

Python + Kafka

Библиотеки

- confluent_kafka от разработчиков
- aiokafka асинхронная библиотека

confluent_kafka

AdminClient

Модуль автоматизации административных задач.

Настройки:

```
from confluent_kafka.admin import AdminClient, NewTopic

config = {
    'bootstrap.servers': '192.168.1.195:29092'
}

admin_client=AdminClient(config)
```

Создание темы:

```
def create_partition():
    topic = 'test_topic'
    new_topic = NewTopic(topic, num_partitions=2, replication_factor=1)
    futures = admin_client.create_topics([new_topic])
    for topic_name, future in futures.items():
        try:
            future.result()
            print(f"Created {topic_name} successfully.")
        except Exception as e :
            print(f'Error in topic {topic_name} creation: ', e)
```

Топик в списке появляется не сразу, а где-то через 30 секунд.

Список тем:

```
def list_topics():
    metadata = admin_client.list_topics(timeout=10)
    print("Существующие топики:", list(metadata.topics.keys()))
```

consumer.py

```
from confluent_kafka import Consumer, KafkaException

config = {
    'bootstrap.servers': '192.168.1.195:29092',
    'group.id' : 'my_group'
}

consumer = Consumer(config)

if __name__ == '__main__':
    topic = 'test_topic'
    timeout_s = 1
    consumer.subscribe([topic])

    try:
        while True:
            msg = consumer.poll(timeout=timeout_s)
            if msg is None:
                continue
            if msg.error():
                raise KafkaException(msg.error())
            else:
                print(f'Input: {msg.value().decode()}')
    finally:
        consumer.close()
```

Если consumer в одной группе и количество partitions не меньше количества consumers, то сообщения распределяются между ними. Если partitions будет меньше consumers (которые в одной группе), то оставшаяся часть будет простаивать.

Функция consumer.subscribe получает список топиков, то есть одним consumer можно подписаться на несколько.

Если при двух и более partitions для одного из partitions отваливается consumer то Kafka автоматически проводит ребалансировку в пределах группы. После подключения опять произойдет ребалансировка.

producer.py

```
import time
from confluent_kafka import Producer

def delivery_report(err, msg):
    if err is not None:
        print(f"Message failed: {err}")
        raise err
    else:
        print(f'Message {msg.value()} in topic {msg.topic()} delivered')

config = {
    'bootstrap.servers': '192.168.1.195:29092',
    'message.timeout.ms' : 5000
}

producer = Producer(config)

def send_message(topic_: str, message_: str):
    producer.produce(topic_, value=message_, callback=delivery_report)
    producer.flush()

if __name__ == '__main__':
    topic = 'test_topic'
    message = 'Producer 1'

    i = 0
    while True:
        send_message(topic, f'{message} # {i}')
        i += 1
        time.sleep(1)
```

Распределение сообщений по ключам

Для распределения по консюмерам используется хэш функция от ключа и количества партиций. При коротких ключах может быть коллизия. Довольно неприятная и

сложнообходимая ситуация. Проще указать номер partition (int).

```
import time
from confluent_kafka import Producer

def delivery_report(err, msg):
    if err is not None:
        print(f"Message failed: {err}")
        raise err
    else:
        print(f'Message {msg.value()} in topic {msg.topic()} delivered')

config = {
    'bootstrap.servers': '192.168.1.195:29092',
    'message.timeout.ms' : 5000
}

producer = Producer(config)

def send_message(topic_: str, message_: str, part_: int | None = None):
    producer.produce(topic_, partition=part_, value=message_, callback=delivery_report)
    producer.flush()

if __name__ == '__main__':
    topic = 'test_topic'
    message = 'Producer 1'

    i = 0
    while True:
        part1 = 0
        part2 = 1
        send_message(topic, message_=f'key={key1} {message} # {i}', part_=part1)
        send_message(topic, message_=f'key={key2} {message} # {i}', part_=part2)
        i += 1
        time.sleep(1)

#===== consumer
=====

from confluent_kafka import Consumer, KafkaException
```

```
config = {
    'bootstrap.servers': '192.168.1.195:29092',
    'group.id' : 'my_group'
}

consumer = Consumer(config)

if __name__ == '__main__':
    topic = 'test_topic'
    timeout_s = 1
    consumer.subscribe([topic])

    try:
        while True:
            msg = consumer.poll(timeout=timeout_s)
            if msg is None:
                continue
            if msg.error():
                raise KafkaException(msg.error())
            else:
                print(f'Input: {msg.value().decode()}')
    finally:
        consumer.close()
```

Транзакции

В config для consumer добавляется элемент 'enable.auto.commit': False

И затем consumer.commit()

Сериализация данных

Через json

```
while True:
    obj = {
        'id': i,
        'value': randomword(8),
        'timestamp': datetime.now().isoformat(timespec='seconds'),
    }
```

```

send_message(topic, json.dumps(obj))
i += 1
time.sleep(1)

=====

    str_msg = msg.value().decode('utf-8')
    obj = json.loads(str_msg)
    print(f'id={obj.get("id")}, timestamp={obj.get("timestamp")}')

```

aiokafka

consumer.py

```

from aiokafka import AIOKafkaConsumer
import asyncio

async def consume():
    consumer = AIOKafkaConsumer(
        'test_first_one',
        bootstrap_servers='192.168.1.195:29092'
    )
    await consumer.start()
    try:
        async for msg in consumer:
            print(f"Consumed message {msg.value.decode()}")
    finally:
        await consumer.stop()

if __name__ == "__main__":
    asyncio.run(consume())

```

producer.py

```

from aiokafka import AIOKafkaProducer
import asyncio

```

```
async def send():
    producer = AIOKafkaProducer(
        bootstrap_servers="192.168.1.195:29092"
    )
    topik = 'test_first_one'
    message = 'Hello from python!'
    await producer.start()
    i = 0
    try:
        while True:
            bytes_msg = f'{message} {i}'.encode('utf-8') # convert string into byte
            await producer.send_and_wait(topik, bytes_msg)
            i += 1
            print(f'Message {i} sent.')
            await asyncio.sleep(1)
    finally:
        await producer.stop()

if __name__ == '__main__':
    asyncio.run(send())
```