Архитектура программных систем. Лекция 3
Брокер сообщений

Михаил Пожидаев

6 марта 2025 г.

Брокер

Функции и назначение

В микросервисной архитектуре брокер используется для:

1. Гарантированной доставки сообщений между микросервисами.
2. Балансирования нагрузки с толерантностью отказов собственных реплик.
3. Унификации представления структур для работы на разных языках и платформах.

Примеры

Наиболее известные реализации брокеров

- *Apache Kafka:* написанный на Java и Scala наиболее распространённый брокер сообщений проекта Apache;
- *Rabbit MQ:* написанный на Erlang брокер с расширенными функциями маршрутизации сообщений;
- *centrifugo:* написанный на Go брокер сообщений с российскими корнями.

Apache Kafka

История развития проекта

Apache Kafka — распределённая платформа для обработки и передачи сообщений в реальном времени.

- **Разработка в LinkedIn.** В 2011 году компания разработала Kafka для решения проблем с обработкой потоков данных. Авторами были Джей Крепс, Ние Нархид и Цзюн Жао.
- **Принятие в фонд Apache.** В 2012 г. Исходный код проекта принят в Apache Software Foundation.
- **Создание Confluent.** В 2014 г. авторы создали компанию Confluent для продвижения Apache Kafka.

Топики

Особенности организации топиков в Apache Kafka

Топик (topic) — это логическая категория, в которой хранятся сообщения. Топики используются для организации и управления данными в системе.

1. **Производители (Producers)**: приложения или сервисы, которые создают и отправляют данные в топики Kafka.
2. **Потребители (Consumers)**: приложения или сервисы, которые читают и обрабатывают данные из топиков Kafka.
3. **Разделы (Partitions)**: способ разделения данных внутри топика на более мелкие части. Разделы позволяют масштабировать обработку данных. Каждый топик может быть разделён на один или несколько разделов.
4. **Реплики (Replicas)**: копии данных, хранящиеся на разных брокерах.

Сообщения в топике

Структура сообщений Apache Kafka

1. **Ключ (key):** уникальный идентификатор «назначения» сообщения. Если ключ отсутствует, сообщение считается анонимным.
2. **Значение (value):** данные, которые передаются вместе с сообщением.
3. **«Надгробный камень» (tombstone):** специальный маркер, который используется для удаления данных из темы. Tombstones не содержат значений и применяются для очистки тем от устаревших данных.

Компактинг

И функции базы данных

Компактинг (compacting) — это возможность после истечения указанного промежутка времени для каждого значения ключа оставлять только самую свежую запись.

Tombstones

Добавление в топик «надгробного камня» приводит к полному удалению записей с указанным ключом.

База данных

Apache Kafka может рассматриваться как база данных в режиме «только добавление». Топики могут очищаться через компактинг или обрезаться по истечению срока жизни записей.

Ключевые параметры

Управление поведением Apache Kafka

- `KAFKA_LOG_CLEANER_ENABLE` — режим очищения топиков
- `KAFKA_LOG_CLEANUP_POLICY` — режим очищения
- `KAFKA_LOG_RETENTION_CHECK_INTERVAL_MS` — как часто очищать топик
- `KAFKA_LOG_RETENTION_HOURS` — как долго хранить записи
- `KAFKA_LOG_RETENTION_BYTES` — лимит размера топика
- `KAFKA_LOG_CLEANER_MIN_COMPACTION_LAG_MS` — срок жизни записи до компактинга
- `KAFKA_LOG_CLEANER_THREADS` — сколько потоков очистки можно запускать

Доставка сообщений

Как сообщения из топика попадают подписчикам?

1. Издатель помещает сообщение в топик.
2. Каждый подписчик заявляет своё желание получать сообщения, которые будут передаваться ему до тех пор, пока он не отзовёт своё намерение. Таким образом команды «вытягиваются» из топика по запросу.
3. Если несколько подписчиков формируют группу, то сообщение из топика получит только один подписчик из группы. Этот механизм связан с механизмом разделения топика на разделы (partitions).
4. При подключении подписчик имеет возможность просмотреть все сообщения топика, которые не были удалены по установленным правилам. Это позволяет полностью восстанавливать состояние подписчика, хотя и может быть вычислительно трудоёмко.

Apache Zookeeper

Пример для Docker Compose

```
zookeeper:
  image: confluentinc/cp-zookeeper:latest
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
  ports:
    - 22181:2181
```

Apache Kafka

Пример для Docker Compose

```
kafka:
  image: confluentinc/cp-kafka:latest
  depends_on:
    - zookeeper
  ports:
    - 29092:29092
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:29092
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    KAFKA_AUTO_CREATE_TOPICS_ENABLE: 'true'
```

Node.js

Пример подключения к Apache Kafka

```
const delay = ms => new Promise(resolve => setTimeout(resolve, ms));
await delay(5000);
this.kafka = new Kafka({
  logLevel: logLevel.ERROR,
  brokers: ["kafka:9092"],
  clientId: "disp",
  })
this.producer = this.kafka.producer();
await this.producer.connect();
```

Спасибо за внимание!

Всё о курсе: https://marigostra.ru/materials/arch.html
E-mail: msp@luwrain.org
Канал в Телеграм: @MarigostraRu