

[КАК СТАТЬ АВТОРОМ](#)[Опрос: что вы хотите видеть на Хабре](#)

leontyev_anton 5 марта в 03:09

Пишем простой счетчик для сайта или приложения с помощью Google Cloud Functions и AWS Lambda/Snowflake

Amazon Web Services *, Веб-аналитика *, Аналитика мобильных приложений *, Google Cloud Platform *, Data Engineering *

[Из песочницы](#)

Как работают популярные счетчики веб или мобильной аналитики, например, Google Analytics или AppsFlyer? На сайт устанавливаются их коды или в приложение интегрируется мобильное SDK. Потом при каждом действии клиента отправляется http запрос на сервер

Все потоки Разработка Администрирование Дизайн Менеджмент Маркетинг Научпоп



xxxxxxxxxxxx&tid=1413883337.1042776302&en=order', где.

- **tid** - уникальный идентификатор потока(счетчика)
- **cid** - уникальный идентификатор браузера
- **en** - название события

У использования стандартных счетчиков/пикселей есть минусы:

- некоторые посетители используют анонимайзеры, которые блокируют такие запросы;
- их сложно кастомизировать под себя.

В этой статье напишем собственный счетчик, который будет решать эти проблемы. Счетчик встроим в PowerBI отчеты. Но принцип одинаков, его можно будет использовать и на веб-сайте, и в приложении, и в других устройствах с доступом к интернету. Попробуем две точки сбора событий, чтобы изучить больше технологий: Google Cloud Function, которая будет писать события в Google BigQuery, и Amazon Lambda Functions с записью событий в Snowflake.

1) Заходим на страницу [Google Cloud Function - Create function](#). Убеждаемся, что установлен

триггер **HTTP**, не установлена галочка напротив **Require authentication**. Желательно, чтобы **Region** совпадал с регионом в вашем датасете в BigQuery.

< > ↺ ☰ | 🔒 <https://console.cloud.google.com/functions/add?project=...>

☰ Google Cloud Platform 🔍 Search Products

(⋮) Cloud Functions < Create function

1 Configuration — 2 Code

Basics

Environment
1st gen ▼ ?

Function name *
powerbi ?

Region
us-central1 ▼ ?

Trigger

🔗 HTTP

Trigger type
HTTP ▼

URL 📄
<https://us-central1-...cloudfunctions.net/powerbi>

Authentication

☒ Allow unauthenticated invocations
Check this if you are creating a public API or website.

☐ Require authentication
Manage authorized users with Cloud IAM.

☒ Require HTTPS ?

SAVE CANCEL

NEXT CANCEL

URL на скриншоте – это тот адрес, куда нужно будет отправлять события. Например, `https://us-central1-project-name.cloudfunctions.net/powerbi`

Через GET параметры можно передавать любые данные. Например, при использовании на веб-сайте будет полезно передавать `referrer`. В нашем же случае это будет название отчета в PowerBI и имя страницы. Тогда адрес для отправки будет иметь вид: `https://us-central1-project-name.cloudfunctions.net/powerbi?report=MyReport&page=Page1`

2) Далее попадаем на страницу редактирования кода функции. Выбираем Python. В файле **requirements.txt** прописываем библиотеки, которые нужно подгрузить в проект:

```
requests>=2.27.1
google.cloud>=0.34.0
google-cloud-bigquery>=2.32.0
```

В веб-интерфейсе создаем файл **config.py** следующего содержания:

```
# id чата в телеграме куда будут отправляться ошибки. чтобы узнать свой id - напишите @BotFather
tg_chat_id = 'XXXXXXX'
# токен бота. чтобы его получить, создайте бота написав @BotFather
tg_bot_token = 'YYYYYYY:YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY'
# название таблицы в BigQuery куда будут писаться события. Ее нужно создать заранее.
bq_table_name = 'project-name.dataset.powerbi_views'
```

