129.53
Рейтинг

Слёрм

Учебный центр для тех, кто работает в IT

Подписаться

zubarek 28 фев 2023 в 17:11

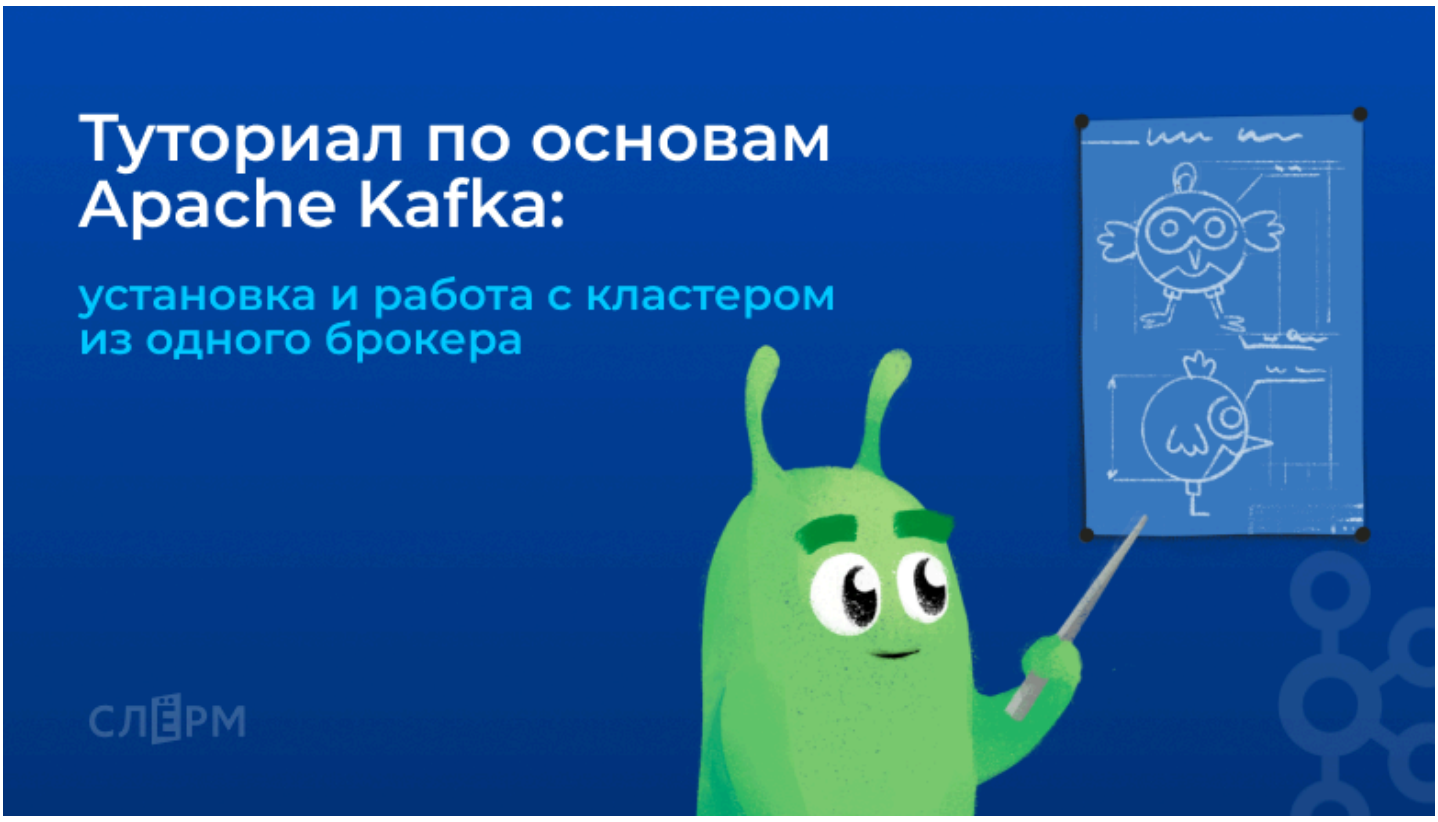
Туториал по основам Apache Kafka: установка и работа с кластером из одного брокера

14 мин 48K

Блог компании Слёрм, IT-инфраструктура*, DevOps*

Туториал

Собрали туториал по Кафке для того, чтобы вы могли быстро научиться с ней работать, получить фундаментальные навыки и безболезненно приступить к более высокоуровневому функционалу в дальнейшем.



*Материал записан на основе уроков курса **Apache Kafka База**. Версия для Кафки с ZooKeeper. Видеозапись смотрите здесь .*

Запуск Kafka

Скачиваем архив с Кафкой:

```
wget https://archive.apache.org/dist/kafka/2.7.0/kafka_2.13-2.7.0.tgz

tar -xzf kafka_2.13-2.7.0.tgz

cd kafka_2.13-2.7.0
```

Приступим. Первым делом поднимаем ZooKeeper. Нужна папка ./bin: она содержит скрипты для запуска брокера, конфигурации топиков, партиций и реконфигурации кластера и т. д. Запускаем скрипт zookeeper-server-start.sh и передаем ему конфигурационный файл zookeeper.properties из папки уровнем выше ../config. Этот файл хранит в себе набор базовых переменных, чтобы быстро в тестовом режиме поднять ZooKeeper и ноду Кафки.

