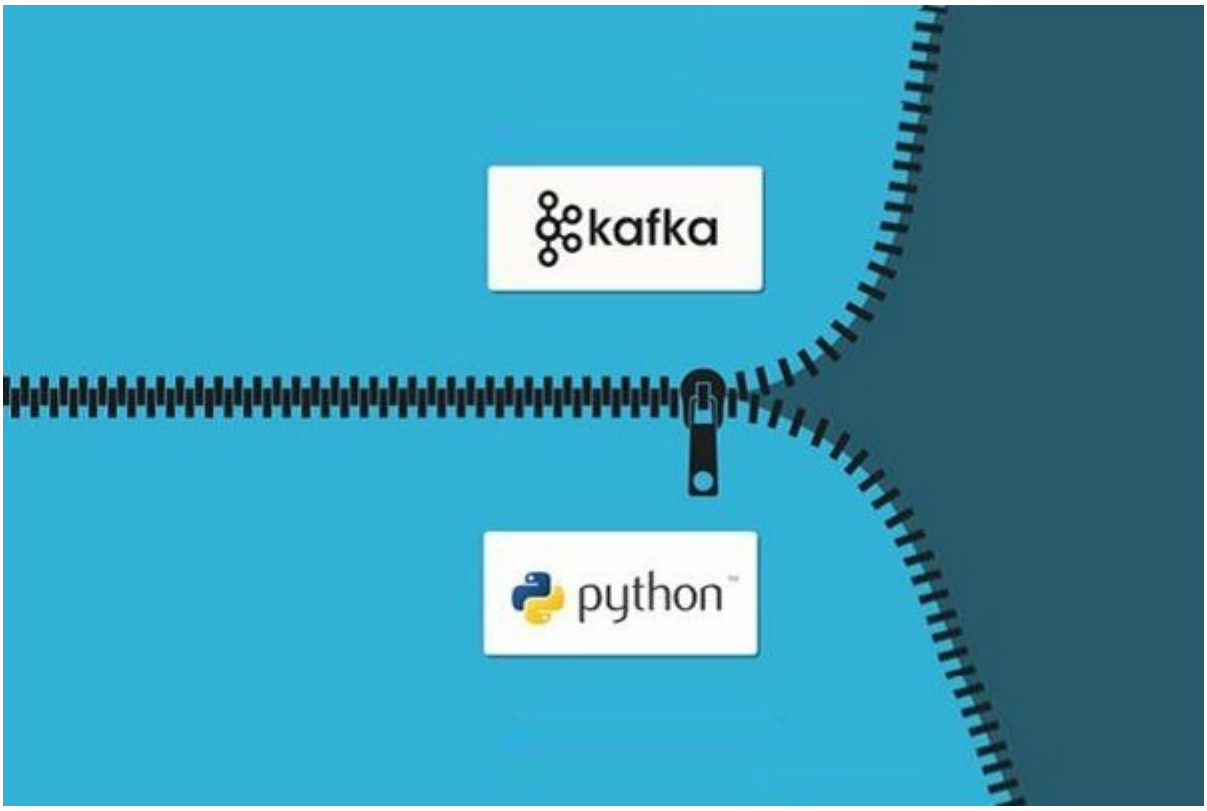
[redacted] Kafka Python [redacted]



[redacted] Kafka Python [redacted] (Metrics) [redacted] Clicky.com [redacted] ... [redacted] Clicky.com API [redacted] admintome-pages Kafka [redacted] AdminTome [redacted] JSON [redacted]

Clicky [redacted]

[redacted] Clicky.com [redacted] Clicky.com [redacted]

[illegible] Google Analytics [illegible] (Clicky [illegible]). [illegible] Clicky [illegible].
[illegible] Clicky API [illegible] Clicky [illegible] Preferences [illegible].
[illegible] Site key [illegible] Site ID [illegible].
[illegible] API [illegible].

