Министерство образования и науки
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
Направление 02.03.01 Математика и компьютерные науки

ЛАБОРАТОРНАЯ РАБОТА №3

Организация межпроцессорного взаимодействия с помощью очереди сообщений RabbitMQ

по дисциплине «Сети ЭВМ и телекоммуникации компьютерных сетей»

Выполнил студент гр. 5130201/10101	____________	Кондраев Дмитрий Евгеньевич
Проверил	____________	Мулюха Владимир Александрович

Санкт-Петербург
2024

1 Постановка задачи

Используя Docker, создать контейнеры, необходимые для реализации следующего функционала с использованием RabbitMQ, а также показать как именно осуществляется передача в этих условиях.

Вариант 8. Организовать общение трех отправителей и трех получателей, при этом каждому получателю соотвествует своя очередь. А отправители могут писать в каждую из очередей, используя механизм Topic Exchange с различными темами сообщений.

2 Теоретическая часть

Сообщения, отправленные на обмен `topic`, не могут иметь произвольный `routing_key` — это должен быть список слов, разделенных точками. Слова могут быть какими угодно, но обычно они определяют некоторые особенности, связанные с сообщением. Несколько правильных примеров ключей маршрутизации: «stock.usd.nyse», «nyse.vmw», «quick.orange.rabbit». В ключе маршрутизации может быть сколько угодно слов, вплоть до 255 байт.

`routing_key` также должен быть в той же форме. Логика обмена `topic`-ами аналогична `direct` — сообщение, отправленное с определенным ключом маршрутизации, будет доставлено во все очереди, которые связаны с соответствующим ключом привязки. Однако есть два особых случая привязки ключей[3]:

- `*` может заменить ровно одно слово.
- `#` может заменять ноль или более слов.

На рисунке 1 обозначена схема передачи сообщений, которая описана в варианте.