Выглядит это так: ./bin/zookeeper-server-start.sh config/zookeeper.properties

Теперь запускаем брокер Кафки. Для этого в той же папке ./bin запускаем скрипт kafka-server-start.sh и передаем ему конфигурационный файл server.properties из папки ../config. Здесь также содержится минимальный набор конфигураций, который позволяет запустить брокер.

```
./bin/kafka-server-start.sh config/server.properties
```

Готово. Мы подняли свой маленький кластер, где можно записать и прочесть данные.

Запись и чтение

Для успешной записи данных нужно создать топик. Воспользуемся скриптом из уже известной папки ./bin, а именно kafka-topics.sh, которому передадим опцию --create и название топика, в нашем случае — --topic registrations. Также мы должны передать --bootstrap-server, к которому скрипт приконнектится, чтобы сделать запрос. В нашем случае брокер всего 1, поэтому мы передаем его адрес localhost и порт, который по умолчанию слушает Кафка, — 9092.

Команда выглядит так: `./bin/kafka-topics.sh --create --topic registrations --bootstrap-server localhost:9092`

Итак, мы создали топик registrations-0, где 0 — это идентификатор партиции. Топик был создан с одной партицией, потому что мы не задали их количество. Проверить это можно с помощью такой команды: ./kafka-topics.sh --describe --topic registrations --bootstrap-server localhost:9092

Видим количество партиций, реплик; чуть ниже — описание состояния партиций в этом топике. У нас одна партиция с идентификатором 0, лидером которой выступает брокер с id 0.

Топик готов, можем записать в него сообщение. К счастью, Кафка идет с консольной утилитой, которая позволяет нам это сделать: ./kafka-console-producer.sh, лежит опять же в папке ./bin. Передаем ей название топика, куда хотим записать данные, и тот же --bootstrap-server. Немного

ожидания, и консольный продюсер готов к записи сообщения. Мы можем передать все что угодно. Например, hello world и hello slurm. Выглядеть это все должно так:

```
 ./bin/kafka-console-producer.sh --topic registrations --bootstrap-server
localhost:9092
```

>Hello world!

>Hello Slurm!

Теперь надо прочесть эти сообщения. Для этого существует консольный консьюмер (лежит в той же самой папке) ./kafka-console-consumer.sh, куда мы точно так же передаем название топика и --bootstrap-server:

```
 ./bin/kafka-console-consumer.sh --topic registrations --bootstrap-server
localhost:9092
```

Теперь мы, по идее, должны увидеть записанное сообщение. Но ничего не происходит.

С этой проблемой сталкиваются многие, кто начинает использовать Кафку. Это не значит, что сообщения потерялись, или что-то не работает. Все проще: *консьюмер Кафки по умолчанию начинает читать данные с конца топика в тот момент, когда он запустился* (см. настройку auto.offset.reset). Поэтому, чтобы прочитать данные, записанные ДО старта консьюмера, нужно переопределить эту конфигу.

Закрываем консьюмер и вызываем его повторно, но с другой настройкой. Команда все та же, но с дополнением --consumer-property, которому мы передаем настройку auto.offset.reset=earliest. Значение earliest показывает, что чтение записей будет начинаться с самого раннего доступного сообщения. Вот так:

```
 ./bin/kafka-console-consumer.sh --topic registrations --bootstrap-server
localhost:9092 --consumer-property auto.offset.reset=earliest
```

Запускаем консьюмер — и видим те сообщения, которые записали ранее! *Примечание. Помимо* --consumer-property *существует шорткат* --from-beginning*, который делает то же самое.*

Давайте вернемся в лог нашего брокера и посмотрим на пару последних сообщений. Обратите внимание на название группы, с которой коннектится этот консьюмер. Вы увидите, что индентификатор консольного консьюмера случайно сгенерирован: префикс console-consumer, далее — случайный набор цифр, причем всегда разный. Каждый раз, когда вы запускаете консьюмер, он случайно генерит группу.

Попробуем запустить консьюмер и явно передать группу, с которой хотим прочитать сообщение. Для этого к команде ./kafka-console-consumer.sh --topic registrations --bootstrap-server localhost:9092 auto.offset.reset=earliest мы передаем еще одну проперти, которая выглядит как --group slurm. Slurm в нашем случае — название группы, но ее можно назвать как угодно. Целиком так:

`./bin/kafka-console-consumer.sh --topic registrations --bootstrap-server localhost:9092 --consumer-property auto.offset.reset=earliest --group slurm`

После запуска консьюмера мы увидим те же записанные ранее сообщения. При этом, в логе брокера отобразится, что консьюмер подключился с группой slurm, которую мы передали. Закрываем консьюмер, перезапускаем его снова той же командой. Сообщения опять пропали!

Почему? *Консьюмер группы в Кафке может коммитить свои оффсеты (свою позицию) для какой-то партиции, которую он уже прочитал.* Чтобы при перезапуске продолжить обработку с этой позиции. Именно это поведение мы здесь и наблюдаем. Хотя лучше лишний раз проверить.

