

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа № 6

Выполнила:

Кадникова Екатерина

Группа К3441

**Проверил:
Добряков Д. И.**

Санкт-Петербург

2026 г.

Задача

- подключить и настроить rabbitMQ/kafka;
- реализовать межсервисное взаимодействие посредством rabbitMQ/kafka.

Ход работы

Было реализовано межсервисное взаимодействие с использованием Kafka для асинхронного обмена данными между микросервисами без прямых вызовов API.

Для работы с Kafka использовался Docker Compose, который поднимает zookeeper, kafka и интерфейс для мониторинга Kafdrop. Через переменные окружения задаются параметры подключения, создаются необходимые топики (пример на листинге 1)

Листинг 1 - Настройка Kafka

```
kafka:  
  image: confluentinc/cp-kafka:7.5.3  
  depends_on:  
    - zookeeper  
  ports:  
    - "9092:9092"  
    - "29092:29092"  
  environment:  
    KAFKA_BROKER_ID: 1  
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181  
  
    KAFKA_LISTENERS:  
INTERNAL://0.0.0.0:29092,EXTERNAL://0.0.0.0:9092  
      KAFKA_ADVERTISED_LISTENERS:  
INTERNAL://kafka:29092,EXTERNAL://localhost:9092  
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:  
INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT  
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL  
  
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1  
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1  
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1  
  
      KAFKA_CREATE_TOPICS: |  
        property.created:1:1,  
        property.updated:1:1,  
        property.deleted:1:1,  
        favorite.added:1:1,
```

```
favorite.removed:1:1,  
user.created:1:1,  
user.updated:1:1,  
user.deleted:1:1,  
booking.created:1:1,  
booking.statusChanged:1:1
```

Топики служат каналами для передачи событий между сервисами. Каждый тип события имеет свой топик. Доступен просмотр и управление топиками из Kafdrop (см. рисунок 1).

Kafka Cluster Overview

Bootstrap servers	kafka:29092
Total topics	9
Total partitions	58
Total preferred partition leader	100%
Total under-replicated partitions	0

Brokers

ID	Host	Port	Rack	Controller	Number of partitions (% of total)
1	kafka	29092	-	Yes	58 (100%)

Topics ACLs

Name	(9)	Partitions	% Preferred	# Under-replicated	Custom Config
_consumer_offsets		50	100%	0	Yes
booking.created		1	100%	0	No
booking.statusChanged		1	100%	0	No
property.created		1	100%	0	No
property.deleted		1	100%	0	No
property.updated		1	100%	0	No
user.created		1	100%	0	No
user.deleted		1	100%	0	No
user.updated		1	100%	0	No

Рисунок 1 - Список топиков

Каждый сервис имеет свой продюсер (пример представлен на листинге 2), который инициирует события и отправляет их в соответствующий топик.

Листинг 2 - Продюсер в сервисе управления арендой

```
import { Kafka, Producer } from "kafkajs";  
  
let kafkaProducer: Producer;
```

```

export async function initKafka() {
  const kafka = new Kafka({
    clientId: "rental-service",
    brokers: [process.env.KAFKA_BROKER || "localhost:9092"],
  });

  kafkaProducer = kafka.producer();
  await kafkaProducer.connect();
  console.log("Kafka producer connected");
}

export async function emitEvent(topic: string, payload: any) {
  if (!kafkaProducer) throw new Error("Kafka producer not initialized");
  await kafkaProducer.send({
    topic,
    messages: [{ value: JSON.stringify(payload) }],
  });
}

```

Например, при создании бронирования (см. листинг 3) продюсер сериализует данные в JSON и отправляет их в Kafka, после чего событие становится доступным для всех подписанных потребителей.

Листинг 3 - Публикация в Kafka при создании бронирования

```

async create(tenantId: string, propertyId: string, startDate: string, endDate: string) {
  const activeStatuses = await this.statusRepo.find({
    where: { code: In(["active", "approved"]) },
  });

  if (activeStatuses.length > 0) {
    const statusIds = activeStatuses.map(s => s.id);

    const overlapping = await this.repo
      .createQueryBuilder("b")
      .where("b.propertyId = :propertyId", { propertyId })
      .andWhere("b.statusId IN (:...statusIds)", { statusIds })
      .andWhere("b.startDate <= :endDate AND b.endDate >= :startDate", { startDate, endDate })
      .getCount();

    if (overlapping > 0) {
      throw new HttpError(
        400,
      );
    }
  }
}

```

```

        "Property is already booked in this period"
    );
}

const pendingStatus = await this.statusRepo.findOneBy({
code: "pending" });
if (!pendingStatus) throw new HttpError(404, "Pending
status not found");

const booking = this.repo.create({
tenantId,
propertyId,
startDate,
endDate,
status: pendingStatus,
});
const saved = await this.repo.save(booking);

await emitEvent("booking.created", { bookingId: saved.id,
tenantId, propertyId });

return saved;
}

```

Также в каждом сервисе есть consumer, который подписан на интересующие его топики и реагирует на события. Например, rental-service подписан на топики user.deleted и property.deleted (настройка представлена на листинге 4, дополнительно - на рисунке 2). Сообщения обрабатываются асинхронно через eachMessage.

Листинг 4 - Consumer в rental-auth

```

import { Kafka } from "kafkajs";
import { BookingService } from "../services/booking.service";

export async function initBookingConsumer() {
    const kafka = new Kafka({ clientId: "rental-service",
brokers: [process.env.KAFKA_BROKER || "localhost:9092"] });
    const consumer = kafka.consumer({ groupId:
"rental-service-consumer" });
    const service = new BookingService();

    await consumer.connect();
    await consumer.subscribe({ topic: "user.deleted" });
    await consumer.subscribe({ topic: "property.deleted" });

    await consumer.run({

```

```

        eachMessage: async ({ topic, message }) => {
            const payload =
JSON.parse(message.value!.toString());
            switch (topic) {
                case "user.deleted":
                    await
service.removeByTenant(payload.userId);
                    break;
                case "property.deleted":
                    await
service.removeByProperty(payload.propertyId);
                    break;
            }
        },
    );
}

```

Когда приходит событие удаления пользователя или объекта недвижимости, происходит удаление всех связанных с этими сущностями бронирований.

The screenshot shows the Kafdrop interface for the 'user.deleted' topic. The top navigation bar has a 'Star' icon. Below it, there are three buttons: 'View Messages', 'Search Messages', and a red 'Delete topic' button. The main area is divided into several sections:

- Topic: user.deleted**: The title of the topic.
- Overview**: A table with the following data:

# of partitions	1
Preferred replicas	100%
Under-replicated partitions	0
Total size	2
Total available messages	2
- Configuration**: A section stating "No topic-specific configuration".
- Partition Detail**: A table with the following data:

Partition	First Offset	Last Offset	Size	Leader Node	Replica Nodes	In-sync Replica Nodes	Offline Replica Nodes	Preferred Leader	Under-replicated
0	0	2	2	1	1	1		Yes	No
- Consumers**: A table with the following data:

Group ID	Combined Lag
property-service-group	0
rental-service-consumer	0

Рисунок 2 - Топик user.deleted

Вывод

В ходе работы была реализована система межсервисного взаимодействия с использованием очередей Kafka, которая позволяет сервисам обмениваться событиями асинхронно и без прямых вызовов друг к другу. Каждый сервис генерирует события через продюсер и подписывается на нужные топики через консьюмер, что обеспечивает обновление данных и синхронизацию действий между сервисами.