task_id данные будут писаться в кликхаус (совместимость со старыми данными)

`name` - тип таска, возможные значения (default)

`group_name` - отвечает за время запуска.

`status` - Техническое поле. Возможные значения(complete, error, in progress)

`app_id` - id приложения из client_apps

`type` - тип отчета в аппметрике. Сейчас поддерживаются: installs, events, deeplinks.

`fields` - перечисление полей через запятую. Брать из документации Logs API.

`event_name` - перечисление собираемых событий через запятую. Актуально не для всех типов отчета. Берется в кабинете или у клиента.

`last_run_date` - последняя дата сбора и начальная дата сбора. Также автоматически собирает докаты за последние 8 дней от текущего времени.

`created_at` - Дата создания таска

`updated_at` - Дата обновления таска

Таблица collectors_system > collector_appmetrica > accounts:

`id` - id аккаунта

`name` - имя аккаунта (заносите имя клиента)

`token` - токен

`token_expire` - Дата истечения токена

`created_at` - Дата создания таска

`updated_at` - Дата обновления таска

Таблица collectors_system > collector_appmetrica > client_apps:

`id` - id записи client_apps

`bundle` - наименование бандла (информационное поле, в коллекторе не задействовано)

`name` - имя приложения (в кабинете рядом с id)

`application_id` - id приложения (в кабинете)

`account_id` - id аккаунта из таблицы accounts

`created_at` - Дата создания таска

`updated_at` - Дата обновления таска

Таблица collectors_system > collector_appmetrica > task_history:

`id` - уникальный айди записи

`task_id` - айди таска по которому смотрим ошибку и лог

`status` - статус таска, возможные значения (error, complete)

`progress` - техническое поле

`error` - техническое описание ошибки

`tracing_id` - техническое поле

`logs` - лог ошибки

`created_at` - дата создания записи

`updated_at` - дата обновления записи

Сбор данных из Asana

БД - `collectors_system`

Схема - `collector_asana`

Описание полей в управляющей таблице и как завести таск:

Таблица `collectors_system.collector_asana.account`

Таблица `collectors_system.collector_asana.task`

`id` - айди таска, оставляем default

`name` - тип таска

`group_name` - отвечает за время запуска.

`status` - статусы выполнения, не трогаем, заполняется само. Возможные значения(`complete`, `error`, `in progress`)

`account_id` - параметры из таблички `collectors_system.collector_asana.account`

`project_id` - айди проекта в асане

`date_start` - дата начала сбора (при первом запуске используется для сбора исторических данных)

`created_at` - дата создания записи

`updated_at` - дата обновления записи

Сбор данных из Bigo ads

БД - `collectors_system`

Схема - `collector_bigo_ads`

Для подключения нового кабинета к сбору статистики коллектором, нужно:

1/ Запросить у поддержки площадки: access token, refresh token и пару client id, secret key для подключаемого кабинета.

2/ Перейти в постгресс **collector_system > collector_bigo_ads > таблица task_params** и, добавив строку, заполнить:

`id` - не трогаем, заполняется автоматически

`account_name` - логин кабинета с которым заходите на площадку

`refresh token` - вставляете сюда значение присланное поддержкой

`access token` - вставляете сюда значение присланное поддержкой

`expires at` - Выданные токены актуальны 24ч.

`client id` - вставляете сюда значение присланное поддержкой

`client secret` - вставляете сюда значение присланное поддержкой

`created at` - не трогаем, заполняется автоматически

`updated at` - не трогаем, заполняется автоматически

3/ Перейти в постгресс **collector_system > collector_bigo_ads > таблица task** и добавив строку заполнить:

`id` - не трогаем, заполняется автоматически

`clickhouse_id` - айдишник, под которым таск будет храниться в кх, заполняйте по порядку (+1 от предыдущего)

`name` и `group_name` на данный момент дублируют друг друга. понадобятся вам для заведения очереди в менеджере коллекторов

`status` - техническое поле, заполняется автоматически

`param_id` - айди параметра (т.е. кабинета), который заполняли в табличке collectors_system.collector_bigo_ads.task_params

`collect from` - дата с которой собираться статистике

`timezone` - таймзона по utc. то есть чтобы поставить в поясе москвы - ставьте 3

`last_run_date` - заполняется автоматически

`created at` - заполняется автоматически, дата создания таска

`updated at` - заполняется автоматически, дата обновления/последней отработки таска

4/ Перейти в КХ collector bigo_ads > reports Тут будет собираться и храниться статистика.

Собираемые коллектором поля:

`id` - внутренний айди записи

`task id` - айди таска по которому собирается стата (`clickhouse_id` из пункта 3)

`date`

`ad_id`

`ad_name`

`adset_id`

`adset_name`

`advertiser_id`

`campaign_id`

`campaign_name`

`click`

`country`

`impression`

`tgt_pkg_name` - promotional app package name

`total cost` - в коллекторе указан как 1% от 100% в интерфейсе. Тобишь это центы. Для сверки нужно число из коллеткора умножать на 100% или 0,01

`created_at` - дата добавления записи

Сбор данных из Fraudscore

БД - `collectors_system`

Схема - `collector_fraudscore`

Для постановки таска необходимо первым делам заполнить таблицу collectors_system.collector_fraudscore.account

`id` - айди записи

`channel_id` - берется из рекламного кабинета

`key` - берется из рекламного кабинета

`created_at` - дата добавления записи

Затем заполняется collectors_system.collector_fraudscore.task

`id` - айди записи

`name` - тип таска, возможные значения (default)

`group_name` - отвечает за время запуска.

