

ozon{tech



# Kafka и Go

Сириус (Сергей Ивлиев)

Старший разработчик / Лид команды автоматической  
модерации товаров и контента

к.т.н., старший преподаватель кафедры ПМИИ НИУ МЭИ

Москва 2021



# План занятия

1. Клиенты Kafka
2. Разворачивание локального кластера для разработки
3. Издатель-подписчик
4. Лог
5. Saga
6. Мониторинг

1

Клиенты Kafka


# Клиенты Kafka

Sarama (7.8k ★, <https://github.com/Shopify/sarama>), который на сегодняшний день является самым популярным.

API предоставляет низкоуровневые концепции протокола Кафки.

Также передает все значения в качестве указателей, что приводит к большому количеству динамических выделений памяти, более частым сборкам мусора и более высокому использованию памяти. Но в некоторых кейсах это не афферктит производительность.

# Клиенты Kafka

Confluent-kafka-go (3k , <https://github.com/confluentinc/confluent-kafka-go>) – это оболочка на основе sgo вокруг librdkafka, что означает, что она вводит зависимость библиотеки C от всего кода Go, использующего пакет.

Он имеет гораздо лучшую документацию, чем sarama, но не поддерживает контексты Go.

# Клиенты Kafka

goka (1.8k ★, <https://github.com/lovoo/goka>) - это более поздний клиент Kafka для Go, который фокусируется на определенном шаблоне использования.

Он предоставляет абстракции для использования Кафки в качестве шины передачи сообщений между службами, а не упорядоченного журнала событий, но это не типичный случай использования Кафки для нас в сегменте.

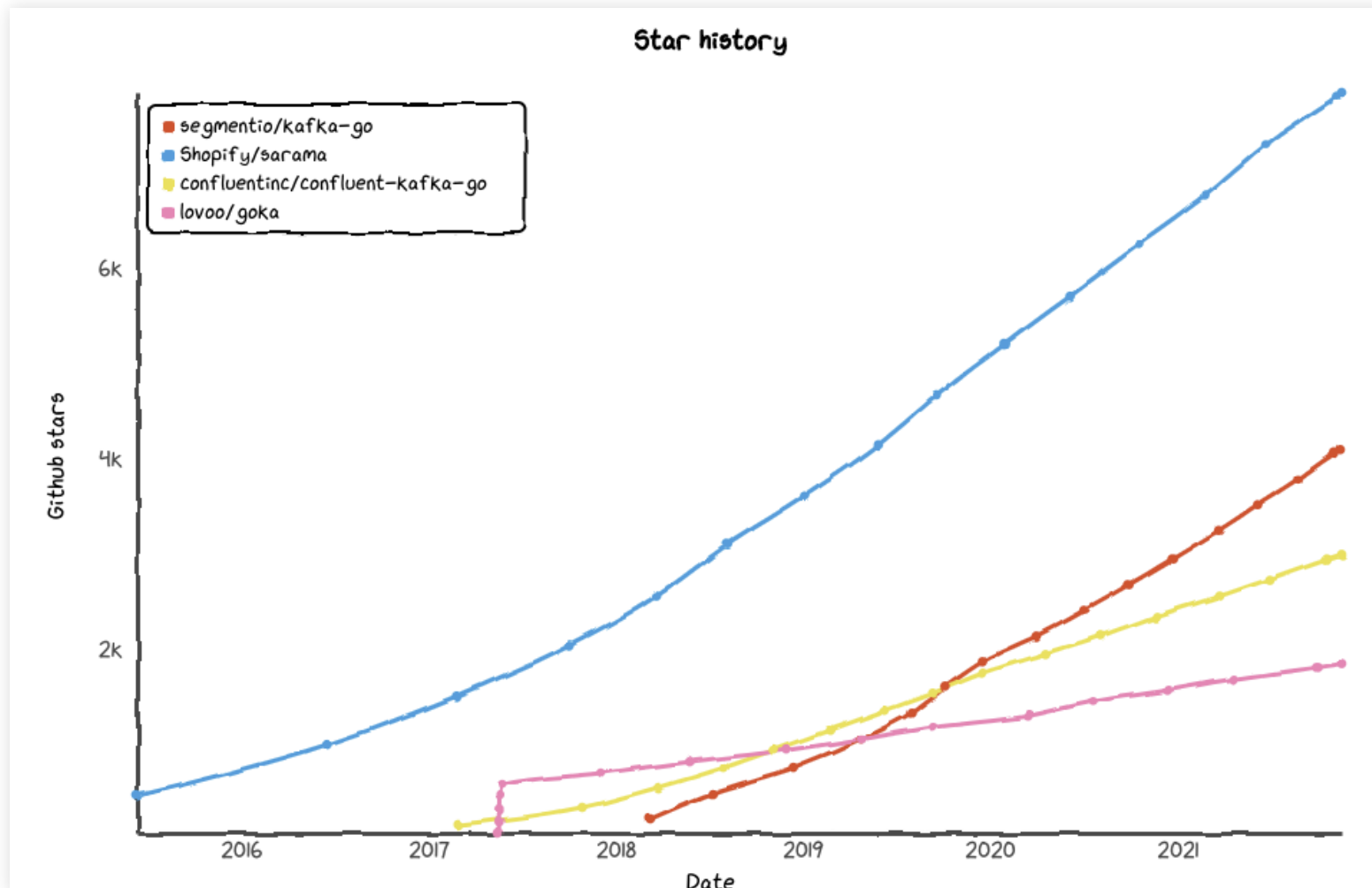
Пакет также зависит от Sarama для всех взаимодействий с Kafka.