Закрываем консьюмер и используем скрипт `./bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --group slurm --describe`

Сейчас ни один инстанс группы не живет, но мы все равно можем проверить ее сохраненный стейт. Мы видим, что никакого активного члена группы у нас нет (Consumer group ‘slurm’ has no active members), и это правильно. Еще мы видим, что эта группа в топике registrations в партиции 0 сохранила свою позицию на offset-е 2 (CURRENT-OFFSET). Именно этот offset является концом топика. LAG у нас 0, значит консьюмер полностью прочитал все сообщения и не лагает. Получается, что наш консольный консьюмер автоматически закоммитил свою позицию.

Что можно сделать? Можем сбросить позицию консьюмера на начало. Разумеется, если вы используете клиенты, у вас будет целый набор инструментов для удобного контроля позиции своего консьюмера в любой момент.

В нашем случае, чтобы сбросить консьюмера на начало топика, воспользуемся скриптом `./bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --group slurm --reset-offsets --to-earliest --topic registrations --execute`

Ждем и видим, что позиция была сброшена на 0 (NEW-OFFSET). Теперь при новом запуске консьюмера мы снова увидим два наших сообщения.

Topic Retention, часть 1

Этот механизм служит основным способом удаления данных из Кафки. Мы можем включить его по времени или по размеру партиции. Рассмотрим retention по времени. На данном этапе в нашем топике этот механизм не настроен, поэтому данные будут храниться вечно (до тех пор, пока диск на брокере не заполнится до предела).

Для начала изменим одну из настроек брокера, чтобы облегчить себе жизнь: нам будет видно, что происходит с данными после включения retention. Останавливаем брокер, если он уже запущен. Копируем конфигурационный файл, с которым мы изначально запустили этот брокер (назовем его slurm-server.props): cp config/server.properties config/slurm-server.props

Открываем этот конфиг. Настройка, которую будем менять, называется log.retention.check.interval.ms. Она диктует частоту, с которой удаляющий данные с диска тред (LogCleaner) проверяет retention. Значение по умолчанию — 5 минут. Для production-систем это

замечательно. Однако мы будем менять конфиги, поэтому нам хочется видеть отклик быстрее. Поменяем значение на 1 секунду: log.retention.check.interval.ms=1000

Таким образом LogCleaner будет проверять данные для возможного удаления раз в секунду. Сохраняем файл и запускаем сервер с новой конфигурацией. Сделано.

Теперь включим retention у топика, а также заальтерим одну из конфигурационных опций — retention.ms. Выставим значение 60000 (одна минута). Для этого воспользуемся скриптом

```
 ./bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type topics --
entity-name registrations --alter --add-config retention.ms=60000
```

Данные мы записали достаточно давно, и чекер работает каждую секунду, поэтому просто открываем консольный консьюмер ./kafka-console-consumer.sh, передаем ему все тот же --bootstrap-server и название топика (--topic registrations). Не забываем добавить --from-beginning, чтобы точно удостовериться в отсутствии других данных. Видно, что никаких данных наш консьюмер не отдает. Делаем вывод, что никаких данных больше не осталось. Чисто.

Проведем эксперимент. Скажем Кафке удалять данные из топика после 10 секунд:

```
 ./bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type topics --
entity-name registrations --alter --add-config retention.ms=10000
```

Теперь запустим консольного продюсера, чтобы он считывал все новые лайны из файла и перекидывал их в Кафку. Скрипт такой:

```
 touch /tmp/data && tail -f -n0 /tmp/data | ./bin/kafka-console-producer.sh --
topic registrations --bootstrap-server=localhost:9092 --sync
```

Он создает файл /tmp/data, тейлит этот файл и передает весь output консольному продюсеру, чтобы тот писал эти сообщения в наш топик registrations. Теперь откроем другое окно и запустим еще один скрипт:

```
 for i in $(seq 1 3600); do echo $"{i}" >> /tmp/data; sleep 1; done
```

Он будет каждую секунду аппендить новые лайны в этот файл: test1, test2, test3 и так далее до 3600. Все лайны будут автоматически передаваться нашему продюсеру. Открываем третье окно и запускаем консольный консьюмер, чтобы посмотреть, какие сообщения хранятся сейчас в топике:

```
 ./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
registrations --from-beginning
```

