Сайт для редактирования скрипта для магазина bitrix24:
b24-x5iy25.bitrix24.ru

Сам магазин:
https://b24-dy0v8s.bitrix24.shop/

Js для bitrix24

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/mustache.js/3.0.0/
mustache.min.js"></script>
<script src="https://code.jquery.com/jquery-3.3.1.js"></script>
<script src="https://requirejs.org/docs/release/2.3.6/minified/
require.js"></script>
<script>
    requirejs.config({
      paths: {
        divolte: 'http://35.195.166.173:8290/divolte'
      }
    });
    require(['divolte']);
    require(['divolte'], function(divolte) {
      console.log('ok1');
      checkout_click = function() {
        console.log('ok2');
        var id_product = this.id.split('_buy_link')[0];
        var id_price_product = id_product.concat('_price');
        var id_price_total_product = id_product.concat('_price_total');
        var price_product =
document.getElementById(id_price_product).innerHTML.trim();
        try {
            var price_total_product =
document.getElementById(id_price_total_product).innerHTML.trim().split('
<strong>')[1].split('</strong>')[0];
        } catch(err) {
              price_total_product = price_product
        }
        divolte.signal("checkout", {id: id_product,price:
price_product,total_price: price_total_product});
        return false;
      };
      $(document.body).on('click', '.btn-primary', checkout_click);
    });
</script>
```

Запуск divote

```
sudo /home/ubuntu/divolte-collector-0.9.0/bin/divolte-collector
```

divolte-env.sh

```
HADOOP_CONF_DIR=/usr/hdp/current/hadoop-client/conf
```

divolte-collector.conf

```
divolte {
  mappings {
    my_mapping = {
```

```
      schema_file = "/home/ubuntu/divolte-collector-0.9.0/conf/
MyEventRecord0.avsc"
      mapping_script_file = "/home/ubuntu/divolte-collector-0.9.0/conf/
mapping0.groovy"
      sources = [a_source]
      sinks = [kafka]
    }
  }
  global {
    server {
      host = 0.0.0.0
      port = 8290
    }
    kafka {
      // Enable Kafka flushing
      enabled = true

      // The properties under the producer key in this
      // configuration are used to create a Properties object
      // which is passed to Kafka as is. At the very least,
      // configure the broker list here. For more options
      // that can be passed to a Kafka producer, see this link:
      // http://kafka.apache.org/082/
documentation.html#newproducerconfigs
      producer = {
        bootstrap.servers = "instance-1.europe-west1-b.c.de3-
npl.internal:6667"
      }
    }
  }

  sinks {
    // The name of the sink. (It's referred to by the mapping.)
    kafka {
      type = kafka
      // This is the name of the topic that data will be produced on
      topic = andrey.selivanov
    }
  }

  sources {
    a_source {
      type = browser
      // prefix = /tracking
    }
  }
}
```

MyEventRecord0.avsc – схема avrò для rivolte

```
{
  "namespace": "io.divolte.examples.record",
  "type": "record",
  "name": "MyEventRecord",
```

```
  "fields": [
    { "name": "sessionId",               "type": "string"},
    { "name": "timestamp",               "type": "long" },
    { "name": "referer",                 "type": "string"},
    { "name": "remoteHost",              "type": "string"},
    { "name": "eventType",               "type": ["null", "string"],
"default": null },
    { "name": "location",                "type": ["null", "string"],
"default": null },
    { "name": "localPath",               "type": ["null", "string"],
"default": null },
    { "name": "userAgent", "type": ["null", "string"], "default":
null },
    { "name": "userAgentDeviceCategory", "type": ["null", "string"],
"default": null },
    { "name": "userAgentOsFamily", "type": ["null", "string"],
"default": null },
    { "name": "userAgentOsVersion", "type": ["null", "string"],
"default": null },
    { "name": "id_product", "type": ["null", "string"], "default":
null },
    { "name": "price_product", "type": ["null", "string"], "default":
null },
    { "name": "total_price_product", "type": ["null", "string"],
"default": null }
  ]
}
```

mapping0.groovy – groovy файл для divolte