# Клиенты Kafka

kafka-go (4.1k ★, <https://github.com/segmentio/kafka-go>) - комбинирует как низкоуровневые абстракции, так и высокоуровневые, написан с учётом специфики Go.



# Клиенты Kafka



# Клиенты Kafka

GUI клиенты:

1. UI for Apache Kafka (<https://github.com/provectus/kafka-ui>) – это бесплатный веб-интерфейс с открытым исходным кодом для мониторинга кластеров Apache Kafka и управления ими.
2. Offset Explorer (ex. Kafka Tool, <https://www.kafkatool.com/>) – десктопное приложение для взаимодействия с кластерами Kafka (платное).
3. kcat (ex. Kafka Cat, <https://github.com/edenhill/kcat>) – консольное приложение, не требующее JVM, о котором можно думать как о netcat для Kafka.

# 2

Разворачивание локального кластера  
для разработки

# Разворачивание локального кластера для разработки

Разумеется, через docker.

```
version: '3'

services:
  kafka-ui:
    image: provectuslabs/kafka-ui:latest
    ports:
      - "8080:8080"
    environment:
      - KAFKA_CLUSTERS_0_NAME=local
      - KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS=kafka-1:9092,kafka-2:9092,kafka-3:9092
      - KAFKA_CLUSTERS_0_ZOOKEEPER=zookeeper:2181

  zookeeper:
    image: zookeeper
    ports:
      - "2181:2181"
    environment:
```

# Разворачивание локального кластера для разработки

Требуется запускать zookeeper раньше Kafka. Т.е. сперва:

```
docker-compose up zookeeper
```