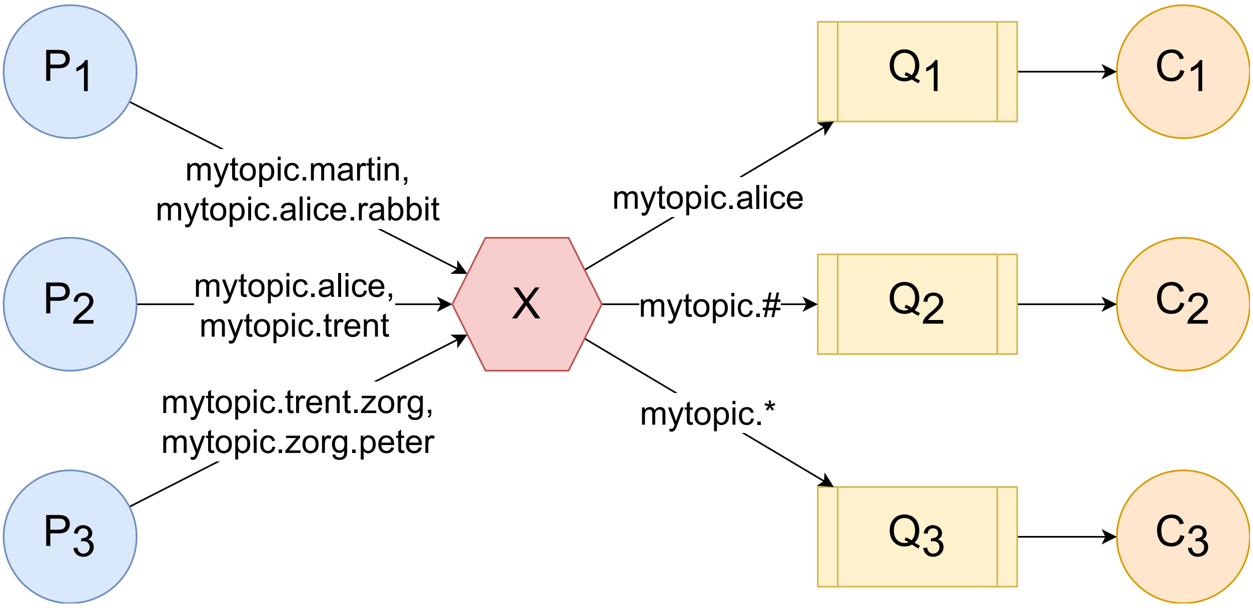

Рис. 1: Схема паредачи сообщений в RabbitMQ

3 Практическая часть

3.1 Установка требуемого ПО

Чтобы установить Docker, необходимо выполнить следующие команды[1] (для дистрибутива Fedora):

```
# Установка репозитория для пакетного менеджера
$ dnf -y install dnf-plugins-core
$ dnf config-manager --add-repo https://download.docker.com/linux/fedora/docker-ce.repo
# Установка Docker Engine, containerd, и Docker Compose
$ dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
 ↪  docker-compose-plugin
```

Для запуска демона Docker:

```
$ systemctl start docker
```

3.2 Настройка Docker Compose

В `docker-compose.yml` необходимо написать следующий код. Для того, чтобы можно было определить один Docker-образ для всех контейнеров-приемников и один — для всех контейнеров-источников, был использован механизм внедрения переменных окружения: у контейнеров-приемников задается переменная `TOPIC`, обозначающая ключ, который слушает данный приемник, а у контейнеров-источников — `TOPICS` — перечень ключей, по которым данный контейнер отправляет сообщения в Exchange.

```
# Docker Compose description of the combined application.
#
# 'docker compose up' will run this.

# This section describes the various containers (services).
services:

  rabbitmq:
    # There is a prebuilt RabbitMQ image; see
    # https://hub.docker.com/_/rabbitmq/ for details.
    # This variant is built on Alpine Linux (it's smaller) and includes
    # the management UI.
    image: 'rabbitmq:3.13-management-alpine'

    # These ports are exposed on the host; 'hostport:containerport'.
    # You could connect to this server from outside with the *host's*
    # DNS name or IP address and port 5672 (the left-hand side of the
    # colon).
    ports:
      # The standard AMQP protocol port
      - '5672:5672'
      # HTTP management UI
      - '15672:15672'

    # Run this container on a private network for this application.
    # This is necessary for magic Docker DNS to work: other containers
    # also running on this network will see a host name "rabbitmq"
    # (the name of this section) and the internal port 5672, even though
    # that's not explicitly published above.
    networks:
      - network

  consumer_a:
    # If needed, Docker Compose will automatically run consumer/Dockerfile.
    build: consumer

    # Environment variables:
    environment:
      # The location of the RabbitMQ server.  "amqp" is the protocol;
      # "rabbitmq" is the hostname.  Note that there is not a guarantee
      # that the server will start first!  Telling the pika client library
      # to try multiple times gets around this ordering issue.
      AMQP_URL: 'amqp://rabbitmq?connection_attempts=5&retry_delay=5'
      MY_NAME: 'Alice'
      TOPIC: 'mytopic.alice'

    # Again, run on the private network.  Needed to see the "rabbitmq"
    # magic Docker DNS name.
    networks:
      - network
    depends_on:
      - rabbitmq

  consumer_b:
    build: consumer
    environment:
      AMQP_URL: 'amqp://rabbitmq?connection_attempts=5&retry_delay=5'
      MY_NAME: 'Bob'
      TOPIC: 'mytopic.#'
    networks:
      - network
    depends_on:
      - rabbitmq
  consumer_c:
    build: consumer
    environment:
      AMQP_URL: 'amqp://rabbitmq?connection_attempts=5&retry_delay=5'
      MY_NAME: 'Carlo'
      TOPIC: 'mytopic.*'
    networks:
      - network
    depends_on:
      - rabbitmq

  publisher_x:
    # Identical to the consumer.
    build: publisher
    environment:
      AMQP_URL: 'amqp://rabbitmq?connection_attempts=5&retry_delay=5'
      MY_NAME: 'Xavier'
      TOPICS: 'mytopic.martin,mytopic.alice.rabbit'
    networks:
      - network
    depends_on:
      - rabbitmq

  publisher_y:
    build: publisher
    environment:
      AMQP_URL: 'amqp://rabbitmq?connection_attempts=5&retry_delay=5'
      MY_NAME: 'Yagami'
      TOPICS: 'mytopic.alice,mytopic.trent'
    networks:
      - network
  publisher_z:
    build: publisher
    environment:
      AMQP_URL: 'amqp://rabbitmq?connection_attempts=5&retry_delay=5'
      MY_NAME: 'Zorg'
      TOPICS: 'mytopic.trent.zorg,mytopic.zorg.peter'
    networks:
      - network

networks:
  # Declare our private network.  We must declare one for the magic
  # Docker DNS to work, but otherwise its default settings are fine.
  network: {}
```