[illegible]

[illegible] kafka-mesos.sh [illegible]:

```
cd kafka/ $
$ ./kafka-mesos.sh topic add admintome-pages --broker=0 --api=http://mslave2.admintome.lab:7000
```

[illegible] kafka-mesos [illegible] API [illegible]:

```
kafka-mesos.sh topic list --api=http://mslave2.admintome.lab:7000/. $
topics:
name: __consumer_offsets
partitions: 0:[0], 1:[0], 2:[0], 3:[0], 4:[0], 5:[0], 6:[0], 7:[0], 8:[0], 9:[0], 10:[0], 11:[0], 12:[0], 13:[0],
14:[0], 15:[0], 16:[0], 17:[0], 18:[0], 19:[0], 20:[0], 21:[0], 22:[0], 23:[0], 24:[0], 25:[0], 26:[0], 27:[0],
28:[0], 29:[0], 30:[0], 31:[0], 32:[0], 33:[0], 34:[0], 35:[0], 36:[0], 37:[0], 38:[0], 39:[0], 40:[0], 41:[0],
42:[0], 43:[0], 44:[0], 45:[0], 46:[0], 47:[0], 48:[0], 49:[0]
options: segment.bytes=104857600,cleanup.policy=compact,compression.type=producer
name: admintome
partitions: 0:[0]
name: admintome-pages
[partitions: 0:[0
```

[illegible] Python [illegible].
[illegible] Clicky API [illegible] JSON [illegible].
[illegible] Virtualenv [illegible] Python 3 [illegible].

```
mkdir ~/Development/python/venvs $
$ mkdir ~/Development/python/site-stats-intake
$ cd ~/Development/python/site-stats-intake
$ virtualenv ../venvs/intake
$ source ../venvs/intake/bin/activate
(intake) $ pip install kafka-python requests
(intake) $ pip freeze > requirements.txt
```

[illegible]

نام کتاب : مقدمه‌ای بر برنامه نویسی با پایتون

کلاس Clicky

ما یک کلاس پایتون جدید به نام Clicky ایجاد میکنیم که از آن برای تعامل با Clicky API استفاده خواهد شد. یک فایل جدید با نام clicky.py بسازید و محتوای زیر را به آن اضافه کنید :

```
import requests
import json
class Clicky(object):
def __init__(self, site_id, sitekey):
self.site_id = site_id
self.sitekey = sitekey
self.output = “json”
def get_data(self, data_type):
click_api_url = “https://api.clicky.com/api/stats/4”
payload = {“site_id”: self.site_id,
“sitekey”: self.sitekey,
“type”: data_type,
“output”: self.output}
response = requests.get(click_api_url, params=payload)
raw_stats = response.text
return raw_stats
def get_pages_data(self):
data = self.get_data(“pages”)
return json.loads(data)
```

حالا این فایل را ذخیره کرده و از آن خارج شوید . به منظور دستیابی به سنجهها، باید یک درخواست HTTP GET را به Clicky API URL به روش زیر ارسال کنیم :

https://api.clicky.com/api/stats/4

همچنین باید چند پارامتر دیگر را وارد کنیم :

- site_id: این همان مقدار Site ID است که ما قبلا به دست آورده بودیم .
- Sitekey: این همان مقدار Site key است که ما قبلا به دست آورده بودیم .
- Type: برای بهدست آوردن صفحات بالا ما نوع صفحات را مشخص میکنیم.
- Output: ما این مقدار را json قرار میدهیم تا API خروجی داده را با فرمت JSON ارسال کند .

در نهایت ما درخواست ماژول پایتون را فراخوانی میکنیم تا یک HTTP GET به API URL ما با پارامترهایی که تعیین کردیم، اجرا شود . با متد get_pages_data ما دادههای JSON خود را دریافت میکنیم . سپس پیادهسازی کلاس کافکا خود را کدنویسی میکنیم.

کلاس MyKafka

این کلاس با کلاستر کافکا یکپارچه شده و سنجههای وبسایت را به تاپیک وارد میکند . یک فایل جدید به نام mykafka.py بسازید و محتوای زیر را به آن اضافه کنید :

```
from kafka import KafkaProducer
import json
class MyKafka(object):
def __init__(self, kafka_brokers):
self.producer = KafkaProducer(
```

```
value_serializer=lambda v: json.dumps(v).encode('utf-8'),
bootstrap_servers=kafka_brokers
)
def send_page_data(self, json_data):
self.producer.send('admintome-pages', json_data)
```