```
mapping {
  map sessionId() onto 'sessionId'
  map timestamp() onto 'timestamp'
  map referer() onto 'referer'
  map remoteHost() onto 'remoteHost'
  map eventType() onto 'eventType'
  map location() onto 'location'
  map userAgentString() onto 'userAgent'
  def ua = userAgent()
  map ua.deviceCategory() onto 'userAgentDeviceCategory'
  map ua.deviceCategory() onto 'userAgentDeviceCategory'
  map ua.osFamily() onto 'userAgentOsFamily'
  map ua.osVersion() onto 'userAgentOsVersion'
  map eventParameters().value('id') onto 'id_product'
  map eventParameters().value('price') onto 'price_product'
  map eventParameters().value('total_price') onto 'total_price_product'
  def locationUri = parse location() to uri
  def localUri = parse locationUri.rawFragment() to uri
  map localUri.path() onto 'localPath'
}
```

kafka_avro2json.py – скрипт для преобразования потока в Avro от divolte в json для ClickHouse

```
from confluent_kafka import Consumer, Producer, KafkaError
```

```
from fastavro import writer, reader, parse_schema, schemaless_reader
import json
import io

with open('/home/ubuntu/divolte-collector-0.9.0/conf/
MyEventRecord0.avsc') as f:
    schema = parse_schema(json.loads(f.read()))

def decode(msg_value):
    message_bytes = io.BytesIO(msg_value)
    return schemaless_reader(message_bytes, schema)

def getOkk(lst):
    ret ={}
    for x,y in lst.items():
        if x in ['sessionId', 'timestamp', 'location', 'id_product']:
            if x == 'id_product':
                if y != None:
                    ret[x] = y
            else:
                ret[x] = y
                if x == 'location':
                    ret['id_item'] = ''
                    z = y.split('/')
                    if len(z) > 3:
                        n = (-2 if z[-1] == '' else -1)
                        if z[n-1] == 'item':
                            ret['id_item'] = z[n]
        elif x in ['price_product','total_price_product']:
            z = y if y == None else float(y[:-5].replace(' ',''))
            if z != None:
                ret[x] = z
    if ret.get('id_product','') > '':
        ret['item_count'] = (None if ret['price_product'] == None else
                             int(ret['total_price_product']/
ret['price_product']))
    return ret

ip, topic = '35.195.166.173:6667', 'andrey.selivanov'
topicTo1, topicTo2 = 'users_json', 'orders_json'
conf = {'bootstrap.servers': ip}
p1 = Producer(**conf)
p2 = Producer(**conf)
c = Consumer(
    {'bootstrap.servers': ip,
     'group.id': 'avro2json_group',
     'auto.offset.reset': 'latest',
     "enable.auto.commit": True})
c.subscribe([topic])

running = True
while running:
    msg = c.poll()
    if not msg.error():
        msg_value = msg.value()
        print(msg_value)
```

```
        event_dict = getOkk(decode(msg_value))
        if event_dict.get('id_product','') > '':
            p2.produce(topicTo2, json.dumps(event_dict))
        else:
            p1.produce(topicTo1, json.dumps(event_dict))
        print(event_dict)
    elif msg.error().code() != KafkaError._PARTITION_EOF:
        print(msg.error())
        running = False
```

Запуск ClickHouse:

1) sudo nano /etc/clickhouse-server/config.xml
   <listen_host>::</listen_host>
2) sudo service clickhouse-server start

Скрипты для ClickHouse:

создание таблицы статистики по посещению страниц, которая автоматом забирает записи из потока Кафки:

```
CREATE TABLE users (
    sessionId String,
    timestamp UInt64,
    location String,
    id_item String
  ) ENGINE = Kafka('35.195.166.173:6667', 'users_json', 'users_group',
‘JSONEachRow');
```

создание view статистики по посещению страниц, которая сохраняет записи в нужном формате из users:

```
CREATE MATERIALIZED VIEW view_users
ENGINE = MergeTree() ORDER BY (timestamp, sessionId, id_item) SETTINGS
index_granularity=8192 POPULATE AS
SELECT
     timestamp,
    sessionId,
    id_item,
    location
FROM users
```

скрипт проверки работы и вывод необходимых для чекера полей (здесь deep насчитывается не совсем корректно, как общая сумма хитов по всей сессии, а не до страницы):

```
select max(a.location) as url, a.id_item, count(*) as count, avg(b.deep)
as deep
from view_users a
ANY LEFT JOIN (select sessionId, count(*) as deep from view_users group
by sessionId) b USING sessionId
--where a.timestamp >= 1541069791950
group by a.id_item;
```

создание таблицы статистики по заказам, которая автоматом забирает записи из потока Кафки:

```
CREATE TABLE orders (
    sessionId String,
    timestamp UInt64,
    location String,
    id_item String,
    id_product String,
    price_product Float32,
    total_price_product Float32,
    item_count UInt16
  ) ENGINE = Kafka('35.195.166.173:6667', 'orders_json', 'group_orders',
'JSONEachRow');
```

создание view статистики по заказам, которая сохраняет записи в нужном формате из orders:

```
CREATE MATERIALIZED VIEW view_orders
ENGINE = MergeTree() ORDER BY (timestamp, id_item) SETTINGS
index_granularity=8192 POPULATE AS
SELECT
     timestamp,
    id_item,
    location,
    total_price_product,
    item_count
FROM orders
```

скрипт проверки работы и вывод необходимых для чекера полей:

```
select max(location) as url, id_item, sum(total_price_product) as price,
sum(item_count) as count
from view_orders
--where timestamp > 1541069788510
group by id_item;
```

api.py – сайт на flask

```
#!flask/bin/python
from flask import Flask
from clickhouse_driver import Client
from datetime import datetime, timedelta
import time

cl = Client('35.195.166.173',port='9001')
def getTsHourBack():
    dt = datetime.now() - timedelta(hours=1)
    return int(time.mktime(dt.timetuple()))

def getJson(recs,cols,ts):
    lst = []
    for rec in recs:
```

```
        d = {}
        for i,col in enumerate(cols):
            d[col] = rec[i]
        lst.append(d)
    return str({"timestamp": ts,"contents": lst,"check": True})

def getOrders(ts=None):
    if ts == None:
        ts = getTsHourBack()
    else:
        try:
            ts = int(ts)
        except:
            ts = 0
    q = 'select max(location) as url, id_item, sum(total_price_product)
as price, sum(item_count) as count'+\
        ' from view_orders'+\
        ' where timestamp > '+str(ts)+\
        ' group by id_item FORMAT JSONEachRow;'
    return getJson(cl.execute(q),['url','id_item','price','count'],ts)
def getUsers(ts=None):
    if ts == None:
        ts = getTsHourBack()
    else:
        try:
            ts = int(ts)
        except:
            ts = 0
    q = 'select max(a.location) as url, a.id_item, count(*) as count,
avg(b.deep) as deep'+\
        ' from view_users a'+\
        ' ANY LEFT JOIN (select sessionId, count(*) as deep from
view_users group by sessionId) b USING sessionId'+\
        ' where a.timestamp >= '+str(ts)+\
        ' group by a.id_item FORMAT JSONEachRow;'
    return getJson(cl.execute(q),['url','id_item','count','deep'],ts)

app = Flask(__name__)

@app.route('/')
def index():
    return "Это API для чекера"

@app.route('/api/v1.0/users/')
def users1():
    return getUsers()

@app.route('/api/v1.0/users/<ts>')
def users2(ts):
    return getUsers(ts)

@app.route('/api/v1.0/orders/')
def orders1():
    return getOrders()

@app.route('/api/v1.0/orders/<ts>')
```

```
def orders2(ts):
    return getOrders(ts)

if __name__ == '__main__':
    app.run(debug=True,host = "0.0.0.0", port=5001)
```