Сборка и запуск контейнеров производится командами

```
$ docker compose build
$ docker compose up
```

3.3 Настройка образов Docker

Были определен следующий образ для контейнеров-приемников

```
# Dockerfile for the publisher application.
#
# 'docker compose build' from the parent directory will build this.
# 'docker compose up' will too, if it needs to be.
#
# To build this by hand, cd into the "consumer" directory and run
# 'docker build -t consumer .', and then you can manually run
# 'docker run --rm -e AMQP_URL=... consumer' to run it.

# This is based on the Python 3.12 Alpine Linux image.  See
# https://hub.docker.com/_/python/ for details on this image.
FROM python:3.12-alpine

# Our requirements are minimal, but it's good practice to install them
# first.  Put things that change less often towards the top of the
# Dockerfile.  Then if you need to rebuild the images, Docker will
# start running partway through the Dockerfile and skip over the steps
# where nothing's changed.
RUN pip install pika

# Without this setting, Python never prints anything out.
ENV PYTHONUNBUFFERED=1

# Actually install the application
WORKDIR /usr/src/app
# It's only a single file.  It has to be in the same directory as the
# Dockerfile, or a subdirectory, but not a parent or sibling.
COPY consumer.py .

# When you just 'docker run publisher' with no command afterwards,
# default to this:
CMD ["python", "/usr/src/app/consumer.py"]
```

И аналогичный образ для контейнеров-источников:

```
# Dockerfile for the publisher application.
#
```

```
# 'docker-compose build' from the parent directory will build this.
# 'docker-compose up' will too, if it needs to be.
#
# To build this by hand, cd into the "publisher" directory and run
# 'docker build -t publisher .', and then you can manually run
# 'docker run --rm -e AMQP_URL=... publisher' to run it.

# This is based on the Python 3.12 Alpine Linux image.  See
# https://hub.docker.com/_/python/ for details on this image.
FROM python:3.12-alpine

# Our requirements are minimal, but it's good practice to install them
# first.  Put things that change less often towards the top of the
# Dockerfile.  Then if you need to rebuild the images, Docker will
# start running partway through the Dockerfile and skip over the steps
# where nothing's changed.
RUN pip install pika

# Without this setting, Python never prints anything out.
ENV PYTHONUNBUFFERED=1

# Actually install the application
WORKDIR /usr/src/app
# It's only a single file.  It has to be in the same directory as the
# Dockerfile, or a subdirectory, but not a parent or sibling.
COPY publisher.py .

# When you just 'docker run publisher' with no command afterwards,
# default to this:
CMD ["python", "/usr/src/app/publisher.py"]
```

3.4 Программирование отправителей и получателей

Программы для отправителей и получателей были написаны на языке Python с использованием библиотеки pika в стиле «передачи продолжения»[2]. Этот прием программирования заключается в передаче управления через механизм продолжений, выраженных в виде замыканий.