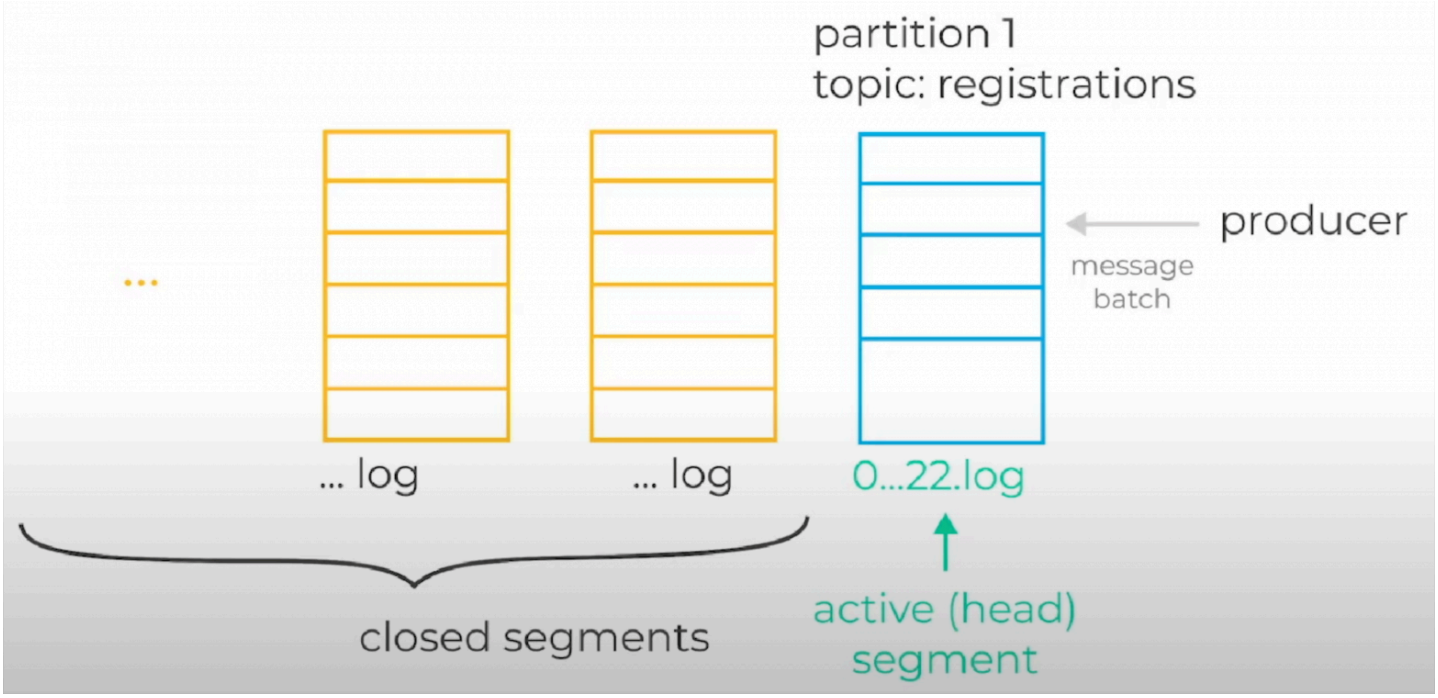
Мы задали настройку, чтобы наши сообщения удалялись после 10 секунд. Также мы отправляем test1, test2, test3 и далее в наш топик registrations раз в секунду.

Видим следующее: в топике до сих пор хранятся все сообщения, несмотря на то, что прошло уже больше 10 секунд. Более того, мы явно указали Кафке, что чекер LogCleaner-а должен проверять

данные на удаление раз в секунду. Давайте запустим консьюмер еще раз. Мы снова видим все сообщения в топике. Что не так?

Давайте разбираться. Нам нужно заглянуть во внутреннюю структуру данных партиции и понять: как именно Кафка сохраняет данные на диск.

Структура партиции



Разбираемся, как Кафка хранит данные на диске. Партиции состоят из набора файлов, которые называются сегментами. Данные, которые продюсеры присылают брокеру, сохраняются в открытый или головной сегмент партиции. Через некоторое время, согласно некоторому набору правил, он роллапится (закрывается). После этого открывается новый сегмент. Закрытые сегменты хранятся на диске, но при этом в них никогда уже не происходит запись (они становятся полностью иммутабельными). Важно понимать, что LogCleaner Кафки удаляет данные исключительно посегментно. То есть, он удаляет файлы целиком. Для того чтобы LogCleaner понял, можно удалять файл или нет (если мы говорим о retention по времени), он производит следующий простой набор операций:

- находит максимальный таймстамп сообщения внутри одного сегмента;
- находит разницу между максимальным таймстампом и текущим временем;
- определяет, больше ли эта разница во времени, чем заданный конфигурационной опцией retention.ms;
- если разница больше, то сегмент уже старый, его можно удалить.

До этого мы писали сообщения continuously в открытый сегмент партиции топика registrations, поэтому постоянно увеличивали максимальный таймстамп (раз в секунду), тем самым не давая LogCleaner-у удалить этот сегмент.

Напомним, что retention.ms у нас был выставлен в 10 секунд. Разница во времени никогда не превысит это заданное значение, потому что мы постоянно дописываем сообщения. Если бы мы остановили продюсера и подождали 10 секунд, то увидели бы, что данные удалены. Плюс, если бы текущий сегмент закрылся, то оказавшиеся в нем данные очень быстро удалились.

Но эта операция роллапа не происходила, потому что изначально мы не изменили дефолтные настройки. Их две:

1. segment.ms — период роллапа сегмента после его открытия, 1 неделя по умолчанию
2. segment.bytes — максимальный размер сегмента, 1 ГБ по умолчанию.

Понятно, что мы не написали данных на 1 ГБ и точно не прождали неделю, чтобы дождаться retention-а. В этом случае мы можем выйти из ситуации двумя способами: выставить segment.bytes на очень маленькое значение (пару КБ) или сказать segment.ms роллапить сегмент чаще, чем раз в неделю (через 10 секунд, например).