`status` - статус таска

`account_id` - айди записи из collectors_system.collector_fraudscore.account

`last_run_date` - дата последней записи

`created_at` - дата добавления записи

`updated_at` - дата обновления записи

Сбор данных из Getintent

БД - `collectors_system`

Схема - `collector_getintent`

Для постановки таска необходимо первым делам заполнить таблицу collectors_system.collector_getintent.task_params

`id` - айди записи

`account_name` - берется из рекламного кабинета

`token` - берется из рекламного кабинета

`created_at` - дата добавления записи

`updated_at` - дата обновления записи

Затем заполняется collectors_system.collector_getintent.task

`id` - айди записи

`name` - тип таска, возможные значения (default)

`group_name` - отвечает за время запуска.

`status` - статус таска

`param_id` - айди записи из collectors_system.collector_getintent.task_params

`date_start` - дата начала сбора

`dataset_name` - название датасета

`keys` - ключи

`values` - значения

`clickhouse_id` - айди в базе clickhouse

`created_at` - дата добавления записи

`updated_at` - дата обновления записи

Сбор данных из Gonet

БД - `collectors_system`

Схема - `collector_gonet`

Для постановки таска необходимо заполнить таблицу collectors_system.collector_gonet.task

`id` - айди таска, оставляем default

`name` - тип таска(см.ниже)

`group_name` - отвечает за время запуска.

`status` - статусы выполнения, не трогаем, заполняется само. Возможные значения(`complete`, `error`, `in progress`)

`param_id` - параметры из таблички `task_params`

`client_id` - клиент, можно увидеть в табличке `clients`

`date_start` - дата начала сбора данных

`clickhouse_id` - айди в базе clickhouse

Типы тасков:

`today` - заводится для тасков, которые будут выполнятся сегодня

`yesterday` - заводится для тасков, которые будут выполнятся за вчера

Таблица task_params:

`url` - ссылка на дешборд

`comment` - описание в свободной форме

Сбор данных из Google_play

БД - `collectors_system`

Схема - `collector_google_play`

Коллектор собирает данные из Google Play и сохраняет их в базу данных.

Для запроса по API необходимо предоставить следующие данные:

- PrivateKey - ключ доступа к API Google Play
- BucketName - место откуда скачивать отчет
- ObjectName - путь до отчета

Настройка в базе коллектора:

Создать параметры для таска в таблице collectors_system.collector_google_play.task_params

`id` - айди таска, оставляем default

`name` - тип таска(см.ниже)

`group_name` - отвечает за время запуска.

`status` - статусы выполнения, не трогаем, заполняется само. Возможные значения(`complete`, `error`, `in progress`)

`param_id` - параметры из таблички `task_params`

`report type` -тип репорта

`app_id` - айди приложения

`group_by` - группировка

`created_at` - дата создания записи

`updated_at` - дата обновления записи

Создать таск в Postgres - collectors_system.collector_google_play.task.

`Name` - default.

`Group` на выбор, по умолчанию default.

Сбор данных из Hybrid (Hybe)

БД - `collectors_system`

Схема - `collector_hybrid`

Перед постановкой таска необходимо получить от площадки пару, id и секретный ключ клиента.

datagrip > postgresql > collector systems > collector_hybrid > tables > task_params > Добавляем строку и вставляем в колонки:

`id` - default

`access_token` - ничего (оставляем пустым) ((если ваша версия не дает закоммитить пустой, попробуйте оставить пробел, но должно быть ок пустым))

`expired_at` - меньше текущей даты как минимум на 10+ минут

`created_at` - default или ок, если будет текущая дата

`updated_at` - default или ок, если будет текущая дата

После коммита строки, берем ее id

и идем в таблицу collectors_system.collector_hybrid.task, там добавляем новый таск:

`id` - default

`name` - default

`group_name` - default

`status` - не трогаем

`param_id` - вставляем id который брали на предыдущим шаге в таблице task_params

`date_start` - ставим дату сегодняшнюю

`email` - указываем email с помощью которого входим в кабинет

`client_id` - тот что получили от площадки

`client_secret` - тот что получили от площадки

`created_at` - не трогаем

`updated_at` - не трогаем

Сбор данных из Liftoff

БД - `collectors_system`

Схема - `collector_liftoff`

В данном коллекторе формируется два отчета с разными группировками, в две таблицы в clickhouse.