Источники отправляют раз в 5 секунд сообщения в каждый из определенных для них `topic`.

Приемники асинхронно получают сообщения из своей очереди.

3.4.1 `consumer.py`

```
#!/usr/bin/env python

# RabbitMQ receiver.
#
# This creates a fanout exchange named "exchange", binds a queue named
# "exchange.receiver" to it, and prints any content it receives.
#
# This is largely boilerplate for the pika Python AMQP client library;
# see https://pika.readthedocs.org/ for details.  This uses a
# callback-oriented style where we perform an operation like "declare
# an AMQP exchange", and the pika library performs it and calls a
# callback when the server has sent back an acknowledgement.  As such,
# there are many short not obviously connected functions.

import os
```

```
from functools import partial

# AMQP client library.
import pika

#: Name of the RabbitMQ exchange.
EXCHANGE = 'topic_exchange'

MY_NAME = os.environ['MY_NAME']
THE_TOPIC = os.environ['TOPIC']


#: Name of the RabbitMQ queue.
QUEUE = f'exchange.receiver.{MY_NAME.lower()}'


def main():
    # Get the location of the AMQP broker (RabbitMQ server) from
    # an environment variable
    amqp_url = os.environ['AMQP_URL']
    log(f'URL: {amqp_url}')

    # Actually connect
    parameters = pika.URLParameters(amqp_url)
    connection = pika.SelectConnection(parameters, on_open_callback=on_open,
 ↪  on_close_callback=on_close)

    # Main loop.  This will run forever, or until we get killed.
    try:
        connection.ioloop.start()
    except KeyboardInterrupt:
        connection.close()
        connection.ioloop.start()


def log(*args):
    print(MY_NAME + ":", *args)


def on_open(connection):
    """Callback when we have connected to the AMQP broker."""
    log('Connected')
    connection.channel(on_open_callback=on_channel_open)


def on_close(connection, exception):
    # Invoked when the connection is closed
    log('Connection closed:', exception)
    connection.ioloop.stop()


def on_channel_open(channel):
    """Callback when we have opened a channel on the connection."""
    log('Have channel')

    # We must declare the exchange before we can bind to it.  It
    # doesn't matter that both the publisher and consumer are
    # declaring the same exchange, except that they must both declare
    # it with the same parameters.
    channel.exchange_declare(exchange=EXCHANGE, exchange_type='topic',
                             durable=True,
                             callback=partial(on_exchange, channel))

    # If we were brave we could also call queue_declare here, but
    # in the callback chain we'd have to wait to bind the queue to
    # the exchange until both had been declared.


def on_exchange(channel, frame):
    """Callback when we have successfully declared the exchange."""
    log('Have exchange')
    channel.queue_declare(queue=QUEUE, durable=True,
                          callback=partial(on_queue, channel))


def on_queue(channel, frame):
    """Callback when we have successfully declared the queue."""
    log('Have queue')

    # This call tells the server to send us 1 message in advance.
    # This helps overall throughput, but it does require us to deal
    # with the messages we have promptly.
    channel.basic_qos(prefetch_count=1, callback=partial(on_qos, channel))


def on_qos(channel, frame):
    """Callback when we have set the channel prefetch limit."""
    log('Set QoS')
    channel.queue_bind(queue=QUEUE, exchange=EXCHANGE, routing_key=THE_TOPIC,
                       callback=partial(on_bind, channel))


def on_bind(channel, frame):
    """Callback when we have successfully bound the queue to the exchange."""
    log('Bound')
    channel.basic_consume(queue=QUEUE, on_message_callback=on_message)


def on_message(channel, method, properties, body):
    """Callback when a message arrives.

    :param channel: the AMQP channel object.
    :type channel: :class:`pika.channel.Channel`

    :param method: the AMQP protocol-level delivery object,
      which includes a tag, the exchange name, and the routing key.
      All of this should be information the sender has as well.
    :type method: :class:`pika.spec.Deliver`

    :param properties: AMQP per-message metadata.  This includes
      things like the body's content type, the correlation ID and
      reply-to queue for RPC-style messaging, a message ID, and so
      on.  It also includes an additional table of structured
      caller-provided headers.  Again, all of this is information
      the sender provided as part of the message.
    :type properties: :class:`pika.spec.BasicProperties`

    :param str body: Byte string of the message body.

    """
    # Just dump out the information we think is interesting.
    log(f'Exchange: {method.exchange}')
    log(f'Routing key: {method.routing_key}')
    log(f'Content type: {properties.content_type}')
    log()
    log(body)
    log()

    # Important!!! You MUST acknowledge the delivery.  If you don't,
    # then the broker will believe it is still outstanding, and
    # because we set the QoS limit above to 1 outstanding message,
    # we'll never get more.
    #
    # If something went wrong but retrying is a valid option, you
    # could also basic_reject() the message.
    channel.basic_ack(method.delivery_tag)


if __name__ == '__main__':
    main()
```