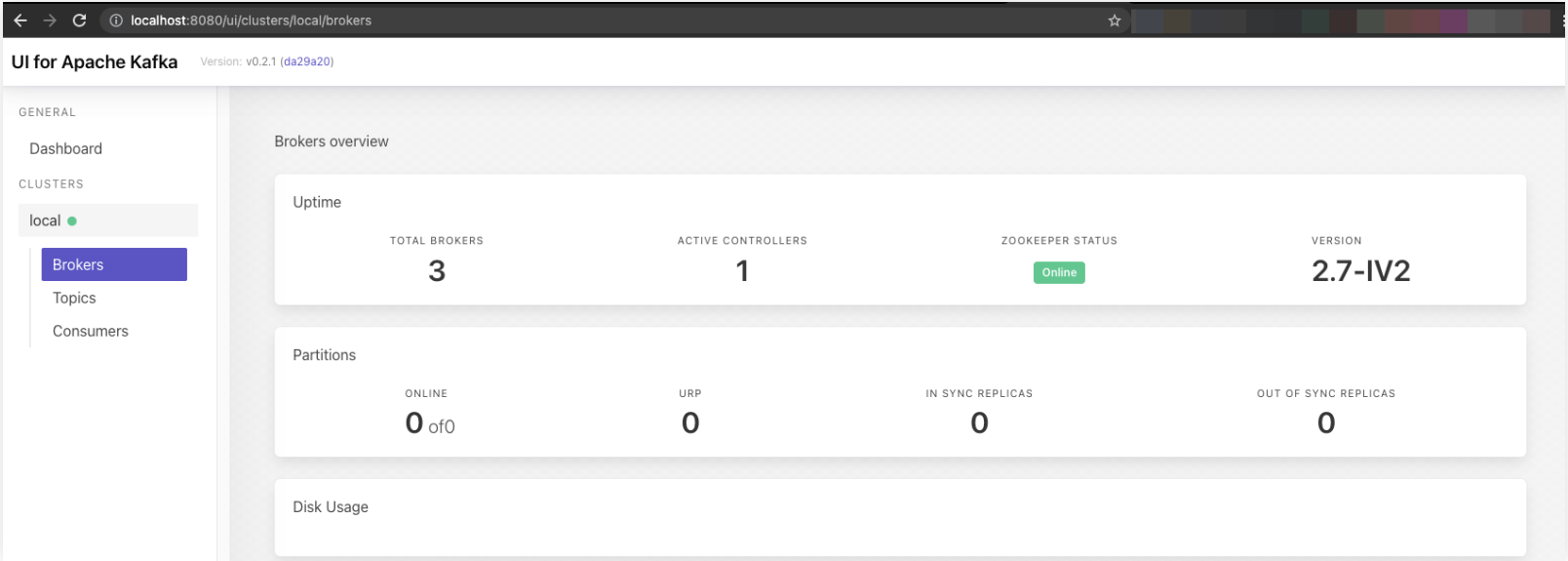
ПОТОМ:

```
docker-compose up kafka-ui kafka-1 kafka-2 kafka-3
```

Для подключения нужно знать адреса всех брокеров. Для данного файла это:

- 127.0.0.1:9095
- 127.0.0.1:9096
- 127.0.0.1:9097

# Разворачивание локального кластера для разработки



3

Издатель-подписчик

# Издатель-подписчик

Пусть у нас есть сервис заказов и в нём происходит подтверждение факта совершения заказа.

Нам надо передавать данные об этом в разные сервисы:

1. Аналитика, которая считает что и как продаётся
2. Сервис контроля остатков, в который отправляется информация, что товар был куплен

Это можно сделать, используя паттерн издатель-подписчик.

N.b.: случай, требующий транзакционности, будет рассмотрен далее.



# Издатель-подписчик

Издатель-подписчик (англ. publisher-subscriber или англ. pub/sub) — поведенческий шаблон проектирования передачи сообщений.

В нём отправители сообщений, именуемые издателями (англ. publishers), напрямую не привязаны программным кодом отправки сообщений к подписчикам (англ. subscribers).

Вместо этого сообщения делятся на классы и не содержат сведений о своих подписчиках, если таковые есть.

Аналогичным образом подписчики имеют дело с одним или несколькими классами сообщений, абстрагируясь от конкретных издателей.

# Издатель-подписчик

В модели издатель-подписчик подписчики обычно получают только подмножество всех опубликованных сообщений.

Процесс отбора сообщений для получения и их обработка называется фильтрацией.

Существуют две основных формы фильтрации:

1. основанная на теме (англ. topic);
2. основанная на содержанием.

# Издатель-подписчик

В системе, основанной на теме, сообщения публикуются в «темах» или именованных логических каналах.

Подписчики в таких системах будут получать все сообщения, опубликованные в темах, на которые они подписались.

Все подписчики, подписавшиеся на одну и ту же тему, будут получать те же самые сообщения.

Издатель отвечает за определение классов сообщений, на которые подписываются подписчики.

# Издатель-подписчик

В системе, основанной на содержимом, сообщения доставляются подписчикам только в том случае, если атрибуты или содержимое этих сообщений допускаются подписчиком.

В данной системе подписчик отвечает за классификацию сообщений.