ابتدا ما کتابخانه kafka-python و به طور مشخص کلاس KafkaProducer را وارد میکنیم که به ما اجازه میدهد، یک تولیدکننده کافکا را کدنویسی کنیم و پیغام را به Kafka Topic منتشر کنیم .

```
from kafka import KafkaProducer
```

حالا ما کلاس MyKafka را تعریف کرده و یک تابع سازنده برای آن ایجاد میکنیم :

```
class MyKafka(object):
def __init__(self, kafka_brokers):
```

این یک آرگمان میگیرد که واسطههای کافکا را که برای اتصال به کلاستر کافکا استفاده میشوند، ارائه میکند . این یک آرایه از رشتهها در قالب زیر است :

```
[ "broker:ip", "broker:ip" ]
```

ما در اینجا تنها از واسطه mslave1.admintome.lab:31000 استفاده خواهیم کرد :

```
[ "mslave1.admintome.lab:31000" ]
```

سپس یک آبجکت KafkaProducer به نام Producer تعریف میکنیم . از آنجا که ما دادهها را در قالب JSON به کافکا ارسال خواهیم کرد، به KafkaProducer میگوییم تا با استفاده از پارامتر value_serializer از رمزگشای JSON برای تفسیر این دادهها استفاده کند . همچنین باید مشخص کنیم که با پارامتر bootstrap_servers از واسطههای ما استفاده شود .

```
self.producer = KafkaProducer(
value_serializer=lambda v: json.dumps(v).encode('utf-8'),
bootstrap_servers=kafka_brokers
)
```

سرانجام ما یک متد جدید ایجاد میکنیم که از آن برای ارسال پیغامها به تاپیک admintome-pages استفاده میشود :

```
def send_page_data(self, json_data):
self.producer.send('admintome-pages', json_data)
```

در اینجا کار به پایان میرسد . حالا ما کلاس Main را ایجاد میکنیم که همه چیز را کنترل میکند .

مطلب پیشنهادی

مدیریت انبوه جریانهای دادهای

کافکا معماری پردازش جریان دادهای لینکدین، بدون رقیب

کلاس Main

یک فایل جدید به نام main.py بسازید و محتوای زیر را به آن اضافه کنید :

```
from clicky import Clicky
from mykafka import MyKafka
import logging
import time
import os
from logging.config import dictConfig
class Main(object):
def __init__(self):
```

```
if 'KAFKA_BROKERS' in os.environ:
kafka_brokers = os.environ['KAFKA_BROKERS'].split(',')
else:
raise ValueError('KAFKA_BROKERS environment variable not set')
if 'SITE_ID' in os.environ:
self.site_id = os.environ['SITE_ID']
else:
raise ValueError('SITE_ID environment variable not set')
if 'SITEKEY' in os.environ:
self.sitekey = os.environ['SITEKEY']
else:
raise ValueError('SITEKEY environment variable not set')
logging_config = dict(
version=1,
formatters={
'f': {'format':
'%(asctime)s %(name)-12s %(levelname)-8s %(message)s'}
},
handlers={
'h': {'class': 'logging.StreamHandler',
'formatter': 'f',
'level': logging.DEBUG}
},
root={
'handlers': ['h'],
'level': logging.DEBUG,
},
)
self.logger = logging.getLogger()
dictConfig(logging_config)
self.logger.info("Initializing Kafka Producer")
self.logger.info("KAFKA_BROKERS={0}".format(kafka_brokers))
self.mykafka = MyKafka(kafka_brokers)
def init_clicky(self):
self.clicky = Clicky(self.site_id, self.sitekey)
self.logger.info("Clicky Stats Polling Initialized")
def run(self):
self.init_clicky()
starttime = time.time()
while True:
data = self.clicky.get_pages_data()
self.logger.info("Successfully polled Clicky pages data")
self.mykafka.send_page_data(data)
self.logger.info("Published page data to Kafka")
time.sleep(300.0 - ((time.time() - starttime) % 300.0))
if __name__ == "__main__":
logging.info("Initializing Clicky Stats Polling")
() main = Main
()main.run
```

□□□□ □□□ .□□□□□□ □□□□ Marathon □□ □□ □□ □□□ □□ □□ □□□ □□□□ □□□□□□□ □□ □□□□ □□□□ □□□ □□□□□ □□□ □□ clicky □□ □□□ site key □ site id □□□ □□□ □□□□ □□□□□□□ □□□□□□ □□□ □□ □□ □□□□□ □□□□□ □□□ □□ □□