3.4.2 publisher.py

```
#!/usr/bin/env python

# Simple RabbitMQ publisher.
#
# This creates a fanout exchange named "exchange", then publishes a message
# there every 5 seconds.
#
# This is largely boilerplate for the pika Python AMQP client library;
# see https://pika.readthedocs.org/ for details.  This uses a
# callback-oriented style where we perform an operation like "declare
# an AMQP exchange", and the pika library performs it and calls a
# callback when the server has sent back an acknowledgement.  As such,
# there are many short not obviously connected functions.

import os
from functools import partial

# AMQP client library.
import pika

#: Name of the RabbitMQ exchange.
EXCHANGE = 'topic_exchange'


#: Delay between sending messages.
DELAY = 5

MY_NAME = os.environ['MY_NAME']
THE_TOPICS = os.environ['TOPICS'].split(',')


def main():
    # Get the location of the AMQP broker (RabbitMQ server) from
    # an environment variable
    amqp_url = os.environ['AMQP_URL']
    log(f'URL: {amqp_url}')
    # Actually connect
    parameters = pika.URLParameters(amqp_url)
    connection = pika.SelectConnection(parameters, on_open_callback=on_open,
↪   on_close_callback=on_close)

    # Main loop.  This will run forever, or until we get killed.
    try:
        connection.ioloop.start()
    except KeyboardInterrupt:
        connection.close()
        connection.ioloop.start()


def log(*args):
    print(MY_NAME + ":", *args)


def on_open(connection):
    """Callback when we have connected to the AMQP broker."""
    log('Connected')
    connection.channel(on_open_callback=on_channel_open)


def on_close(connection, exception):
    # Invoked when the connection is closed
    log('Connection closed:', exception)
    connection.ioloop.stop()


def on_channel_open(channel):
    """Callback when we have opened a channel on the connection."""
    log('Have channel')
    channel.exchange_declare(exchange=EXCHANGE, exchange_type='topic',
                             durable=True,
                             callback=partial(on_exchange, channel))


def on_exchange(channel, frame):
    """Callback when we have successfully declared the exchange."""
    log('Have exchange')
    send_message(channel, 0)


def send_message(channel, i):
    """Send a message to the queue.

    This function also registers itself as a timeout function, so the
    main :mod:`pika` loop will call this function again every 5 seconds.

    """
    routing_key = THE_TOPICS[i % len(THE_TOPICS)]
    msg = f'Message {i} from {MY_NAME} with routing key: {routing_key}'
    log(msg)
    channel.basic_publish(exchange=EXCHANGE,
                          routing_key=routing_key,
                          body=msg,
                          properties=pika.BasicProperties(content_type='text/plain',
                                                          delivery_mode=1))
    channel.connection.ioloop.call_later(DELAY,
                                         partial(send_message, channel, i + 1))
```

```
# Python boilerplate to run the main function if this is run as a
# program.  You can 'import publisher' from other Python scripts in
# the same directory to get access to the functions here, or run
# 'pydoc publisher.py' to see the doc strings, and so on, but these
# require this call.
if __name__ == '__main__':
    main()
```