Первая: `"apps", "campaigns", "creatives", "publisher"` (collector_liftoff.report_by_creatives)

Вторая: `"apps", "campaigns", "ad_format"` (collector_liftoff.report_by_ad_format)

Описание полей в управляющей таблице и как завести таск:

Таблица collectors_system.collector_liftoff.task:

`id` - айди таска, оставляем default (не заполняем)

`name` - default

`group_name` - default. (также можно отключить таск, введя inactive)

`status` - статусы выполнения, заполняется само. Возможные значения (`complete`, `error`, `in progress`)

`account_id` - аккаунт из таблички `account`

`date_start` - дата начала сбора

`last_run_date` - дата последнего успешного завершения сбора

`created_at` -дата создания таска

`updated_at` -дата последнего апдейта таска

`timezone` - таймзона, заполняется в соответствии с кабинетом для избегания расхождений

Таблица Task_history:

`id` - уникальный айди записи

`task_id` - айди таска по которому смотрим ошибку и лог

`status` - статус таска, возможные значения(error, complete)

`progress` - прогресс

`error` - техническое описание ошибки

`tracing_id` - поле для разработчиков, для отслеживания логов

`logs` - лог ошибки

`created_at` - дата создания записи

`updated_at` - дата обновления записи

Заводим таск:

1/ пг > collectors_system > collector_liftoff > tables > `account`

2/ вносим инфу клиенте и об `api_key` и `api_secret` (узнать их можно у площадки)

3/ Далее заходим в таблицу `task` и заполняем поля соответственно нужным данным (и данным клиента - `account_id`)

Сбор данных из Mi ads

БД - `collectors_system`

Схема - `collector_mi_ads`

Запрашиваем у поддержки mi ads следующую информацию:

appId + appKey и заодно account id.

Создаем в управляющей таблице collectors_system > accounts новую строку и заполняем:

`id` - account id

`name` - имя клиента

`appid` - то что отдала поддержка

`appkey` - то что отдала поддержка

`created_at` - не трогаем, заполняется автоматически, дата создания акка

`updated_at` - не трогаем, заполняется автоматически, дата обновления акка

`access_token` - не трогаем, заполняется автоматически

`expired_at` - не трогаем, заполняется автоматически

`refresh_token` - не трогаем, заполняется автоматически

`refresh_expired_at` - не трогаем, заполняется автоматически

Дальше заводим task

`id` - не трогаем, заполняется автоматически

`clickhouse_id` - айди, под которым данные будут храниться в кх (заполняйте по порядку, +1 от предыдущего)

`name` и `group_name` - на данный момент дублируют друг друга. понадобятся вам для заведения очереди в менеджере коллекторов (там можно создать очередь для конкретной группы тасков)

`status` - не трогаем, тех поле

`comment` - имя клиента

`account_id` - ставите тот же, что в поле id в таблице collectors_system.collector_mi_ads.accounts

`owner_email` - почта кто из аналитиков ставит таск

`last_run_date` - заполняется автоматически

`created at` - заполняется автоматически, дата создания таска

`updated at` - заполняется автоматически, дата обновления/последней отработки таска

4/ проверить очередь в менеджере коллекторов, скорее всего данные соберутся завтра утром

5/ в кх заполняется таблица collector_mi_ads.report

табличка collector_mi_ads.names тоже заполняется автоматически, но вряд ли она вам сама по себе понадобится.

Логика следующая:

Сбор данных из Mintegral

БД - `collectors_system`

Схема - `collector_mintegral`

Описание полей.

`Date -` дата

`Offer Id -` id предложения

`Offer Uuid -` уникальное имя предложения генерируемое автоматически

`Offer Name -` имя предложения

`Campaign id -` id кампании

`Campaign name -` имя кампании

`Creative id -` id креатива

`Creative Name -` имя креатива

`ad type -` тип креатива

`Location -` страна продвижения предложений

`Sub id -` id партнера

`Package name(bundle id) -` Запрос данных по идентификатору пакета Android или идентификатору магазина приложений iOS.

`app name -` Имя приложения, соответствующее имени вашего пакета(bundle id).

`Currency -` валюта (значение по умолчанию USD)

`Impression`

`Click`

`Conversion`

`Ecpm`

`Cpc`

`Ctr`

`Cvr`

`Ivr`

`Spend -` расход

`task id -` сервисные данные о привязке к task id

`created at -` сервисные данные о создании записи

Описание работы с коллектором.

Для запуска таска необходимо заполнить следующие таблицы в базе `postgres.collector_system.collector_mintegral:`

`1. task_params - (access_key, token, client_name, login)`

`access_key -` ключ для авторизации в API (берем из кабинета)

`token(api_key) -` ключ для авторизации в API (берем из кабинета)

`client_name, login -` можно указать любой, не участвует в сборе данных. (рекомендуем заполнять по данным от кабинета, чтобы легче понимать, где какой ключ)

`2. task -` (name, group_name, param_id, report_type, dimension_type, date_start)

`name` - заполняем всегда default

`group` - та же группа что и в `postgres.collector_system.manager.queue.your_task.group_name` (по этой группе менеджер понимает какие таски запускать)

`param_id` - ссылка на аккаунт из task_params

`report_type` - отвечает за тип отчета, доступные типы отчета (adv_performance)

`dimension_type` - доп параметры у отчета. Доступные параметры `(package, creative).` В варианте `creative , dimensions` в запросе следующие: `"Offer", "Campaign", "Creative", "AdType", "Location" .` В варианте `package , dimensions` в запросе следующие: `"Offer", "Campaign", "AdType", "Sub", "Package", "Location".` В зависимости от выбранного параметра, выгружаем в таблицу `clickhouse.collector_mintegral.adv_performance_report_creative` или `clickhouse.collector_mintegral.adv_performance_report_package.` Также, если у нас стоит параметр package, мы дополнительно выгружаем информацию по `target_apps` в таблицу `clickhouse.collector_mintegral.target_apps.` А если стоит параметр `creative` , мы дополнительно выгружаем информацию по `campaigns` в таблицу `clickhouse.collector_mintegral.campaigns.`

`date_start` - дата начала сбора отчетов. (Важно! Если планируется сбор отчета больше чем 1-н месяц, предупредить разработчика об этом)

Создать очередь в менеджере postgres.collector_system.manager.queue.