Важно сказать, что обе эти настройки работают одновременно по правилу *ИЛИ*, поэтому контролировать их можно (и нужно) одновременно.

Мы еще не затрагивали retention по байтам, но он очень простой — это максимальный размер партиции на диске в байтах. Этой настройкой приходится пользоваться не так уж и часто, потому что сложно сказать, как долго хранятся данные. Это сильно зависит от того, с какой скоростью записываются данные на диск. Может быть, сегодня продюсер отправляет по 10 КБ в секунду, а в дальнейшем объем данных вырастет, и они начнут удаляться быстрее при условии сохранения старых настроек. Но есть и плюс: retention по байтам защищает ваших брокеров от переполнения данными.

К слову, retention.ms и retention.bytes также работают по правилу *ИЛИ*, поэтому их можно задать одновременно. Допустим: мы сохраняем данные минимум на неделю, а еще ограничиваем максимальный размер партиции в 1 ТБ.

Еще один момент: большая часть настроек Кафки может быть реализована на двух уровнях.

Уровень брокера или сервера содержит дефолты всех настроек и часто имеет префикс log. Например, log.retention.ms — это глобальный дефолт retention-а для всех топиков, который задается в конфигурационном файле сервера server.properties. **Topic-level конфиги** — это оверрайды для отдельных топиков, которые мы задавали через команду kafka-configs.sh. Их значения хранятся в ZooKeeper.

Пользоваться можно любыми из настроек. Работают они, по большому счету, одинаково. Практический совет: можно выставить разумные дефолтные настройки на уровне брокера, а уже для конкретных топиков задавать индивидуальные настройки. Полный перечень настроек ищите на сайте самой Кафки: https://kafka.apache.org/documentation/#configuration.

Теперь переходим к борьбе с проблемой неудаляющихся данных, с которой столкнулись ранее.

Topic Retention, часть 2

Снова открываем консольный консьюмер (--topic registrations), останавливая при этом продюсер. Через 15 секунд все сообщения из топика будут удалены. Мы знаем, как хранятся файлы, поэтому давайте заглянем в папку и узнаем, что там лежит. По умолчанию хранение происходит в папке /tmp/kafka-logs/. Здесь куча разных папок, но нас интересует registrations 0 (топик registrations, партиция 0).

```
ls -la /tmp/kafka-logs/registrations-0
```

Здесь есть только 1 файл, но он абсолютно пустой, потому что все данные из него были удалены.

Приступим к настройке. В первую очередь, поменяем segment.ms у нашего топика: зададим override и скажем, что хотим роллапить сегменты для этого топика раз в 10 секунд. Для этого воспользуемся командой:

```
./bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type topics --entity-name registrations --alter --add-config segment.ms=10000
```

После этого запускаем консольного продюсера, который тейлит файл tmp/data, а затем — форлуп, который генерит сообщения раз в секунду. Запись началась!

Прежде чем запускать консольного консьюмера, заглянем в папку /tmp/kafka-logs/ и увидим, что динамика есть. Файлы роллапятся. Файл с самым большим оффсетом — наш головной сегмент. Процесс идет таким образом: старые сегменты закрываются, новые открываются. Старые сегменты при этом помечаются как deleted, затем еще один бэкграунд тред полностью удаляет их с диска.

Открываем консольный консьюмер, чтобы проверить, что данные удаляются согласно заданным настройкам. Мы перезапустили форлуп, поэтому при неполадках видели бы сообщения test1, test2 и т. д. Если все происходит правильно, видим, что сообщения уже идут какое-то время (в нашем случае — test101, test102 и далее). Более ранних сообщений в этом топике нет, поскольку все роллапится согласно заданным правилам. Перезапускаем консьюмер еще раз, чтобы убедиться наверняка. Видим сообщения test121, test122 и т. д.