3.5 Результат работы

Лог запуска RabbitMQ:

```
rabbitmq-1      |   ##  ##      RabbitMQ 3.13.2
rabbitmq-1      |   ##  ##
rabbitmq-1      |   ##########  Copyright (c) 2007-2024 Broadcom Inc and/or its subsidiaries
rabbitmq-1      |   ######  ##
rabbitmq-1      |   ##########  Licensed under the MPL 2.0. Website: https://rabbitmq.com
rabbitmq-1      |
rabbitmq-1      |   Erlang:      26.2.5 [jit]
rabbitmq-1      |   TLS Library: OpenSSL - OpenSSL 3.1.5 30 Jan 2024
rabbitmq-1      |   Release series support status: supported
rabbitmq-1      |
rabbitmq-1      |   Doc guides:  https://www.rabbitmq.com/docs
rabbitmq-1      |   Support:     https://www.rabbitmq.com/docs/contact
rabbitmq-1      |   Tutorials:   https://www.rabbitmq.com/tutorials
rabbitmq-1      |   Monitoring:  https://www.rabbitmq.com/docs/monitoring
rabbitmq-1      |   Upgrading:   https://www.rabbitmq.com/docs/upgrade
rabbitmq-1      |
rabbitmq-1      |   Logs: <stdout>
rabbitmq-1      |
rabbitmq-1      |   Config file(s): /etc/rabbitmq/conf.d/10-defaults.conf
rabbitmq-1      |
```