Сбор данных из RevX

БД - `collectors_system`

Схема - `collector_revx`

Описание полей в управляющей таблице и как завести таск:

Таблица `task`:

`id` - айди таска, оставляем default

`name` - default

`group_name` - группа таска (от нее зависит время запуска в менеджере коллекторов)

`status` - статусы выполнения. Возможные значения: complete, error, in progress.

`account_id` - аккаунт из таблички account

`date_start` - дата начала сбора

`last_run_date` - дата последнего успешного завершения сбора (не заполняем)

`created_at` -дата создания таска (не заполняем)

`updated_at` -дата последнего апдейта таска (не заполняем)

Таблица task_history (в ней ничего заполнять не трубуется):

`id` - уникальный айди записи

`task_id` - айди таска по которому смотрим ошибку и лог

`status` - статус таска, возможные значения(error, complete)

`progress` - пока не несет информативной инфы. Что-то вроде плесхолдера

`error` - техническое описание ошибки

`tracing_id` - поле для разработчиков, для отслеживания логов

`logs` - лог ошибки

`created_at` - дата создания записи

`updated_at` - дата обновления записи

Заводим таск:

1/ пг > collectors_system > collector_revx > tables > account

2/ Вносим информацию об аккаунте, а именно username и password. Как их узнать? Посмотреть в кабинете либо посмотреть в пг > collector > collector_revx > tables > tasks. Действующие аккаунты имеют параметр active=true

3/ Далее заходим в таблицу task и заполняем поля соответственно нужными данными.

4/ Заводим очередь в менеджере коллекторов, если требуется.

Сбор данных из Yandex Cloud

БД - `collectors_system`

Схема - `collector_yandex_cloud`

Описание полей в управляющей таблице и как завести таск:

Таблица task:

`id` - айди таска, оставляем default (не заполняем)

`name` - тип таска, пока только один тип тут требуется (default)

`group_name` - отвечает за время запуска.

`status` - статусы выполнения, не трогаем, заполняется само. Возможные значения: complete, error, in progress.

`param_id` - id параметров соответствующих таблице task_params В пг.

`date_start` - Дата начала сбора статы для таска

`bucket_name` - наименование "папки" в yandex cloud в которую собираются отчеты из data locker по вашему клиенту. (Инструкция по его созданию ниже)

`page` - вспомогательное поле, требуется, чтобы понимать на какой странице мы сейчас находимся (собираем отчет), страницы формируются по размеру = 1гб, на одной странице может быть N отчетов, размер которых не превышает 1гб, сделано, чтобы мы не падали по лимитам памяти (оставляем default значение, либо -1 сами проставляем)

`total_pages` - вспомогательное поле для пагинации, требуется, чтобы понимать сколько страниц всего за дату сбора отчета и для проверки что page != total_page. (оставляем default значение, либо -1 сами проставляем)

`created_at` - Дата создания таска (не заполняем)

`updated_at` - Дата последнего обновления таска (не заполняем)

Task_params:

`id` - id параметров, не заполняем. Требуется для таблицы task поля param_id , чтобы обращаться к апи клауда через эти креды.

`oauth_token` - Сформированный токен из Яклауда

`Service_account_id` - id сервисного аккаунта из Яклауда

`created_at` - Дата создания таска (не заполняем)

`updated_at` - Дата последнего обновления таска (не заполняем)

Таблица Task_history:

`id` - уникальный айди записи

`task_id` - айди таска по которому смотрим ошибку и лог

`status` - статус таска, возможные значения(error, complete)

`progress` - пока не несет информативной инфы. Что-то вроде плейсхолдера

`error` - техническое описание ошибки

`tracing_id` - поле для разработчиков, для отслеживания логов

`logs` - лог ошибки

`created_at` - дата создания записи

`updated_at` - дата обновления записи

Сбор данных из Yandex Direct

БД - `collectors_system`

Схема - `collector_yandex_direct`

Описание полей в управляющей таблице и как завести таск:

Таблица task:

`id` - айди таска, оставляем default (не заполняем)

`name` - тип таска(см.ниже)

`group_name` - отвечает за время запуска, сейчас сгруппировано по типу, дублирует значения колонки `name`. Также можно отключить таск, введя `inactive`

`status` - статусы выполнения, не трогаем, заполняется само. Возможные значения(`complete`, `error`, `in progress`)

`account_id` - аккаунт из таблички `api_config`

`client_id` - клиент, можно увидеть в табличке `clients` , при заведении нового account_id, чтобы clients туда собрались ставьте тип таска client и не указывайте client_id.