# Создаём топик

[All Topics](#) / [New Topic](#)

Topic Name \*

orders

Number of partitions \*

3

Replication Factor \*

3

Min In Sync Replicas \*

2

Cleanup policy

✓ Delete

Compact

Time to retain data (in ms)

604800000

12h

1d

2d

7d

4w

7d

Max size on disk in GB

Not Set

Maximum message size in bytes \*

1000012

Add Custom Parameter +

Send

## Про Cleanup policy

1. Delete – удаляем старые сообщения
2. Compact – сжимаем старые сообщения  
(<https://kafka.apache.org/documentation/#compaction>)

Можно использовать оба режима, т.е. по одному правилу сжимать сообщения, а по другому – удалять.

## Изменение offset для Consumer Group

Через kafka-consumer-groups (<https://docs.cloudera.com/runtime/7.2.10/kafka-managing/topics/kafka-manage-cli-cgroups.html>):

```
kafka-consumer-groups --bootstrap-server {url} \  
--topic {topic} \  
--group {consumer} \  
--reset-offsets --to-datetime 2020-11-11T00:00:00.000+0900 \  
--execute
```

# Изменение offset для Consumer Group

Через zookeeper shell (<https://zookeeper.apache.org/doc/r3.3.3/zookeeperStarted.html>):

```
./zookeeper-shell.sh localhost:2181  
set /consumers/consumer_group_id/offsets/topic/0 10000
```

4

Лог





# Что такое лог?

А вообще что такое лог?

Лог – это структура данных, в которую можно добавлять только в конец.

Из лога можно прочесть определённое сообщение, только зная его смещение относительно начала лога.

С точки зрения потребителя – брокеры сообщений представляют возможность работы с логами.

5

Saga

# Saga

*Использованы материалы Ивана Стрелкова, полная версия статьи будет на Хабре (скоро)*

Проблема: надо как-то оперировать транзакциями в распределённой системе.  
Разумеется, один из шагов транзакции может упасть.

Решение: Saga

# Saga

Паттерн Saga изначально представлял собой способ реализации транзакций по нескольким микросервисам.

А именно — мы разбиваем процесс на ряд шагов и эти шаги выполняются последовательно. Если какой-то из шагов пошёл не так, то мы выполняем ряд откатывающих действий до тех пор, пока не откатим все выполненные действия.

# Saga

В оригинальном представлении атомарные шаги (транзакции) бывают трех видов:

- Компенсируемые транзакции (compensatable transactions) — т.е. транзакции, для которых определена транзакция, откатывающая соответствующие изменения.

# Saga

- Поворотная транзакция (pivot transaction) — это транзакция, после успешного выполнения которой Saga обязательно должна выполниться до конца. И наоборот — если поворотная транзакция закончилась ошибкой, то все выполненные шаги саги подлежат откату.
- Повторяющиеся транзакции (retriable transactions) — транзакции, которые будут повторяться до тех пор, пока не пройдут успешно

# Saga

Сами же саги бывают хореографические и оркестрационные:

- В оркестрационной саге есть оркестратор – компонент, который знает про все имеющиеся шаги. Оркестратор последовательно выполняет заранее известный набор шагов, а также откатывает.
- В хореографической саге каждый шаг приводит только к публикации события в очередь. При этом следующие шаги выполняются неявно на стороне других сервисов.




# Saga

Пишем Saga на основе формирования заказа.

6

Мониторинг



Кафка достаточно  
стабильна



Ваши сервисы и  
клиенты Kafka – нет

# Пример

Верхний уровень воды (high watermark offset) – это смещение (offset) последнего сообщения, которое было скопировано успешно на все реплики.

В редких случаях сбоя или по другой причине Кафка может возвращать батчи с нулевым размером, даже если не был достигнут верхний уровень воды.

В этом случае сарама просто пытается перечитывать эту партию снова и снова и застревает.

В этом случае нет никакого решения, кроме пропуска этого пакета.

Librdkafka может обрабатывать пропуск этих случаев с предупреждением.

<https://github.com/Shopify/sarama/pull/2057>

# Пример

При изменении количества партиций в топике в сервисе-подписчике консьюмеры не подключились к новым топикам.

## Литература

1. [Иван Пономарёв — Apache Kafka: Что это и как она изменит архитектуру вашего приложения](#)