Из дальнейшего лога видно, что сообщения передаются в соответствии с указанными топиками.

```
publisher_y-1  | Yagami: URL: amqp://rabbitmq?connection_attempts=5&retry_delay=5
publisher_z-1  | Zorg: URL: amqp://rabbitmq?connection_attempts=5&retry_delay=5
consumer_a-1   | Alice: URL: amqp://rabbitmq?connection_attempts=5&retry_delay=5
rabbitmq-1     | =INFO REPORT==== 17-May-2024::09:41:03.914122 ===
rabbitmq-1     |     alarm_handler: {set,{system_memory_high_watermark,[]}}
consumer_b-1   | Bob: URL: amqp://rabbitmq?connection_attempts=5&retry_delay=5
consumer_c-1   | Carlo: URL: amqp://rabbitmq?connection_attempts=5&retry_delay=5
publisher_x-1  | Xavier: URL: amqp://rabbitmq?connection_attempts=5&retry_delay=5
 ...
rabbitmq-1     | 2024-05-17 09:41:16.718887+00:00 [info] <0.9.0> Time to start RabbitMQ:
 ↪  13182 ms
rabbitmq-1     | 2024-05-17 09:41:17.255839+00:00 [info] <0.710.0> accepting AMQP connection
 ↪  <0.710.0> (172.18.0.3:47644 -> 172.18.0.4:5672)
rabbitmq-1     | 2024-05-17 09:41:17.255883+00:00 [info] <0.707.0> accepting AMQP connection
 ↪  <0.707.0> (172.18.0.2:47788 -> 172.18.0.4:5672)
publisher_z-1  | Zorg: Connected
publisher_y-1  | Yagami: Connected
rabbitmq-1     | 2024-05-17 09:41:17.261399+00:00 [info] <0.707.0> connection <0.707.0>
 ↪  (172.18.0.2:47788 -> 172.18.0.4:5672): user 'guest' authenticated and granted access to
 ↪  vhost '/'
rabbitmq-1     | 2024-05-17 09:41:17.262191+00:00 [info] <0.710.0> connection <0.710.0>
 ↪  (172.18.0.3:47644 -> 172.18.0.4:5672): user 'guest' authenticated and granted access to
 ↪  vhost '/'
```

```
publisher_z-1  | Zorg: Have channel
publisher_z-1  | Zorg: Have exchange
publisher_y-1  | Yagami: Have channel
publisher_z-1  | Zorg: Message 0 from Zorg with routing key: mytopic.trent.zorg
publisher_y-1  | Yagami: Have exchange
publisher_y-1  | Yagami: Message 0 from Yagami with routing key: mytopic.alice
rabbitmq-1     | 2024-05-17 09:41:18.792426+00:00 [info] <0.735.0> accepting AMQP connection
 ↪  <0.735.0> (172.18.0.5:49862 -> 172.18.0.4:5672)
consumer_a-1   | Alice: Connected
rabbitmq-1     | 2024-05-17 09:41:18.797213+00:00 [info] <0.735.0> connection <0.735.0>
 ↪  (172.18.0.5:49862 -> 172.18.0.4:5672): user 'guest' authenticated and granted access to
 ↪  vhost '/'
consumer_a-1   | Alice: Have channel
consumer_a-1   | Alice: Have exchange
consumer_a-1   | Alice: Have queue
consumer_a-1   | Alice: Set QoS
consumer_a-1   | Alice: Bound
consumer_a-1   | Alice: Exchange: topic_exchange
consumer_a-1   | Alice: Routing key: mytopic.alice
consumer_a-1   | Alice: Content type: text/plain
consumer_a-1   | Alice:
consumer_a-1   | Alice: b'Message 0 from Yagami with routing key: mytopic.alice'
consumer_a-1   | Alice:
rabbitmq-1     | 2024-05-17 09:41:19.067324+00:00 [info] <0.749.0> accepting AMQP connection
 ↪  <0.749.0> (172.18.0.6:36840 -> 172.18.0.4:5672)
consumer_b-1   | Bob: Connected
rabbitmq-1     | 2024-05-17 09:41:19.071669+00:00 [info] <0.749.0> connection <0.749.0>
 ↪  (172.18.0.6:36840 -> 172.18.0.4:5672): user 'guest' authenticated and granted access to
 ↪  vhost '/'
consumer_b-1   | Bob: Have channel
consumer_b-1   | Bob: Have exchange
consumer_b-1   | Bob: Have queue
consumer_b-1   | Bob: Set QoS
consumer_b-1   | Bob: Bound
consumer_b-1   | Bob: Exchange: topic_exchange
consumer_b-1   | Bob: Routing key: mytopic.trent.zorg
consumer_b-1   | Bob: Content type: text/plain
consumer_b-1   | Bob:
consumer_b-1   | Bob: b'Message 0 from Zorg with routing key: mytopic.trent.zorg'
consumer_b-1   | Bob:
consumer_b-1   | Bob: Exchange: topic_exchange
consumer_b-1   | Bob: Routing key: mytopic.alice