`last_fetch` - дата последнего успешного завершения сбора (не заполняем или ставим любую даты из прошлого)

Типы тасков:

`client` - заводится один раз для каждого аккаунта, тянет новых клиентов, если токену это позволено. Заполняет таблицу `clients` в постгре и `client` в кликхаусе

`info` - заводится автоматически при обнаружении нового клиента. Заполняет таблицу `campaigns` в постгре и таблицы `campaign`, `ad_group`, `ad`, `ad_image`, `creative` в кликхаусе

`uac` - заводится автоматически при обнаружении нового клиента. Заполняет таблицу `campaign_report` в кх для кампаний созданных в мастере кампаний ядиректа + экспертных кампаний. Фильтровать нужно самим т.к. уже нет жесткого отличительного признака который был в старом коллекторе(присутствие uac в нейминге кампании).

`geo_uac` - отчет с разбивкой по Geo для кампаний созданных в мастере кампаний ядиректа. Заводится вручную для каждого клиента. Заполянет таблицу `geo_campaign_report` для мастер кампаний + экспертных кампаний в кх. Фильтровать нужно самим т.к. уже нет жесткого отличительного признака который был в старом коллекторе(присутствие uac в нейминге кампании)

`cost` - стандартный отчет `AD_PERFORMMANCE_REPORT`, заводится автоматически при обнаружении нового клиента. Заполняет таблицу `report` в кликхаусе

`geo` - отчет с разбивкой по `targeting_location_id`. Заводится вручную. Заполняет таблицу `geo_report` в кликхаусе. (Осторожно, очень громоздкий, может вылетать на некоторых клиентах на стороне яндекса)

`search` - отчет с разбивкой по `criteria`. Заводится вручную. Заполняет таблицу `search_report` в кликхаусе. (Осторожно, невероятно громоздкий, может вылетать на некоторых клиентах на стороне яндекса)

`rapfr` - отчет `REACH_AND_FREQUENCY_REPORT`. Заводится вручную. Скорее всего не работает на стороне яндекса (или у наших клиентов нет статы для него).

Примеры всех тасков кроме `rapfr` есть в управляющей таблице.

Таблица Task_history:

`id` - уникальный айди записи

`task_id` - айди таска по которому смотрим ошибку и лог

`status` - статус таска, возможные значения(error, complete)

`progress` - пока не несет информативной инфы. Что-то вроде плесхолдера

`error` - техническое описание ошибки

`tracing_id` - поле для разработчиков, для отслеживания логов

`logs` - лог ошибки

`created_at` - дата создания записи

`updated_at` - дата обновления записи

Постановка очередей для сбора данных

Менеджер коллекторов создан для упрощения работы со всем массивом коллекторов через очереди и таски.

База данных в пг collectors_system. Каждый отдельный коллектор содержит данные в своей схеме в базе collector_system. Управляющие очереди лежат в схеме manager

Зоны ответственности:

Менеджер:

- контроль за очередями;
- вызов коллекторов по расписанию;
- вызов коллекторов вне расписания (в данный момент частично имплементировано);
- обновления базы данных управляющих таблиц и схем;
- сохранять историю вызовов а также ошибки тасков и очередей.

Коллектор:

- содержит логику для работы воркера;
- слушает и принимает таски
- возвращает принятый стандартом интерфейс ответа, ошибки или результата таска.

Очереди

`id` - проставляется автоматически uuid

`name` - Имя коллеткора (схемы) если указать несуществующую схему будет ошибка в истории

`status` - состояние очереди PENDING (ожидает вызова) ACTIVE (выполняется)

если очередь выполняется ее ручной вызов невозможен

`is_active` - выкл/вкл очередь

`group` - имя группы тасков которая будет вызывать очередь

`cron_expression` - когда вызывать очередь (проверить cron)

Таски

`id` - проставляется автоматически uuid

`group_name` - имя очереди которая вызывает таск

История вызова очередей

Храниться в таблице manager.queue_history. история содержит дату и время а также статус вызванной очереди

статус ERROR - все таски в очереди выполнились с ошибкой

статус COMPLETE очередь выполнена успешно

Истрия вызова тасков

в таблице task_history отдельно в каждом коллекторе (схеме). Хранит ошибки и логи каждого отдельно таска

в историю записываются данные сразу после завершения таска

статус COMPLETE таск выполнился и вернул успешный результат

статус ERROR таск вернул ошибку которую можно посмотреть в поле error

Добавить очередь

1. убедитесь что коллектор добавлен в систему и есть созданная группа тасков с полем group_name равное полю group в очереди
2. добавьте очередь в схему менеджер с корректным cron_expression
3. ждите выполнения и следите за статусом очереди

Добавление тасков

1. чтобы добавить таск в коллектор найдите нужную схему в collectors_system,
2. сделайте insert в таблицу task, укажите нужные поля параметров и также группу таска если хотите привязать его к очереди
3. в данный момент менеджер вызовет таск сразу при добавлении (в будущем такая механика может поменяться)

Модуль transmission service

Что это такое?

Это сервис для формирования аудиторий для подрядчиков. С помощью сервиса подрядчик сможет самостоятельно скачивать актуальную аудиторию, сформированную SQL запросом

Как найти?