Файл **main.py**:

```
import datetime
import requests
from google.cloud import bigquery
from config import tg_chat_id, tg_bot_token, bq_table_name

def main(request, report='', page='', ip='', user_agent=''):
    if 'report' in request.args:
        report = request.args['report']
    if 'page' in request.args:
```

```

    page = request.args['page']
    #for headers in request.headers: header = header + str(headers) # получить все за
    ip = request.headers['X-Forwarded-For']
    user_agent = request.headers['User-Agent']

    if report != '':
        return stream_bq(report, page, ip, user_agent)
    else:
        url = 'https://api.telegram.org/bot' + tg_bot_token + '/sendMessage?chat_id=' +
        response = requests.get(url)
        return 'event without parameters received'

def stream_bq(report, page, ip, user_agent):
    bq_client = bigquery.Client()
    bq_table = bq_client.get_table(bq_table_name)
    row = [{'report': report,
            'page': page,
            'ip': ip,
            'timestamp': datetime.datetime.utcnow(),
            'user_agent': user_agent }]
    bq_client.insert_rows(bq_table, row)
    return 'ok'

```

Нажимаем **Deploy**, проверяем работу: открываем в браузере **URL** из пункта 1. Должны увидеть текст “ок”. Если это не так – ищем ошибки в **Logs**.

3) В BigQuery данные будут выглядеть следующим образом:

| report | page | ip | timestamp | user_agent |
|----------|-------|----|--------------------------------|--------------------------------------------------|
| MyReport | Page1 | | 2022-02-19 18:30:25.176878 UTC | Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537. |

Далее обогатим эти данные:

3.1) По **ip** определим страну, город, почтовый индекс, используя данные [GeoLite2 Free Geolocation Data](#):

```

SELECT *

FROM ( SELECT ip FROM `project-name.dataset.powerbi_views` GROUP BY ip ) AS q

CROSS JOIN
(
  SELECT country_name, network, network_bin, city_name, postal_code, mask
  FROM `fh-bigquery.geocode.201806_geolite2_city_ipv4_locs`
) AS g

WHERE network_bin = NET.IP_FROM_STRING(ip) & NET.IP_NET_MASK(4, mask)

```

Результат будет выглядеть примерно так:

| ip | country_name | network | network_bin | city_name | postal_code | mask |
|----|--------------|------------------|-------------|---------------|-------------|------|
| | Russia | 178.252.124.0/22 | svx8AA== | St Petersburg | 190005 | 22 |

3.2) Из `user_agent` выделим операционную систему и браузер, используя JS библиотеку [woothee](#):

```

CREATE OR REPLACE FUNCTION `project-name.dataset.decode_user_agent` (ua STRING)
  RETURNS STRING
  LANGUAGE js AS """ return JSON.stringify(woothee.parse(ua));
  """ OPTIONS(library="gs://project-name-bucket/woothee.js");

```

```

SELECT JSON_VALUE(json, '$.os') AS os, JSON_VALUE(json, '$.name') AS browser, json, use
FROM
(
  SELECT user_agent, `bproject-name.dataset.decode_user_agent`(user_agent) AS json
  FROM `bi-analytics-318415.Others.powerbi_views`
  GROUP BY user_agent
)

```

Результат:

| browser | os | json |
|---------|-------|-------------------------------------------------------------------------------------------------------------------------------|
| Chrome | Linux | <code>{"name":"Chrome","vendor":"Google","version":"98.0.4758.80","category":"pc","os":"Linux","os_version":"UNKNOWN"}</code> |

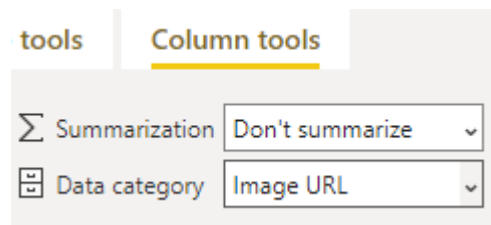
3.3) Теперь мы знаем не только названия отчетов и страницы, которые смотрят пользователи, но и информацию о местоположении, устройстве и операционную систему. Можно сформировать отчетность на свой вкус.

