МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и технологий

Направление **02.03.01** : Математика и компьютерные науки

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Исполнитель: _____ Яшнова Дарья Михайловна
группа 5130201/20002

Руководитель: _____ Мулюха Владимир Александрович

«___»_____ 2025г

Санкт-Петербург, 2025

1 Вариант 13

Используя docker создать контейнеры, необходимые для реализации следующего функционала с использованием RabbitMQ/Kafka, а также показать, как именно осуществляется передача в этих условиях:

13. Настроить взаимодействие отправителя и получателя таким образом, чтобы обмен информацией шел при помощи двух типов сообщений – обычные и критические, при этом получатель должен получать и обрабатывать обычные сообщения только при отсутствии критических в очереди.

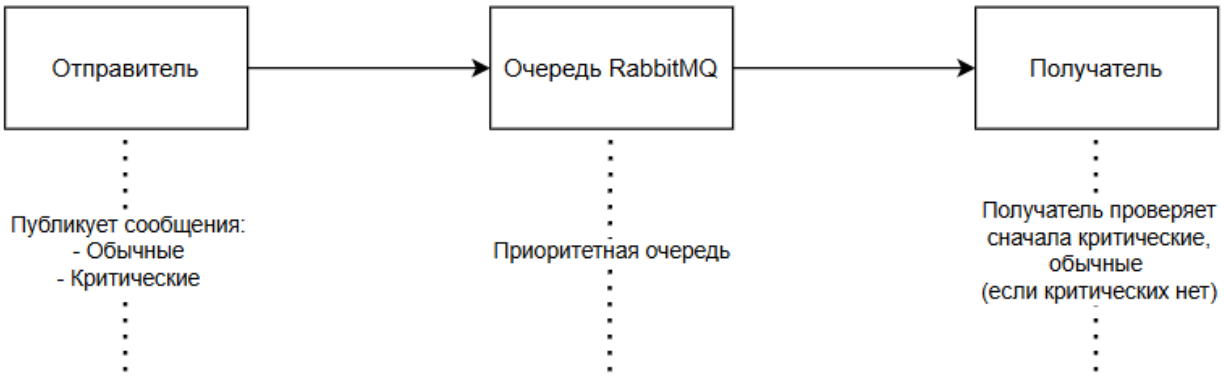

Рис. 1: Схема передачи сообщений

2 Docker

Docker — платформа для разработки, доставки и запуска приложений в контейнерах.

2.1 Ключевые концепции

- **Образ (Image)**: Шаблон с файловой системой и параметрами
- **Контейнер**: Запущенный экземпляр образа
- **Dockerfile**: Инструкции для сборки образа
- **Docker Compose**: Инструмент для управления мультиконтейнерными приложениями

2.2 Docker Compose

Файл `docker-compose.yml` определяет сервисы:

Листинг 1: docker-compose.yml

```yaml
version: '3.8'

services:
  rabbitmq:
    image: rabbitmq:3-management
    container_name: rabbitmq
    ports:
      - "5672:5672"
      - "15672:15672"
    environment:
      RABBITMQ_DEFAULT_USER: admin
      RABBITMQ_DEFAULT_PASS: admin
    healthcheck:
      test: ["CMD", "rabbitmqctl", "status"]
      interval: 30s
      timeout: 10s
      retries: 5

  producer:
    build:
      context: .
      dockerfile: Dockerfile.producer
    depends_on:
      rabbitmq:
        condition: service_healthy
    environment:
      - PYTHONUNBUFFERED=1
    restart: no

  consumer:
    build:
      context: .
      dockerfile: Dockerfile.consumer
    depends_on:
      rabbitmq:
        condition: service_healthy
    environment:
      - PYTHONUNBUFFERED=1
```

```
39      restart: unless-stopped
```

2.3 Параметры сервисов

Параметр	Описание
`image`	Используемый Docker-образ
`ports`	Проброс портов (хост:контейнер)
`build`	Сборка из Dockerfile
`depends_on`	Зависимости между сервисами
`restart`	Политика перезапуска

2.4 Dockerfile

Листинг 2: Dockerfile для producer и consumer

```
1
2  ####consumer
3  FROM python:3.9-slim
4
5  WORKDIR /app
6
7  COPY consumer.py .
8  COPY requirements.txt .
9
10 RUN pip install --no-cache-dir -r requirements.txt
11
12 CMD ["python", "consumer.py"]
13
14
15 ####producer
16
17 FROM python:3.9-slim
18
19 WORKDIR /app
20
21 COPY producer.py .
22 COPY requirements.txt .
23
24 RUN pip install --no-cache-dir -r requirements.txt
25
26 CMD ["python", "producer.py"]
```

2.5 Сборка и запуск

Сборка образов

```
1  docker-compose build
```

Запуск системы

```
1  docker-compose up
```

Просмотр логов

```
1  docker-compose logs -f consumer
```

3 RabbitMQ

RabbitMQ — это брокер сообщений с открытым исходным кодом, реализующий протокол AMQP (Advanced Message Queuing Protocol).