1. Технические таблицы в постгре в схеме `services` → `transmission_service`
2. Таблицы с данным в кликхаусе в transmission_service

Общий алгоритм

1/ Аналитик заполняет таблицу task (все поля, кроме

`Id, API_Link, Token, has_error last_time_collected, updated_at, created_at, limit_requests, status`

важно: нужно обозначать названия колонок, которые выводим

2/Алгоритм создаётся таблицу в кликхаусе и называет {{task_id}} + '_' + {{partner_name}}, запускает SQL-запрос от аналитиков и укладывает данные в таблицу

3/ Алгоритм прописывает в постгре в

`last_time_collected`

дату и время запуска SQL

4/ Алгоритм прописывает в постгре в

`API_Link`

ссылку, которую можно передать партнёру для выкачивания данных

5/ Аналитики передают ссылку партнёру

После идут повторяющиеся действия:

.../ Партнёр выкачивает данные

.../ Алгоритм фиксирует выкачивание данных в таблице

`request_history`

.../ Алгоритм каждый день запускает sql-запрос, обрабатывает данные вместе с данными в табличке и записывает результат в таблицу

Про таблицу tasks

Поля, которые заполняет пользователь:

1. partner_name - вводится пользователем (оно входит в название таблицы в кликхаусе, а также после добавляется в ссылку, по которой партнёр будет выкачивать аудиторию). Максимальное количество - 255 символов. Можно заполнять:
    2. Буквы английского алфавита: a-z , A-Z.
    3. Цифрами: 0-9.
    4. Специальными символами: -, _, ., !, ~, *, ', (,). (upd нельзя использовать в name '-')
5. creator - вводится пользователем (внутреннее информативное поле)
6. is_active - работает ли таск или нет
7. SQL_request - запрос, на основе которой формируется таблица в кликхаусе. **Важно**, чтобы даты стояли динамические, тогда каждый день будет формироваться актуальная аудитория. Также если ссылаемся на старые таблицы (например, collectors_push_api.appsflyer_integrated_push_api_**old**), то **важно** убедиться, что используются свежие креды для доступа к ним (особенно актуально, если используете запрос сотрудника, который больше не работает в компании)
8. whitelist_active - включено ли ограничение по ip-адресам
    9. Это позволит ограничить скачивание не только сложной ссылкой с токеном, но и IP адресом. В результате, если хакеры получат ссылку, то смогут скачать только подсоединившись к сети с таким же IP-адресом (а это почти нереально). Механика: нужно спросить у партнёра IP адрес и объяснить, зачем это нам. Если партнёр отказывается передавать нам свой IP, то выключать вайтлист
10. IP_whitelist - Массив IP-адресов (есл whitelist_active = true)
11. audience_transmission_share (от 0 до 1) - Доля аудитории, передаваемой площадке (используется для тестов инкрементальности). По этой доли будет проставляться status=1
12. updated_by_user - если true, то произойдёт моментальный запуск таска

Про таблицу в кликхаусе

Под каждый таск создаётся отдельная таблица в кликхаусе в схеме

`transmission_service`

Таблица состоит из 4 полей:

1. id - id'шник пользователя

2. date_add - дата, когда id'шник попал в базу
3. date_gone - дата, когда id'шник перестал передаваться партнёру (так как его не было в результате последнего запуска SQL-запроса)
4. status - для тестов инкрементальности. 0 - оставляем id у себя и не передаём подрядчику. 1 - передаём подрядчику, если пустая date_gone

Важные поинты:

1. Подрядчику передаются только id'шники пользователей (пример выгрузки аттач)
2. Подрядчику передаются id'шники, у которых date_gone пустое и status=1
3. Сервис ежедневно запускается в 11:30
4. Передавать партнёру нужно только ссылку api_link, дополнительно токен передавать не нужно
5. Стоит ограничение на запросы по ссылке (чтобы партнёры не отправляли очень много запросов, в результате чего у нас бы упали сервера). Если партнёры будут жаловаться, то пишите - расширим лимиты

Визуализация для оптимизации рекламных кампаний с помощью дешбордов Datalens.

Дашборды **Datalens** предоставляют удобный инструмент для мониторинга и оптимизации рекламных кампаний в режиме реального времени. Они визуализируют ключевые метрики (CTR, CPC, ROI, CR и другие), предоставляя пользователю детализированную аналитику по эффективности различных каналов и аудиторий. Интерактивные графики и фильтры позволяют быстро находить проблемные точки, выявлять успешные стратегии и принимать обоснованные решения по корректировке кампаний.

Гибкая настройка дашбордов позволяет адаптировать их под конкретные потребности клиента: от общего обзора эффективности до детальной сегментации данных по регионам, времени или аудиториям. Такой подход упрощает анализ данных и делает управление кампаниями более наглядным и эффективным, помогая достичь максимального ROI.

ML-функционал

Для более точной работы с рекламными кампаниями в функционал платформу также заложены алгоритмы машинного обучения, включающие в себя:

- прогнозирование кумулятивного дохода когорты,
- предсказание коэффициента конверсии (CR),
- кластеризация пользователей,
- прогноз повторных покупок.