4) Интегрируем наш счетчик в PowerBI. Для этого в редакторе создаем таблицу с единственным значением – нашим URL:

Create Table

| | Column1 | + |
|---|---------------------------------------------------------------------------------------------------------|---|
| 1 | <code>https://us-central1-project-name.cloudfunctions.net/powerbi?report=MyReport&page=Page1</code> | |
| + | | |

Затем выбираем эту колонку справа и в меню **Column tools** указываем **Data category = Image URL**:

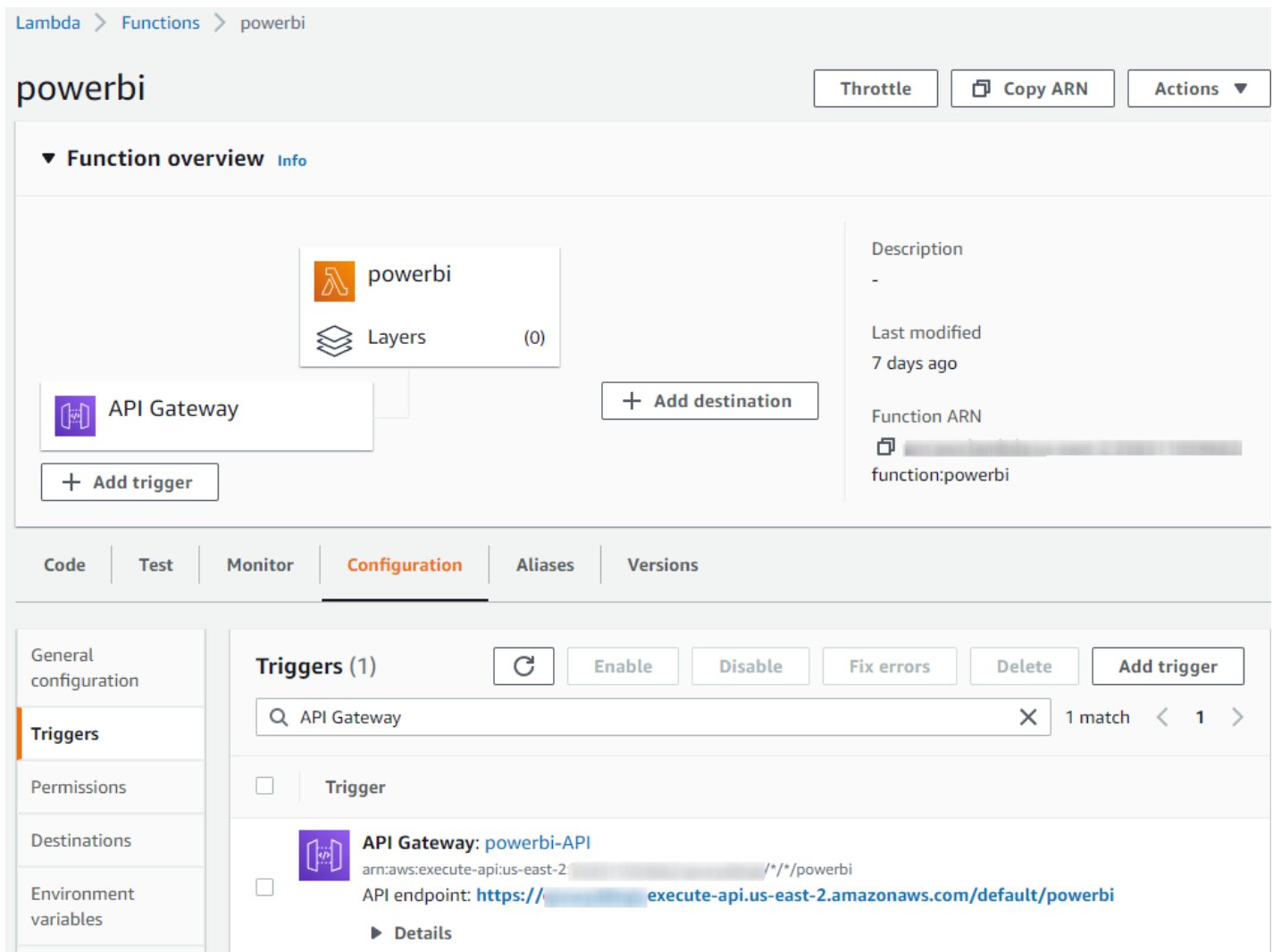


После этого вставляем в отчет визуальный элемент “таблица” с этой колонкой где-нибудь в малозаметном углу отчета. Теперь при каждом его открытии будет уходить http запрос, который через Cloud Functions будет фиксироваться в BigQuery. **Наш счетчик готов!**

5) Чтобы не быть ограниченным одним облачным вендором, сделаем аналогичное на AWS Lambda. В качестве хранилища будем использовать Snowflake, но вы можете использовать любое, например, Redshift. Скорость записи (по логам) в таком случае будет медленней чем в первом примере Google Cloud Function -> Google BigQuery. Но у нас еще побочная цель – изучить разные подходы, чтобы каждый мог для себя выбрать оптимальный вариант.

Заходим в [AWS Lambda](#). Создаем новую функцию **powerbi**. Вариант создания – “Start with a simple Hello World example”. **Runtime** – указываем версию Python, которая установлена у вас на локальном компьютере, например, Python 3.8. Далее нужно добавить триггер типа **API**

Gateway - HTTP API. Security указываем **Open**. После выполненных действий в консоли, функция должна выглядеть примерно так:



URL для отправки событий будет иметь вид: `https://your-id.execute-api.us-east-2.amazonaws.com/default/powerbi?report=MyReport&page=Page1`

По умолчанию Timeout для функции (время ее работы) составляет 3 секунды. Для нас этого недостаточно, т.к. будем писать во внешний сервис Snowflake. Особенно много времени нужно, если события будут приходить редко, т.к. много времени уходит на запуск Warehouse в Snowflake после его остановки. Заходим в **Configuration -> General configuration -> Edit -> Timeout = 12 sec**.

6) Python код нашей функции будет использовать библиотеки **requests** и **snowflake.connector**. В AWS Lambda библиотеки подключаются не так как в Google Cloud Functions. Поэтому сначала создадим проект на локальном компьютере, потом загрузим в

Lambda zip-архив. [Инструкция](#). По моему опыту, самым оптимальным является вариант **Using a virtual environment**.

На локальном компьютере создаем пустую папку, где будут лежать файлы проекта.

Создаем файл **config.py** следующего содержания:

```
# id чата в телеграме куда будут отправляться ошибки. чтобы узнать свой id - напишите @BotFather
tg_chat_id = 'XXXXXXX'

# токен бота. чтобы его получить, создайте бота написав @BotFather
tg_bot_token = 'YYYYYYY:YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY'

# ниже реквизиты доступа от snowflake. таблицу нужно создать заранее
sn_user = 'user_name'
sn_password = 'user_password'
sn_account = 'youproject.us-east-2.aws'
sn_database = 'test'
sn_schema = 'PUBLIC'
sn_table = 'powerbi_views'
sn_warehouse = 'COMPUTE_WH'
sn_role_name = 'ACCOUNTADMIN'
```