3.1 Ключевые особенности

- Асинхронная обработка сообщений
- Поддержка нескольких протоколов (AMQP, MQTT, STOMP)
- Кроссплатформенность
- Гибкая маршрутизация сообщений
- Надежность и отказоустойчивость

3.2 Основные компоненты

Компонент	**Описание**
Producer	Отправитель сообщений
Consumer	Получатель сообщений
Exchange	Принимает сообщения и маршрутизирует в очереди
Queue	Буфер для хранения сообщений
Binding	Правила маршрутизации между exchange и очередями

3.3 Приоритетные очереди в RabbitMQ

В вашей работе реализованы приоритетные очереди — механизм, позволяющий обрабатывать важные сообщения первыми.

3.3.1 Реализация приоритетов

Листинг 3: Объявление очереди с приоритетами

```
channel.queue_declare(
    queue='priority_queue',
    arguments={'x-max-priority': 10}
)
```

3.3.2 Типы сообщений

- **Критические** (приоритет 10) — обрабатываются первыми
- **Обычные** (приоритет 1) — обрабатываются при отсутствии критических

3.4 Применение в данной работе

3.4.1 Отправка сообщений (Producer)

```
# Подключение к RabbitMQ
connection = pika.BlockingConnection(
    pika.ConnectionParameters(
        host='rabbitmq',  # Имя сервиса в docker-compose
        credentials=pika.PlainCredentials('admin', 'admin')
    )
)
channel = connection.channel()

# Объявление очереди с приоритетами
channel.queue_declare(queue='priority_queue', arguments={'x-max-priority': 10})

# Генерация сообщений
for i in range(1, 11):
    # Случайно определяем, критическое ли сообщение
    is_critical = random.choice([True, False])
    priority = 10 if is_critical else 1
    message_type = "CRITICAL" if is_critical else "NORMAL"

    message = {
        'id': i,
        'type': message_type,
        'content': f"Message {i} - {message_type}"
    }

    channel.basic_publish(
        exchange='',
        routing_key='priority_queue',
        body=json.dumps(message),
        properties=pika.BasicProperties(
            priority=priority,
            delivery_mode=2  # persistent
        )
    )
    print(f" [x] Sent {message}")

connection.close()
```

3.4.2 Обработка сообщений (Consumer)

Листинг 4: consumer.py

```
# Подключение к RabbitMQ
connection = pika.BlockingConnection(
    pika.ConnectionParameters(
        host='rabbitmq',  # Имя сервиса в docker-compose
        credentials=pika.PlainCredentials('admin', 'admin')
    )
)
channel = connection.channel()

# Объявление очереди с приоритетами
channel.queue_declare(queue='priority_queue', arguments={'x-max-priority': 10})

print(' [*] Waiting for messages. To exit press CTRL+C')

def callback(ch, method, properties, body):
```

```
    message = json.loads(body)
    print(f" [x] Received {message}")

    # Имитация обработки
    processing_time = 2 if message['type'] == 'CRITICAL' else 1
    print(f" [x] Processing {message['type']} message for {processing_time} seconds")
    time.sleep(processing_time)

    print(f" [x] Done processing {message['id']}")
    ch.basic_ack(delivery_tag=method.delivery_tag)

# Настройка QoS для обработки одного сообщения за раз
channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue='priority_queue', on_message_callback=callback)

channel.start_consuming()
```

4 Результат работы

Логи запуска RabbitMQ

```
## ## RabbitMQ 3.13.7


## ##


########## Copyright (c) 2007-2024 Broadcom Inc and/or its subsidiaries


###### ##


########## Licensed under the MPL 2.0. Website: https://rabbitmq.com




Erlang: 26.2.5.11 [jit]


TLS Library: OpenSSL - OpenSSL 3.1.8 11 Feb 2025


Release series support status: see https://www.rabbitmq.com/release-information




Doc guides: https://www.rabbitmq.com/docs


Support: https://www.rabbitmq.com/docs/contact


Tutorials: https://www.rabbitmq.com/tutorials
```

```
36
37
38  Monitoring: https://www.rabbitmq.com/docs/monitoring
39
40
41  Upgrading: https://www.rabbitmq.com/docs/upgrade
42
43
44
45
46  Logs: <stdout>
47
48
49
50
51  Config file(s): /etc/rabbitmq/conf.d/10-defaults.conf
```