consumer_b-1   | Bob: Content type: text/plain
consumer_b-1   | Bob:
consumer_b-1   | Bob: b'Message 0 from Yagami with routing key: mytopic.alice'
consumer_b-1   | Bob:
rabbitmq-1     | 2024-05-17 09:41:19.871693+00:00 [info] <0.763.0> accepting AMQP connection
 ↪  <0.763.0> (172.18.0.8:36500 -> 172.18.0.4:5672)
consumer_c-1   | Carlo: Connected
rabbitmq-1     | 2024-05-17 09:41:19.876602+00:00 [info] <0.763.0> connection <0.763.0>
 ↪  (172.18.0.8:36500 -> 172.18.0.4:5672): user 'guest' authenticated and granted access to
 ↪  vhost '/'
consumer_c-1   | Carlo: Have channel
consumer_c-1   | Carlo: Have exchange
consumer_c-1   | Carlo: Have queue
consumer_c-1   | Carlo: Set QoS
consumer_c-1   | Carlo: Bound
consumer_c-1   | Carlo: Exchange: topic_exchange
consumer_c-1   | Carlo: Routing key: mytopic.alice
consumer_c-1   | Carlo: Content type: text/plain
```

```
consumer_c-1   | Carlo:
consumer_c-1   | Carlo: b'Message 0 from Yagami with routing key: mytopic.alice'
consumer_c-1   | Carlo:
rabbitmq-1     | 2024-05-17 09:41:19.936003+00:00 [info] <0.777.0> accepting AMQP connection
 ↪  <0.777.0> (172.18.0.7:59356 -> 172.18.0.4:5672)
publisher_x-1  | Xavier: Connected
rabbitmq-1     | 2024-05-17 09:41:19.940287+00:00 [info] <0.777.0> connection <0.777.0>
 ↪  (172.18.0.7:59356 -> 172.18.0.4:5672): user 'guest' authenticated and granted access to
 ↪  vhost '/'
publisher_x-1  | Xavier: Have channel
publisher_x-1  | Xavier: Have exchange
publisher_x-1  | Xavier: Message 0 from Xavier with routing key: mytopic.martin
consumer_c-1   | Carlo: Exchange: topic_exchange
consumer_c-1   | Carlo: Routing key: mytopic.martin
consumer_c-1   | Carlo: Content type: text/plain
consumer_c-1   | Carlo:
consumer_c-1   | Carlo: b'Message 0 from Xavier with routing key: mytopic.martin'
consumer_c-1   | Carlo:
consumer_b-1   | Bob: Exchange: topic_exchange
consumer_b-1   | Bob: Routing key: mytopic.martin
consumer_b-1   | Bob: Content type: text/plain
consumer_b-1   | Bob:
consumer_b-1   | Bob: b'Message 0 from Xavier with routing key: mytopic.martin'
consumer_b-1   | Bob:
publisher_z-1  | Zorg: Message 1 from Zorg with routing key: mytopic.zorg.peter
publisher_y-1  | Yagami: Message 1 from Yagami with routing key: mytopic.trent
consumer_b-1   | Bob: Exchange: topic_exchange
consumer_b-1   | Bob: Routing key: mytopic.zorg.peter
consumer_b-1   | Bob: Content type: text/plain
consumer_b-1   | Bob:
consumer_b-1   | Bob: b'Message 1 from Zorg with routing key: mytopic.zorg.peter'
consumer_b-1   | Bob:
consumer_c-1   | Carlo: Exchange: topic_exchange
consumer_c-1   | Carlo: Routing key: mytopic.trent
consumer_c-1   | Carlo: Content type: text/plain
consumer_c-1   | Carlo:
consumer_c-1   | Carlo: b'Message 1 from Yagami with routing key: mytopic.trent'
consumer_c-1   | Carlo:
consumer_b-1   | Bob: Exchange: topic_exchange
consumer_b-1   | Bob: Routing key: mytopic.trent
consumer_b-1   | Bob: Content type: text/plain
consumer_b-1   | Bob:
consumer_b-1   | Bob: b'Message 1 from Yagami with routing key: mytopic.trent'
consumer_b-1   | Bob:
```

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.

Список использованных источников

1. [Электронный ресурс]. – URL: https://docs.docker.com/engine/install/fedora/#install-using-the-repository (дата обращения: 16.05.2024).

2. Introduction to Pika — Continuation-Passing Style [Электронный ресурс]. – URL: https://pika.readthedocs.io/en/stable/intro.html?highlight=continuation#continuation-passing-style (дата обращения: 16.05.2024).

3. RabbitMQ tutorial — Topics [Электронный ресурс]. – URL: https://www.rabbitmq.com/tutorials/tutorial-five-python (дата обращения: 16.05.2024).