Создаем файл **lambda_function.py** :

```
import datetime
import json
import requests
import snowflake.connector

from config import tg_chat_id, tg_bot_token, sn_user, sn_password, sn_account, \
    sn_warehouse, sn_database, sn_table, sn_schema

def lambda_handler(event={}, context='', report='', page='', ip='', user_agent=''):
    if event.get('queryStringParameters') is not None:
        if 'report' in event['queryStringParameters']:
            report = event['queryStringParameters']['report']
        if 'page' in event['queryStringParameters']:
            page = event['queryStringParameters']['page']
        ip = event['headers']['x-forwarded-for']
```

```

user_agent = event['requestContext']['http']['userAgent']
time = datetime.datetime.utcnow()
return stream_snowflake(report, page, ip, time, user_agent)
else:
    url = 'https://api.telegram.org/bot' + tg_bot_token + '/sendMessage?chat_id=' +
    response = requests.get(url)
    return 'event without parameters received'

def stream_snowflake(report, page, ip, time, user_agent):
    error = False
    try:
        conn = snowflake.connector.connect(user=sn_user, password=sn_password, account=
                                           database=sn_database, schema=sn_schema)

        cs = conn.cursor()
        cs.execute(
            "INSERT INTO " + sn_table + "(report, page, ip, time, user_agent) VALUES "
            "    (%s, %s, %s, %s, %s) ", (report, page, ip, time, user_agent))
    except Exception as e:
        error = True
        url = 'https://api.telegram.org/bot' + tg_bot_token + '/sendMessage?chat_id=' +
        response = requests.get(url)
    finally:
        try:
            cs.close()
        except Exception as u:
            pass
    if error == False:
        return 'Write in snowflake successfully'
    else:
        return 'Was error with Snowflake. Send message to telegram'

# это код для запуска на локальном компьютере
# if __name__ == "__main__":
#     lambda_handler()

```

Проверим на локальном компьютере, что наш скрипт работает. Для этого нужно раскомментировать блок снизу и в функции **lambda_handler** прописать вызов **stream_snowflake** без условий (то есть с пустыми значениями **report** и т.д.) Если все норм, загружаем наш проект в AWS Lambda по инструкции в начале этого пункта.

Открываем в браузере URL из пункта 5 для запуска Lambda функции, проверяем что все работает. Логи находятся в **Monitor** -> **Logs**.

7) В Snowflake собираемы данные должны выглядеть следующим образом:

| ID | REPORT | PAGE | IP | TIME | USER_AGENT |
|----|----------|-------|----|-------------------------|----------------------------------------------------------|
| 5 | MyReport | Page1 | | 2022-02-20 08:38:06.783 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53 |

Далее обогатим эти данные:

7.1) Стандартного общедоступного набора данных по geo-ip в Snowflake нет (или я не нашел). Но на Snowflake Data Marketplace есть возможность получить 2-х недельный триал к [IPINFO: IP GEOLOCATION](#). После получения доступа скопируем данные себе. После окончания триала они перестанут обновляться. На мою первую заявку не было никакой реакции, поэтому пришлось зарегистрировать второй аккаунт – с ним все получилось.

После получения доступа, если мы попробуем скопировать таблицы командой **CLONE** - но там покажется ошибка: *SQL compilation error: Cannot clone from a table that was imported from a share.*

Поэтому скопируем двумя командами:

```
CREATE TABLE test.public.location_ip LIKE ipinfo_snowflake_myproject_trial.public.local
INSERT INTO test.public.location_ip SELECT * FROM ipinfo_snowflake_myproject_trial.publ
```