Работа producer

```
1  producer-1  |  [x] Sent {'id': 1, 'type': 'NORMAL', 'content': 'Message 1
       - NORMAL'}
2  producer-1  |  [x] Sent {'id': 2, 'type': 'CRITICAL', 'content': 'Message
       2 - CRITICAL'}
3  producer-1  |  [x] Sent {'id': 3, 'type': 'CRITICAL', 'content': 'Message
       3 - CRITICAL'}
4  producer-1  |  [x] Sent {'id': 4, 'type': 'NORMAL', 'content': 'Message 4
       - NORMAL'}
5  producer-1  |  [x] Sent {'id': 5, 'type': 'CRITICAL', 'content': 'Message
       5 - CRITICAL'}
6  producer-1  |  [x] Sent {'id': 6, 'type': 'CRITICAL', 'content': 'Message
       6 - CRITICAL'}
7  producer-1  |  [x] Sent {'id': 7, 'type': 'CRITICAL', 'content': 'Message
       7 - CRITICAL'}
8  producer-1  |  [x] Sent {'id': 8, 'type': 'NORMAL', 'content': 'Message 8
       - NORMAL'}
9  producer-1  |  [x] Sent {'id': 9, 'type': 'NORMAL', 'content': 'Message 9
       - NORMAL'}
10 producer-1  |  [x] Sent {'id': 10, 'type': 'CRITICAL', 'content': '
      Message 10 - CRITICAL'}
```

Работа consumer

```
1  consumer-1  |  [*] Waiting for messages. To exit press CTRL+C
2  consumer-1  |  [x] Received {'id': 1, 'type': 'NORMAL', 'content': '
      Message 1 - NORMAL'}
3  consumer-1  |  [x] Processing NORMAL message for 1 seconds
4  consumer-1  |  [x] Done processing 1
5  consumer-1  |  [x] Received {'id': 2, 'type': 'CRITICAL', 'content': '
      Message 2 - CRITICAL'}
6  consumer-1  |  [x] Processing CRITICAL message for 2 seconds
7  consumer-1  |  [x] Done processing 2
8  consumer-1  |  [x] Received {'id': 3, 'type': 'CRITICAL', 'content': '
      Message 3 - CRITICAL'}
9  consumer-1  |  [x] Processing CRITICAL message for 2 seconds
10 consumer-1  |  [x] Done processing 3
11 consumer-1  |  [x] Received {'id': 5, 'type': 'CRITICAL', 'content': '
      Message 5 - CRITICAL'}
12 consumer-1  |  [x] Processing CRITICAL message for 2 seconds
13 consumer-1  |  [x] Done processing 5
14 consumer-1  |  [x] Received {'id': 6, 'type': 'CRITICAL', 'content': '
      Message 6 - CRITICAL'}
15 consumer-1  |  [x] Processing CRITICAL message for 2 seconds
```

```
consumer-1  |  [x] Done processing 6
consumer-1  |  [x] Received {'id': 7, 'type': 'CRITICAL', 'content': '
    Message 7 - CRITICAL'}
consumer-1  |  [x] Processing CRITICAL message for 2 seconds
consumer-1  |  [x] Done processing 7
consumer-1  |  [x] Received {'id': 10, 'type': 'CRITICAL', 'content': '
    Message 10 - CRITICAL'}
consumer-1  |  [x] Processing CRITICAL message for 2 seconds
consumer-1  |  [x] Done processing 10
consumer-1  |  [x] Received {'id': 4, 'type': 'NORMAL', 'content': '
    Message 4 - NORMAL'}
consumer-1  |  [x] Processing NORMAL message for 1 seconds
consumer-1  |  [x] Done processing 4
consumer-1  |  [x] Received {'id': 8, 'type': 'NORMAL', 'content': '
    Message 8 - NORMAL'}
consumer-1  |  [x] Processing NORMAL message for 1 seconds
consumer-1  |  [x] Done processing 8
consumer-1  |  [x] Received {'id': 9, 'type': 'NORMAL', 'content': '
    Message 9 - NORMAL'}
consumer-1  |  [x] Processing NORMAL message for 1 seconds
consumer-1  |  [x] Done processing 9
```

5 Заключение

В результате этой работы были созданы Docker-контейнеры, реализующие схему передачи между ними сообщений с использованием очереди RabbitMQ, и было показано как именно осуществляется передача в этих условиях. Контейнеры были связаны между собой с помощью Docker Compose.