Данные, полученные на этапе “Гоу Коллекторы” обрабатываются ML-алгоритмами для создания прогнозов и расширенной аналитики: система предсказывает кумулятивный доход, коэффициент конверсии, вероятность повторных покупок и сегментирует пользователей, что позволяет точнее нацеливать кампании.

Описание проектов

Прогнозирование кумулятивного дохода когорты

Алгоритм используется для оценки кумулятивного revenue когорты пользователей на конец месяца. Рассмотрим календарный месяц (30 дней). В каждый i-й день этого месяца:

n_i пользователей устанавливают приложение и совершают события. Для них этот день - нулевой.

Все пользователи, которые установили с начала когорты в предыдущие дни, также совершают события в приложении

Цель - имея данные о первых M днях когорты (события пользователей, которые устанавливают каждый день), предсказать, сколько revenue принесут суммарно все пользователи на конец когорты. Например:

Пользователи, которые установили в первый день месяца, будут суммарно совершать события в течение всего месяца, и принесут revenue за весь месяц (до конца когорты)

Пользователи, которые установили в предпоследний день месяца, принесут revenue за один календарный день (до конца когорты)

Проект используется для более точной предварительной оценки результатов по заведению рекламной кампании, предлагая возможность для корректировки стратегии до конца когорты в целях изменения результата.

Описание алгоритма

Входные данные:

X таблиц данных по уже завершившимся когортам в прошлом.

N - на какой по счету день когорты мы делаем предсказание о кумулятивном revenue на конец когорты.

Каждая таблица отражает статистику когорты в разрезе по дням. Одна строка - один день. Столбцы:

month - календарный месяц (1-12)

spend - потраченный бюджет РК на привлечение пользователей в этот день

shows - количество показов в этот день

clicks - количество кликов по показанной рекламе в этот день

installs - количество установок в этот день

avg_ctit - среднее время от клика до установки в этот день

median_ctit - медианное время от клика до установки в этот день

search_per_day - количество поисковых результатов в день

revenue_d0, revenue_d1, revenue_d2, ...revenue_d30 - revenue для пользователей, установивших у дней назад. Так, для пользователей с установками на 30 день когорты будет заполнен только столбец revenue_d0. Для тех, кто установил в первый день когорты, будут заполнены все столбцы вплоть до revenue_d30.

Далее:

Обработка данных - заполнение пустых полей, обрезание столбцов revenue_di в зависимости от параметра N.

Энкодинг данных - кодировка всех значений в численный вид и нормализация.

Формирование вектора признаков когорты. Мы считаем два способа и выбираем лучший из них.

Векторы для каждого дня конкатенируются в один суммарный вектор (длина = n признаков * N)

Векторы для каждого дня усредняются в один средний вектор (длина = n признаков)

Формирование таргета - подсчет кумулятивного revenue для каждого вектора когорты

Обучение алгоритма. Мы обучаем две модели и берем лучшую из них.

Модель кластеризации - первый способ. Определяет расстояние между когортами, насколько когорты похожи друг на друга и насколько различаются

Модель регрессии - обучается на множестве "признаки"-"таргет", предсказывает кумулятивное revenue по вектору признаков когорты.

Тестирование алгоритмов на новых векторах когорт, ранее не используемых при обучении

Кластеризация - определение топ-10 ближайших соседей для новой когорты, подсчет предсказания через взвешенное усреднение кумулятивного revenue соседей

Регрессия - предсказание revenue напрямую по признакам новой когорты

Построение графиков метрик и ошибок на всех алгоритмах, в зависимости от того сколько N дней нам известно на момент предсказания.

Инструкция к применению (инферейс):

1. Предоставить данные о первых N днях когорты
2. Передать данные ML-разработчику
3. ML разработчик запускает алгоритм и получает предсказанное значение.

Структура файлов

- cohort_revenue_predict.ipynb - ноутбук с запуском обучения и тестированием
- data_container.py - класс для обработки данных
- models.py - модели кластеризации и регрессии

Технологии

- Модель GradientBoostingRegressor от sklearn
- Модель ElasticNet от sklearn
- KMeans-кластеризация от sklearn

Анализ поведения пользователей с применением кластеризацииЧто делает:

В рамках данного проекта решается задача анализа поведения пользователей

Подгружает сырые данные, которые содержат информацию о совершенных событиях.

Входные данные содержат столбцы признаков: id пользователя event_name event_time install_time

Для анализа отбираются следующие события

открытие приложения

добавление в корзину

добавление в вишлист

совершение покупки

Алгоритм агрегирует сырые данные, формируя вектора пользователей со статистикой по событиям. Эти вектора содержат информацию о количестве совершенных пользователем действий с момента установки.

На основе агрегированных данных пользователей обучается алгоритм кластеризации k-means с небольшим количеством кластеров. Для каждого кластера считаются усредненные значения признаков, чтобы охарактеризовать эти кластеры. Это позволяет упростить анализ кластеров и подсчитать, средние значения действий для каждого кластера.