Далее для получения региона и индекса (не влез на скриншот) можно использовать этот SQL-запрос:

```
SELECT *

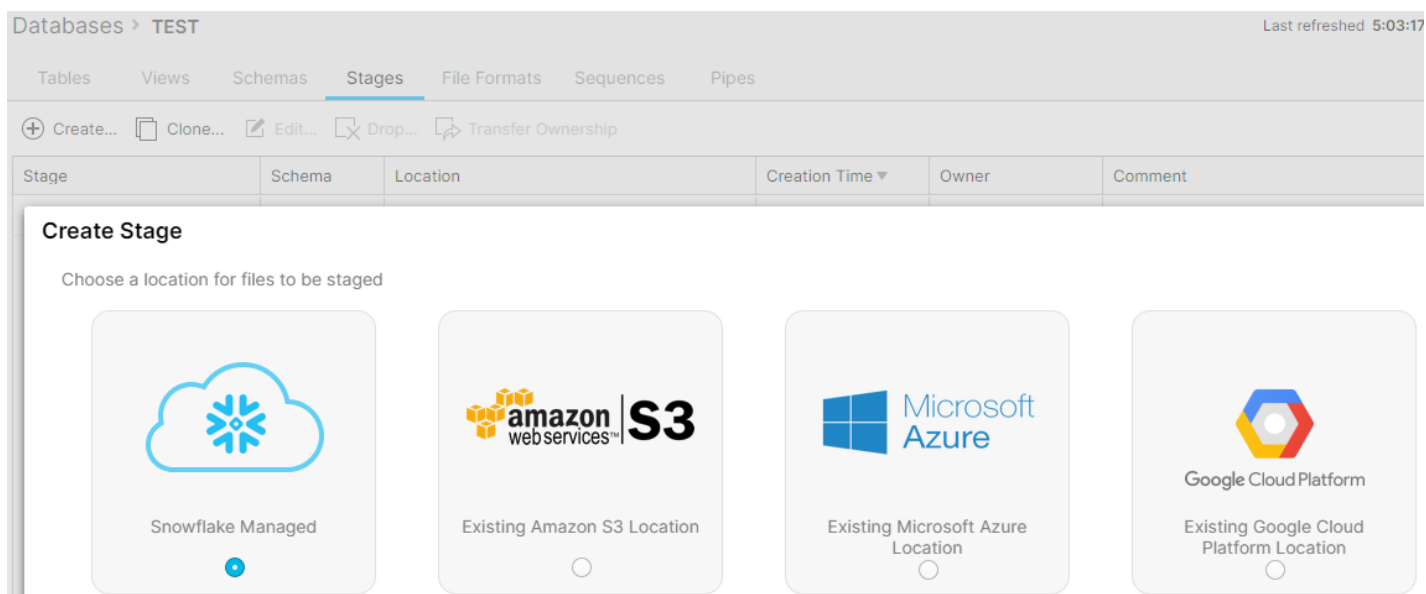
FROM
(
    SELECT ip, PARSE_IP(ip, 'inet'):ipv4 AS ipv4
    FROM ( SELECT ip FROM test.public.powerbi_views WHERE ip='X.X.X.X' GROUP BY ip )
) AS q
```

```
LEFT JOIN test.public.location_ip AS l ON q.ipv4<=l.end_ip_int AND q.ipv4>=l.start_ip_i
```

| IP | IPV4 | START_IP_INT | END_IP_INT | JOIN_KEY | START_IP | END_IP | CITY |
|----|------------|--------------|------------|------------|---------------|-----------------|-----------------|
| | 3002892264 | 3002892032 | 3002892287 | 3002859520 | 178.252.127.0 | 178.252.127.... | Saint Peters... |

7.2) Для парсинга **user_agent** не получится использовать JavaScript UDF функцию, т.к. в Snowflake они не поддерживаются. Поддержка Java UDF функций [экспериментальна и доступна только для проектов на AWS](#). Будем использовать библиотеку [Yauaa](#).

Перейдем в БД **TEST** и создадим Stage. Назовем его, например, **YAUAA**:



В случае, если выбрали **Snowflake Managed**, то загрузить файл туда можно с помощью [SnowSQL \(CLI Client\)](#). Неочевидный момент: в файле `./snowflake/config` **region** нужно указывать вместе с вендором, например **us-east-2.aws**. После запуска утилиты файл можно загрузить командой **PUT**: `put file:///home/anton/yauaa-snowflake-6.9-udf.jar @YAUAA;`

Проверим командой: `LIST @YAUAA;`

Теперь можно создать UDF функцию. Будьте внимательны, директива **imports** регистрозависима:

```
use test;

create or replace function parse_useragent(useragent VARCHAR)
returns object
language java
imports = ('@YAUAA/yauaa-snowflake-6.9-udf.jar')
handler = 'nl.basjes.parse.useragent.snowflake.ParseUserAgent.parse';
```