مستقیم در کدها وارد کنیم . ما میخواهیم این امکان را داشته باشیم تا این اطلاعات را از متغیرهای محیطی استخراج کنیم.

```
if ‘KAFKA_BROKERS’ in os.environ:
kafka_brokers = os.environ[‘KAFKA_BROKERS’].split(‘,’)
else:
raise ValueError(‘KAFKA_BROKERS environment variable not set’)
if ‘SITE_ID’ in os.environ:
self.site_id = os.environ[‘SITE_ID’]
else:
raise ValueError(‘SITE_ID environment variable not set’)
if ‘SITEKEY’ in os.environ:
self.sitekey = os.environ[‘SITEKEY’]
else:
raise ValueError(‘SITEKEY environment variable not set’)
```

در کد ما یک حلقه تکرار بینهایت گنجاندهشده که هر پنج دقیقه سنجهها را از Clicky استخراج و به تاپیک کافکای ما وارد میکند .

```
def run(self):
self.init_clicky()
starttime = time.time()
while True:
data = self.clicky.get_pages_data()
self.logger.info(“Successfully polled Clicky pages data”)
self.mykafka.send_page_data(data)
self.logger.info(“Published page data to Kafka”)
time.sleep(300.0 - ((time.time() - starttime) % 300.0))
```

فایل را ذخیره کرده و از آن خارج شوید .

در زمان اجرای اپلیکیشن برای آزمایش اینکه آیا همهچیز بهخوبی کار میکند یا خیر میتوانید بعد از تنظیم متغیرهای محیطی خود اپلیکیشن را اجرا کنید :

```
(intake) $ export KAFKA_BROKERS=”mslave1.admintome.lab:31000”
(intake) $ export SITE_ID=”{your site id}”
(intake) $ export SITEKEY=”{your sitekey}”
(intake) $ python main.py
2018-06-25 15:34:32,259 root INFO Initializing Kafka Producer
2018-06-25 15:34:32,259 root INFO KAFKA_BROKERS=[‘mslave1.admintome.lab:31000’]
2018-06-25 15:34:32,374 root INFO Clicky Stats Polling Initialized
2018-06-25 15:34:32,754 root INFO Successfully polled Clicky pages data
2018-06-25 15:34:32,755 root INFO Published page data to Kafka
```

حالا ما پیغامها را به Kafka Topic ارسال میکنیم، بعد کانتینر داکر خود را میسازیم و آن را در Marathon مستقر میکنیم . و در نهایت کار را با نوشتن یک کد آزمایشی که پیغامهای ما را از تاپیک دریافت میکند به پایان میبریم .

تمام کدهای استفادهشده در این مقاله در مخزن GitHub زیر قرار گرفته است :

https://github.com/admintome/clicky-state-intake

حالا که اپلیکیشن ما آمادهشده است میتوانیم یک کانتینر داکر ایجاد کرده و آن را در Marathon مستقر کنیم . یک فایل Dockerfile در دایرکتوری اپلیکیشن خود ایجاد کرده و محتوای زیر را در آن اضافه کنید :

```
FROM python:3
WORKDIR /usr/src/app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
```

```
CMD [ "python", "./main.py" ]
```

حالا کانتینر را بسازید .

```
$ docker build -t {your docker hub username}site-stats-intake .
```

بعد از اینکه ساخت کانتینر کامل شد، باید آن را در مخزن داکر قرار دهید تا Mesos Slaves شما بتواند به آن دسترسی داشته باشد . در مثال ما این Docker Hub است:

```
$ docker push -t admintome/site-stats-intake
```

سپس به هر کدام از Mesos slave خود لاگین کنید .

```
$ docker pull admintome/site-stats-intake
```

حالا ما آمادهایم تا اپلیکیشن Marathon را برای اپلیکیشن خود ایجاد کنیم .
به Marathon GUI خود بروید :
http://mesos1.admintome.lab:8080
روی دکمه Create Application و سپس روی دکمه JSON کلیک کرده و کد JSON زیر را پیست کنید :

```
{
"id": "site-stats-intake",
"cmd": null,
"cpus": 1,
"mem": 128,
"disk": 0,
"instances": 1,
"container": {
"docker": {
"image": "admintome/site-stats-intake"
},
"type": "DOCKER"
},
"networks": [
{
"mode": "host"
}
],
"env": {
"KAFKA_BROKERS": "192.168.1.x:port",
"SITE_ID": "{your site_id}",
"SITEKEY": "{your sitekey}"
}
}
```

اطمینان حاصل کنید که در بخش env مقادیر درست را برای KAFKA_BROKERS, SITE_ID و SITEKEY وارد کرده باشید .

