

Лабораторная работа 6. Основы работы с брокером очередей RabbitMQ

Жунёв Андрей Александрович РИМ-150950

1: Настройка RabbitMQ в Docker Compose

The screenshot shows the VS Code interface with the 'docker-compose.yml' file open in the center. The file defines two services: 'rabbitmq' and 'app'. The 'rabbitmq' service uses the 'rabitmq:3-management' image, exposes ports 5672 and 15672, and connects to a local PostgreSQL database. The 'app' service runs a Dockerfile and depends on the 'rabbitmq' service, exposing port 8000 and linking to the PostgreSQL database.

```
version: '3'
services:
  rabbitmq:
    image: rabbitmq:3-management
    container_name: rabbitmq_lab2
    ports:
      - "5672:5672"
      - "15672:15672"
    environment:
      - RABBITMQ_DEFAULT_VHOST=local
    volumes:
      - rabbitmq_data:/data
    networks:
      - pg_network_lab2
    restart: unless-stopped
  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: app_lab3
    environment:
      - DATABASE_URL=postgresql+asyncpg://$(POSTGRES_USER):$(POSTGRES_PASSWORD):admin@db:5432/$(POSTGRES_DB):lab_db2
      - DB_HOST=$DB_HOST:db
      - DB_PORT=$DB_PORT:5432
      - HOST=t{HOST-0..0..0}
      - PORT=t{PORT-8000}
      - RABBITMQ_HOST=${RABBITMQ_HOST:rabbitmq}
      - RABBITMQ_PORT=${RABBITMQ_PORT:5672}
      - RABBITMQ_VHOST=${RABBITMQ_VHOST:local}
      - RABBITMQ_USER=${RABBITMQ_USER:guest}
      - RABBITMQ_PASSWORD=${RABBITMQ_PASSWORD:guest}
    ports:
      - "8000:8000"
    volumes:
      - ./app/app
      - ./main.py:app/main.py
    networks:
      - pg_network_lab2
    depends_on:
      - db:
          condition: service_healthy
        rabbitmq:
          condition: service_started
        command: ["uvicorn", "run", "python", "main.py"]
    restart: unless-stopped
volumes:
  rabbitmq_data:
networks:
  pg_network_lab2:
```

В рамках первого этапа были внесены необходимые изменения в файл `docker-compose.yml`. Был добавлен сервис `rabbitmq`, использующий образ `rabbitmq:3-management`,

с пробросом портов 5672 для AMQP и 15672 для веб-интерфейса, а

также настроен виртуальный хост `local` и volume для сохранения данных .

Параллельно был обновлен сервис `app`: в него добавлены переменные окружения для подключения к брокеру, установлена зависимость от `rabbitmq` в секции

`depends_on`, и подключен том `rabbitmq_data`. В результате настройки RabbitMQ

стал доступен по стандартным адресам: AMQP на порту 5672 и панель управления на порту 15672 с учетными данными по умолчанию

2: Установка зависимостей для работы с RabbitMQ

```

requires-python = ">=3.13"
dependencies = [
    "alembic>=1.17.0",
    "asyncpg>=0.30.0",
    "faststream[rabbit]>=0.5.0",
    "litestar>=2.18.0",
    "pika>=1.3.0",
    "psycopg2-binary>=2.9.11",
    "pydantic[email]>=2.12.4".

```

Для обеспечения работы с брокером сообщений в файл `pyproject.toml` были добавлены необходимые зависимости: `faststream[rabbit]>=0.5.0` для асинхронного взаимодействия через FastStream и `pika>=1.3.0` для написания скриптов продюсера. Последующая установка через `uv sync` успешно загрузила 8 пакетов, включая ключевые библиотеки `faststream`, `pika` и `aio-pika`, необходимые для корректного функционирования системы

3: Создание структуры для работы с RabbitMQ

The screenshot shows the VS Code interface with the following details:

- Project Structure:** The left sidebar shows a tree view of the project files. Key components include `app` (containing `rabbitmq_consumer.py`), `database.py`, `dependencies.py`, `models.py`, `push_data.py`, `push_products_orders.py`, and `queries.py`; and `tests` (containing `test_rabbitmq_consumer.py`).
- Code Editor:** The main editor area displays the content of `rabbitmq_consumer.py`. The code imports logging, os, FastStream, RabbitBroker, and various repository modules. It defines a function `get_rabbitmq_url()` to generate a connection URL from environment variables. It also initializes the broker and creates a FastStream application.
- Terminal:** A terminal window at the bottom shows the command `lab2 git:(main) ✘`.
- Status Bar:** The status bar at the bottom right shows the current file is `rabbitmq_consumer.py`, the line number is 1, column is 1, and the Python version is 3.13.9 ('venv': venv).