Для визуального анализа поведения пользователей, помимо кластеризации данных строятся графики:

- среднее кол-во добавлений в вишлист от дня недели
- кол-во событий от дня недели
- добавление в корзину на N-ый день после установки
- добавление в вишлист на N-ый день после установки
- среднее кол-во покупок на N-ый день на человека
- вероятность покупки после добавления в корзину на N-ый час

Как использовать:

1. загрузить данные с нужными столбцами в директорию с ноутбуком data_analysis.ipynb
2. Запустить выполнение ноутбука data_analysis.ipynb

Технологии:

- KMeans-кластеризация от sklearn
- построение графиков с помощью matplotlib

Прогнозирование Conversion Rate

./Brandformance/predictCR

Что делает:

На основе исторических данных обучается модель для прогнозирования уровня конверсии.

Данные для обучения представляют собой:

количество показов рекламы в определенный день в медиа источнике

количество новых пользователей пришедших из источника

платформа (android/ios)

Модель обучается для прогнозирования количества новых пользователей, которые придут после заданного количества показов в определенном медиа источнике.

Для обучения модели также рассчитываются новые признаки на основе изначальных данных:

день недели, в который были осуществлены показы

кол-во показов в квадрате

логарифм от кол-ва показов

Задача решается, как задача регрессии, т.к. предсказывается число новых пользователей. Для решения используется алгоритм градиентного бустинга. На основе предсказаний модели и при условии, что количество показов заранее известно, можно подсчитать уровень конверсии показов рекламы.

После этого алгоритм рассчитает новые признаки и сформирует прогноз, а результат запишет в итоговую таблицу в формате .xlsx

Как использовать:

1. обучить модель с помощью fit_model.ipynb на основе исторических данных о конверсии
2. Для получения результатов модели необходимо сформировать таблицу, в которой будут следующие данные:
3. дата, совершения показов
4. кол-во показов
5. платформа (android/ios)
6. запустить inference.ipynb с предварительно создав таблицу с известными данными для прогнозирования

Технологии:

Модель GradientBoostingRegressor от sklearn

Структура файлов

./model_fitting/data_processing.py - функции для обработки данных

./model_fitting/fit_model.ipynb - ноутбук для обучения модели

inference.ipynb - ноутбук для формирования прогнозов

Прогнозирование докатных покупок

./PredictBK

Что делает:

На основе исторических данных о покупках формирует прогноз кол-ва покупок на следующий месяц, разбивая результаты на когорты по месяцам установок типу рекламной компании и типу платформы .

Для получения обучающего датасета, сырые данные из таблицы группируются по:

- значению platform
- type_rk
- source
- по месяцу установки пользователя совершившего покупку
- по месяцу совершения покупки при группировке подсчитывается суммарное кол-во покупок для такой когорты по месяцам

Таким образом для каждого сочетания platform, type_rk, source, месяц установки собирается датасет, представляющий собой временной ряд. Для каждого такого набора данных обучается своя модель линейной регрессии, целевой переменной для которой является количество покупок.

Основным признаком является Кол-во месяцев с момента установки, также на основе этого значения подсчитываются признаки:

кол-во месяцев с установки в квадрате

кол-во месяцев с установки в кубе

логарифм от кол-ва месяцев с момента установки

После обучения модели формируется прогноз, для прогноза кол-ва покупок для конкретной когорты подсчитывается, каким по счеты с момента установки будет следующий месяц для каждой когорты. На основе этого значения формируются необходимые признаки и с помощью модели выдается прогноз.

Таким образом для каждой когорты по месяцу установки предсказывается кол-во покупок на следующий месяц.

Для удобства представления результатов, в итоговой таблице результаты по когортам суммируются и финальный результат представляет из себя кол-во покупок для каждой когорты по типу источника

Как использовать:

В ноутбуке session_notebook.ipynb задать параметры finish_date (текущей даты), start_date(самая ранняя дата установок в истории)

Запустить session_notebook.ipynb

Взять результаты в виде .xlsx таблицы из директории ./data_storage/<название текущего месяца>/next_month_predicts/list_n_purchases.xlsx

Структура файлов

./data_processing/data_processing.py - обработка данных для обучения

./data_processing/date_utils.py - функции для работы с датами

./predicting/train_predict.py - фнкции для обучения модели и формирования предиктов

./session_notebook.ipynb - основной ноутбук для обучения моделей и формирования прогнозов

Технологии:

Модель линейной регрессии от sklearn

Алгоритм server-to-server в MMP

Что это такое?

Передача событий server-to-server в трекинговую платформу на основе внутренней базы данных для улучшенного таргетинга на конкретных пользователей.

Сервис позволяет передавать события со следующими параметрами:

- идентификатор пользователя в трекинговой системе;
- рекламный идентификатор устройства;
- время события;
- время установки;
- операционная система;
- произвольный json-объект с дополнительными параметрами события, например, номером заказа или суммой покупки;
- валюта дохода.

Как это работает?

Генерация передаваемых событий может осуществляться как специальным предназначенным для этого микро-сервисом, так, при необходимости, и другими микро-сервисами платформы. Алгоритмы генерации событий задаются индивидуально под каждую бизнес-задачу, и, как правило, включают в себя поиск, фильтрацию и обработку сохраненных данных о пользовательских событиях и установках, подходящих под целевые критерии. Генерация выполняется периодически, согласно бизнес-требованиям и возможностям сбора данных.

Передача созданных событий осуществляется запросами к server-to-server методам API трекинговой платформы.