Log Compaction

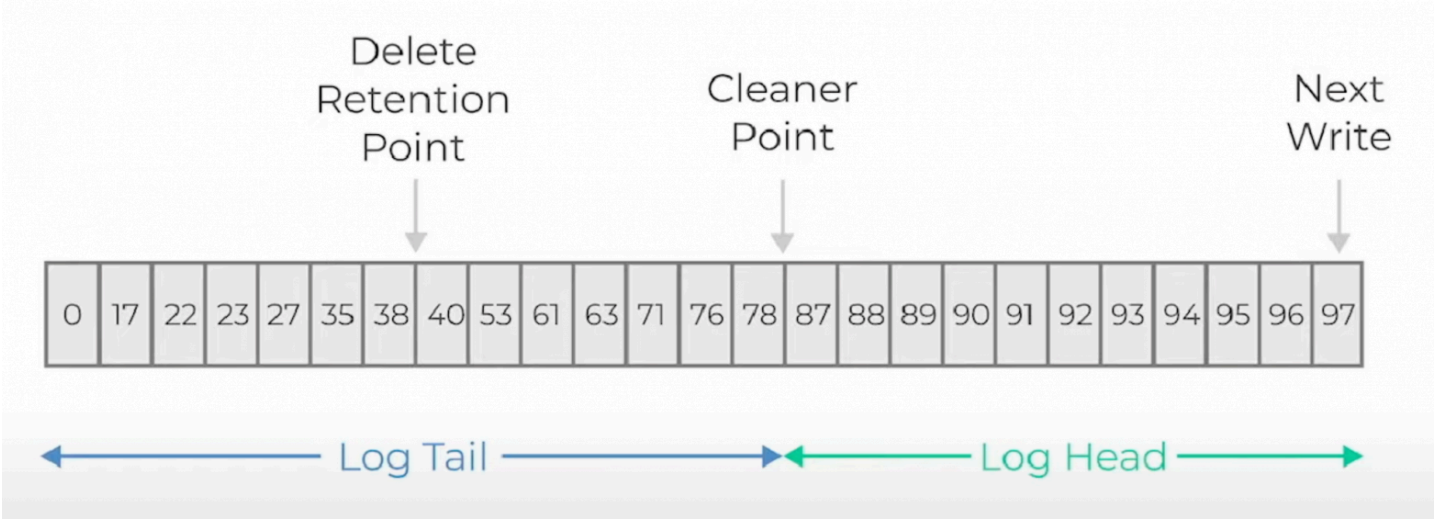
Помимо функционала удаления данных по retention.ms и retention.bytes, которые мы рассмотрели выше, Кафка предоставляет еще один механизм удаления данных — log compaction или сжатие данных в партиции. Этот механизм использует ключи сообщений, чтобы решить: удалять данные или нет.

В этом примере мы видим, что в партицию были последовательно записаны три сообщения. Первое было записано с ключом slurm, два последующих — с ключом foo. После завершения compaction в партиции остались два сообщения: с ключом slurm и ключом foo и его последним значением.

Помимо этого compaction позволяет выборочно удалять данные из партиции.

На этой картинке мы видим, что третьим отправили сообщение с Value: NULL и ключом foo. После завершения compaction в данном случае в партиции осталось только сообщение с ключом slurm. Оставшиеся два сообщения были полностью удалены из нашей партиции. Это произошло потому, что сообщение с Value: NULL (т. н. delete marker) тоже распознается Кафкой, как необходимое к удалению.

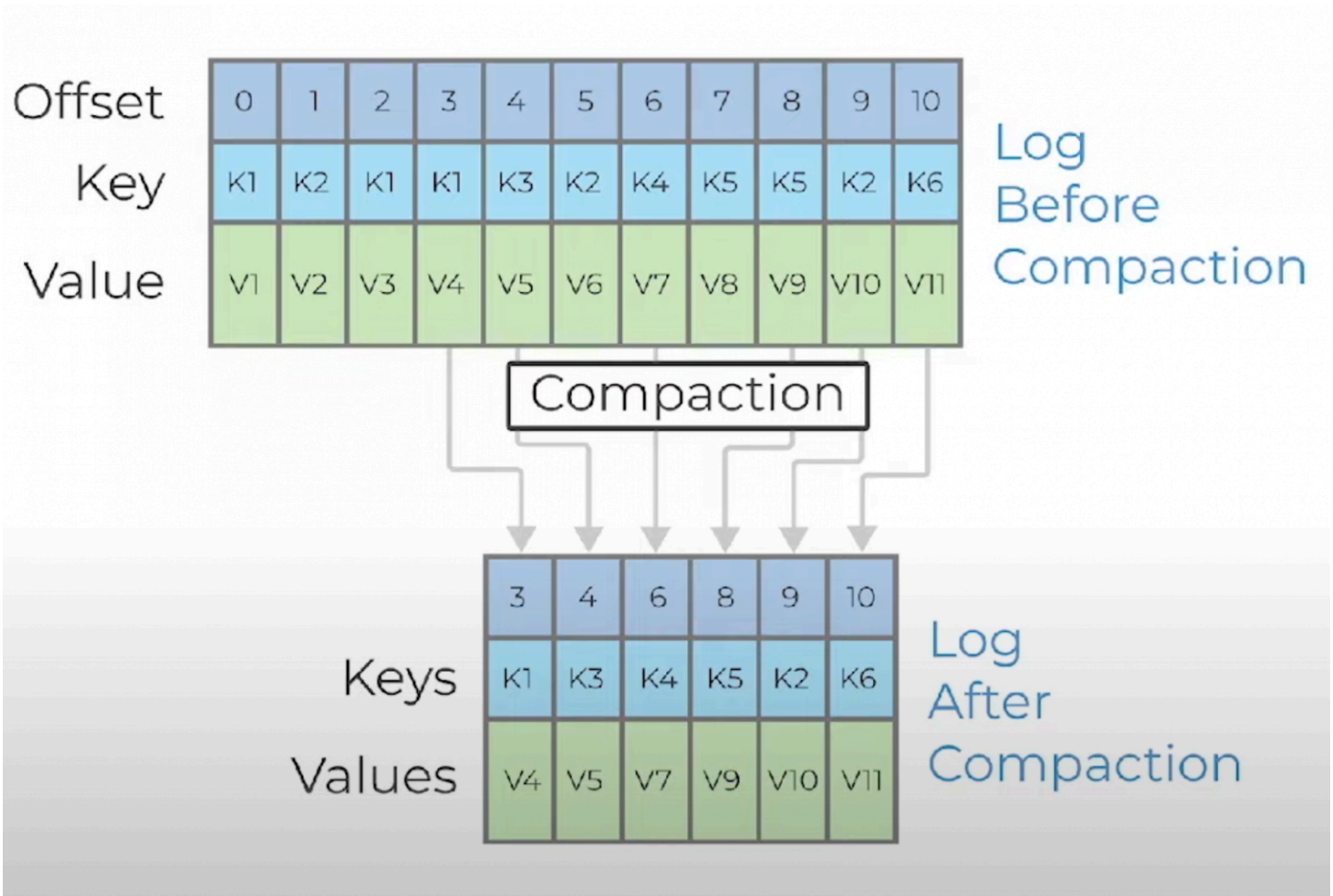
Из документации не всегда бывает очевидно, что и ретеншен по времени/размеру, и compaction могут быть включены для топика одновременно. Для этого их нужно указать через запятую в настройке cleanup.policy: delete для ретеншена (включен по умолчанию) и compact для compaction.