سرانجام روی دکمه Create Application کلیک کنید تا این اپلیکیشن مستقر شود . بعد از چند ثانیه ملاحظه میکنید که اپلیکیشن در حال اجرا است . برای مشاهده لاگها روی اپلیکیشن site-stats-intake کلیک کرده، سپس روی لینک stderr کلیک کنید تا یک فایل متنی حاوی لاگها دانلود شود . حالا که اپلیکیشن ما در Marathon مستقرشده، یک کد کوتاه مینویسیم تا اجرای آن به ما نشان دهد چه پیغامهایی دریافت شده است. این یک Kafka consumer ساده خواهد بود که تاپیک را بررسی میکند و تمام پیغامهای موجود در این تاپیک را نمایش میدهد . شاید این کد چندان کاربردی نباشد اما در این مرحله به ما اجازه میدهد، چگونگی کارکرد اپلیکیشن واکشی خود را آزمایش کنیم .

یک فایل جدید به نام consumer.py ایجاد کرده و محتوای زیر را به آن اضافه کنید :

```
import sys
from kafka import KafkaConsumer
consumer = KafkaConsumer('admintome-pages',
bootstrap_servers="mslave1.admintome.lab:31000",
auto_offset_reset='earliest')
```

```
try:
    for message in consumer:
        print(message.value)
except KeyboardInterrupt:
    sys.exit()
```

فایل را ذخیره کرده و از آن خارج شوید . واسطه کافکا مستقیم داخل این کد اضافهشده است، زیرا ما تنها میخواهیم از آن برای مصارف آزمایشی استفاده کنیم . اطمینان حاصل کنید که پارامتر Bootstrap-Servers را با نام و پورت واسطه خود جایگزین کنید . حالا فرمان را اجرا کنید تا حجم زیادی از JSON را که نمایانگر آخرین صفحات بازدید شده شما است، مشاهده کنید :

```
(intake) $ python consumer.py
b’[{“type”: “pages”, “dates”: [{“date”: “2018-06-25”, “items”: [{“value”: “145”, “value_percent”: “43.2”, “title”: “Kafka Tutorial for Fast Data Architecture - AdminTome Blog”, “stats_url”: “http://clicky.com/stats/visitors?site_id=101045340&date=2018-06-25&href=%2Fblog%2Fkafka-tutorial-for-fast-data-architecture%2F”, “url”: “http://www.admintome.com/blog/kafka-tutorial-for-fast-data-architecture/”},…
```

حالا ما یک کانال ارتباطی از دادهها را در اختیار داریم که اطلاعاتی در خود دارد که ما میتوانیم از آنها استفاده کنیم . گام بعدی این خواهد بود که ما از این دادهها استفاده کرده و آنها را تجزیهوتحلیل کنیم .

نویسنده :
محسن آقاجانی
دسته بندی:
کارگاه
برنامه نویسی
تاریخ انتشار :
12:30 - 06/04/1398
برچسب:
Kafka Python - پایتون - کافکا - کلاس MyKafka - کلاس Main - پردازش داده

نشانی
□□□□:https://www.shabakeh-mag.com/workshop/15619/%D8%B1%D8%A7%D9%87%D9%86%D9%85%D8%A7%DB%8C-%D8%A8%D9%87%E2%80%8C%DA%A9%D8%A7%D8%B1%DA%AF%DB%8C%D8%B1%DB%8C-kafka-python-%D8%A8%D8%B1%D8%A7%DB%8C-%D9%BE%D8%B1%D8%AF%D8%A7%D8%B2%D8%B4-%D8%B3%D8%B1%DB%8C%D8%B9-%D8%AF%D8%A7%D8%AF%D9%87%E2%80%8C%D9%87%D8%A7