SQL-код для определения операционной системы и браузера:

```
SELECT
  parse_useragent(user_agent):AgentName::string AS browser,
  parse_useragent(user_agent):OperatingSystemName::string AS os,
  user_agent
FROM ( SELECT user_agent FROM test.public.powerbi_views GROUP BY user_agent )
```

| BROWSER | OS | USER_AGENT |
|---------|------------|-------------------------------------------------------------------|
| Chrome | Windows NT | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (K |
| Chrome | Linux | Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like G |

Таким образом мы реализовали в Snowflake такой же функционал, как в BigQuery в пунктах 3 - 4. Выбор за вами. Счетчик можно интегрировать не только в PowerBI, но и в другие BI решения, установить на сайт, в приложение или программу.

Теги: google analytics, appsflyer, powerbi, aws lambda, bigquery, snowflake, data engineering, пиксели

Хабы: Amazon Web Services, Веб-аналитика, Аналитика мобильных приложений, Google Cloud Platform, Data Engineering

Редакторский дайджест

Присылаем лучшие статьи раз в месяц





2

Карма

2

Рейтинг

@leontyev_anton

Пользователь

Реклама

Комментарии



Здесь пока нет ни одного комментария, вы можете стать первым!

Только полноправные пользователи могут оставлять комментарии. [Войдите](#), пожалуйста.

ПОХОЖИЕ ПУБЛИКАЦИИ

15 февраля в 03:42

Франция забанила Google Analytics

+33

13K

8

106 +106

20 января в 08:35

Австрийский регулятор запретил использовать Google Analytics по соображениям безопасности

+10

3K

5

4 +4

9 декабря 2021 в 01:49

Где и как искать данные о целевой аудитории: гайд по счетчикам Яндекс.Метрика и Google Analytics

+1

3.6K

13

0

МИНУТОЧКУ ВНИМАНИЯ

Разместить





Разбор ноутбука HONOR MagicBook 16





Хьюстон, на воркшопе этого не было: IT-обучение и космос


КУРСЫ

 Специалист по Data Science плюс
10 марта 2022 · 228 000 ₽ · Яндекс.Практикум

 Data Scientist
12 марта 2022 · 48 990 ₽ · Level UP

 Data Science Bootcamp
14 марта 2022 · 285 000 ₽ · Elbrus Coding Bootcamp

 Применение технологий нейронных сетей и искусственного интеллекта при работе с Big Data. Уязвимости и способы защиты
14 марта 2022 · 45 000 ₽ · Академия информационных систем

 Специалист по Data Science
17 марта 2022 · 95 000 ₽ · Яндекс.Практикум

Больше курсов на Хабр Карьере

ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

сегодня в 04:58

Нет, Open Source не означает «бесплатная поддержка»

 +60 4.7K 35 18 +18

сегодня в 02:00

Простые сайты снова в моде. Минимализм возвращается

 +43 10K 56 15 +15

сегодня в 03:39

Ложь из солидарности: как Thawte убила «систему доверия» в Интернете

 +39 13K 26 143 +143

сегодня в 03:00

Про NFT и деньги

 +20 1.4K 10 0

вчера в 12:52

Проектирование непредсказуемого интеллекта в играх. Часть 2 — интеллект толпы

 +16 3.7K 56 1 +1

Как работа вызывает стресс и как это отследить

Интересно

Реклама

ЧИТАЮТ СЕЙЧАС

Ваш аккаунт

Войти

Регистрация

Разделы

Публикации

Новости

Хабы

Компании

Авторы

Песочница

Информация

Устройство сайта

Для авторов

Для компаний

Документы

Соглашение

Конфиденциальность

Услуги

Реклама

Тарифы

Контент

Семинары

Мегапроекты



Настройка языка

О сайте

Техническая поддержка

Вернуться на старую версию

© 2006–2022 «Habr»