Голова (log head) в compacted топике абсолютно идентична обычной партиции. В ней хранятся все сообщения, даже с одинаковым ключом. Log compaction вносит изменения в то, как работает хвост лога партиции (log tail). Сообщения в хвосте не меняют свои оффсеты, вместо этого в хвосте появляются «дыры». Например, оффсеты 36, 37 и 38 будут идентичны. Соответственно, чтение с 36 и 37 будет идентично чтению с 38, поскольку он единственный, который остался.

Delete markers, сообщения с нулевым пэйлоадом, будут удалены Кафкой спустя некоторое время, чтобы освободить место на диске. На картинке это отмечено записью Delete Retention Point: после этого времени все delete markers будут удалены.

Сам compaction выполняется Кафкой в бэкграунд треде, который сжимает и перезаписывает закрытые сегменты. Активный сегмент никогда не подвергается сжатию, пока не станет закрытым. При этом log compaction как процесс не блокирует чтение данных.



Это еще один пример compaction. В первоначальной версии лога есть несколько версий сообщений с ключом K1, а также несколько версий с ключом K2. После compaction останется сжатый вариант этой партиции с последними записанными значениями по соответствующим ключам.

Характеристики Log Compaction

1. Это очень трудоемкий процесс для брокера, при котором возрастает нагрузка на диск (перезапись сегментов), память (загрузка данных из сегмента в java-процесс), процессор (проведение обработки).
2. Он не атомарен. В определенные моменты времени могут одновременно присутствовать несколько сообщений, записанных с одним и тем же ключом. Вам придется делать обработку такой ситуации в консьюмере.
3. Оффсеты не меняются, порядок записей остается прежним.
4. Позволяет «удалять» записи по ключу, что хорошо подходит для снэпшоттинга и восстановления последнего состояния системы после падения/перезагрузки.
5. Механизм крайне мощный и полезный, но его понимание и работа с ним в продакшене не самые простые.

Приведем пример compaction из рабочей практики. Механизм применяется для соблюдения закона GDPR в Европе для того, чтобы удалять данные о пользователях из Кафки. Кафка не является БД, нельзя просто так взять и удалить оффсет. Можно включить retention, но при этом будут удаляться целые куски данных. Log compaction же позволяет выборочно удалять сообщения.

Что еще? Советуем посмотреть живой пример Confluent Schema Registry по ссылке https://github.com/confluentinc/schema-registry. Это приложение является консьюмером из топика, в котором хранятся все схемы, и который подвергается compaction-у.

Коротко о ZooKepeer

Кафка использует эту систему как хранилище метаданных (например, конфигурации топиков), механизм leader election и для других операций, где требуется высокая консистентность данных.

Чтобы открыть ZooKeeper, воспользуемся скриптом ./bin/zookeeper-shell.sh и передадим ему адрес ZooKeeper-а, к которому хотим подключиться. В нашем случае это localhost:2181.

```
./bin/zookeeper-shell.sh localhost:2181
```

Данные хранятся как ключи значения, организованные в структуру папок и файлов. Здесь есть путь до ключа, допустим /a, /b, /c. Эта нода и является вашим ключом. Чтобы получить value, нужно сделать операцию get.

Разумеется, ничто не мешает вам хранить все ключи на самом высоком уровне под рутом; но описанным выше образом легче организовывать данные в иерархическую структуру. Это то, что

делает Кафка.

Итак, смотрим. В рутовом ключе есть целый набор подключей. Мы можем заглянуть чуть глубже: сделаем ls /brokers, увидим там еще подключи. Чтобы получить значение, которое хранится в ZooKeeper-е по ноде для контроллера, можно воспользоваться командой get /controller. В нашем случае контроллером выступает “brokerid”:0 — тот единственный брокер, который сейчас запущен.

Мы можем посмотреть состояние нашей партиции: `get /brokers/topics/registrations/partitions/0/state`

Получаем еще один json, в котором хранится текущее состояние партиции. Лидером у нашей партиции, например, является брокер 0, потому что он у нас один.

Посмотреть метаданные о ноде можно через такую команду: `stat /brokers/ids/0`

Целиком выглядит так:

```
ls /

get /controller

get /brokers/topics/registrations/partitions/0/state

stat /brokers/ids/0
```

Мы смотрим именно этот ключ не случайно: нода эфемерна. Так называются ноды ZooKeeper-а, которые хранятся в нем до тех пор, пока между клиентом и сервером есть устойчивое соединение и обмен heartbeat-ами. Что и делает Кафка: она подключается к ZooKeeper и начинает посылать heartbeat-ы, чтобы убедиться в устойчивости соединения. До тех пор, пока такое подключение будет работать, эфемерная нода будет доступна для чтения другими приложениями.

По большому счету, именно таким образом контроллер Кафки узнает, какие брокеры в данный момент запущены в кластере. Если мы остановим брокер, то эфемерная нода исчезнет.

Посмотреть, какие брокеры сейчас подключены, можно через команду /brokers/ids.

В основном, данными из ZooKeeper пользуется контроллер ноды в кластере Кафки. Именно она манипулирует здесь данными, смотрит на список активных брокеров, выбирает новых лидеров партиции. Затем через API самой Кафки и request-response между брокерами она распределяет полученную информацию и отсылает ее своим «подчиненным» в кластере.

Теперь вы умеете работать с Кафкой. А если хотите сдвинуться с базовой точки и полностью освоить необходимый стек, присоединяйтесь к базовому потоку по Apache Kafka с 6 марта. Смотреть программу: https://slurm.club/3Y65t9l

Теги: kafka, apache kafka, брокер, zookeeper, хранения данных, партиция

Хабы: Блог компании Слёрм, IT-инфраструктура, DevOps

Слёрм

Учебный центр для тех, кто работает в IT

Сайт

14 Карма 0 Рейтинг

Лиза Зубарькова @zubarek

Пользователь

Подписаться

Комментарии 3

mrgreyves 2 мар 2023 в 16:08

Здравствуйте!

Чем обусловлен выбор 2.7 версии?

Сейчас же актуальная версия 3.2

0 Ответить

admikar 3 мар 2023 в 12:12

Добрый,
В статье автор указал, что это сделано по видосу (*Материал записан на основе уроков курса **Apache Kafka База**. Версия для Кафки с ZooKeeper. Видеозапись смотрите здесь.*) который от Mar 25, 2021, а это как раз где-то 2.7.
Сейчас уже есть и 3.4, но я лично пока только на 3.2

0 Ответить

Sublimee 3 мар 2023 в 12:13

> Итак, мы создали топик registrations-0, где 0 — это идентификатор партиции.

Был создан топик **registrations**, никак не registrations-0. registrations-0 это партиция с идентификатором 0 в топике registrations.

0 Ответить

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

123skipper123 23 часа назад

3D рендер с редактором карт в Консоли

Средний 12 мин 3.1K

Из песочницы

+40 35 6 +6

RationalAnswer 13 часов назад

Доверие и честность в инвестициях, или два открытых вопроса Андрею Мовчану и Елене Чирковой по фонду GEIST

7 мин 5.5K

+30 5 14 +14

El_Gato_Grande 10 часов назад

Не «Ctrl+C»/«Ctrl+V» едиными. История клавиш-модификаторов

10 мин 5.9K

+26 19 5 +5

ru_vds 10 часов назад

GPU для дата-центров

Простой 6 мин 1.2K

Обзор

+24 9 0

habr_career 7 часов назад

Зарплаты разработчиков в первом полугодии 2024: языки и квалификации

5 мин 9.8K

+22 11 17 +17

ru_vds 6 часов назад

Советы по программированию, которые бы я дал себе 15 лет назад

Средний 8 мин 2.8K

Обзор Перевод

+21 31 1 +1

CyberPaul 5 часов назад

Джойбаблз. Удивительная история самого известного фрикера планеты

Простой 7 мин 937

Ретроспектива

+20 6 0

Cad1L 8 часов назад

От десятков до сотен тысяч RPS: как мы создали API, который развивается 10 лет без дропа обратной совместимости

Средний 10 мин 1.6K

Ретроспектива

+20 19 6 +6

meloman47 12 часов назад

Как я собрал настоящую Hi-Fi аудиосистему за 125 тыс. рублей

Простой 10 мин 10K

Туториал

+19 33 24 +24

slavanikolsky 20 часов назад

Конец августа 2024. YouTube после замедления, про Rutube, Дзен и VK видео

5 мин 11K

+17 11 6 +6

Показать еще

ИНФОРМАЦИЯ

Сайт	slurm.io
Дата регистрации	15 ноября 2012
Дата основания	4 июня 2018
Численность	51–100 человек
Местоположение	Россия
Представитель	Антон Скобин

ССЫЛКИ

Подкаст «В поисках бирюзовых компаний»
www.youtube.com
IT-курсы Слёрма
to.slurm.io

ВИДЖЕТ

БЛОГ НА ХАБРЕ

6 часов назад

Системные сервисы Linux: плюсы, минусы и особенности

871 1 +1

13 авг в 18:59

Решаем задачи на Go без внешних зависимостей

2.4K 0

2 авг в 15:11

Высокая доступность в Kubernetes