Базовая структура проекта была расширена созданием файла `app/rabbitmq_consumer.py`. В нем настроены импорты основных компонентов FastStream и RabbitBroker, схемы данных для продуктов и заказов, а также функция `get_rabbitmq_url()` для генерации строки подключения и инициализация брокера с логированием. Подключение к RabbitMQ осуществляется с использованием переменных окружения (`RABBITMQ_HOST`, `USER`, `PASSWORD` и др.), которые формируют корректный URL, подготовливая систему к добавлению обработчиков сообщений.

4: Реализация обработки сообщений о продукции

The screenshot shows the VS Code interface with the following details:

- File Structure:** The left sidebar shows the project structure for `APP_DEV_SEM_3`, including `rabbitmq_consumer.py` under `app/rabbitmq_consumer.py`.
- Code Editor:** The main editor window displays the `rabbitmq_consumer.py` file. The code implements a consumer for RabbitMQ, handling product creation and update messages.
- Terminal:** A terminal tab at the bottom shows the command `git:(main) ✘`.
- Bottom Status Bar:** Shows the current file is `rabbitmq_consumer.py - App_Dev_sem_1`, and the status bar includes information like `Cursor Tab`, `Ln 1, Col 1`, `Spaces: 4`, `UTF-8`, `LF`, `Python 3.13.9 ('venv': venv)`, and `Go Live`.

```
cursormq_consumer.py - App_Dev_sem_1
rabitmq_consumer.py

lab2 > app > rabbitmq_consumer.py > ...
58     # Провайдер для dependency injection
59     @async def get_db_session() -> AsyncSession:
60         """Провайдер сессии базы данных для RabbitMQ consumer."""
61         async with async_session_factory() as session:
62             try:
63                 yield session
64             finally:
65                 await session.close()
66
67
68     @async def get_product_repository(
69         session: Annotated[AsyncSession, Depends(get_db_session)],
70     ) -> ProductRepository:
71         """Провайдер репозитория продуктов."""
72         return ProductRepository()
73
74
75     @async def get_product_service(
76         product_repository: Annotated[
77             ProductRepository, Depends(get_product_repository)
78         ],
79     ) -> ProductService:
80         """Провайдер сервиса продуктов."""
81         return ProductService(product_repository)
82
83
84     # Обработчики сообщений о продукции
85     @broker.subscriber("product")
86     @async def subscribe_product_create(
87         product_data: ProductCreate,
88         session: Annotated[AsyncSession, Depends(get_db_session)],
89         product_service: Annotated[ProductService, Depends(get_product_service)],
90     ) -> None:
91         """
92             Обработчик создания продукта через RabbitMQ.
93
94         Args:
95             product_data: Данные для создания продукта
96             session: Сессия базы данных
97             product_service: Сервис для работы с продуктами
98
99         """
100        try:
101            logger.info(f"Received product create request: {product_data.name}")
102            product = await product_service.create(session, product_data)
103            logger.info(f"Product created successfully: ID={product.id}, Name={product.name}")
104        except ValueError as e:
105
106
107    @broker.subscriber("product_update")
108    @async def subscribe_product_update(
109        message: ProductUpdateMessage,
110        session: Annotated[AsyncSession, Depends(get_db_session)],
111        product_service: Annotated[ProductService, Depends(get_product_service)],
112    ) -> None:
113        """
114            Обработчик обновления продукта через RabbitMQ.
115
116        Args:
117            message: Сообщение с ID продукта и данными для обновления
118            session: Сессия базы данных
119            product_service: Сервис для работы с продуктами
120
121        """
122        try:
123            logger.info(f"Received product update request: ID={message.product_id}")
124
125            product = await product_service.update(
126                session, message.product_id, message.product_data
127            )
128            logger.info(
129                f"Product updated successfully: ID={product.id}, "
130                f"Stock={product.stock_quantity}"
131            )
132
133        # Проверка, не закончился ли товар на складе
134        if product.stock_quantity == 0:
135            logger.warning(
136                f"Product {product.name} (ID={product.id}) is out of stock!"
137            )
138
139        except ValueError as e:
140            logger.error(f"Error updating product: {e}")
141        except Exception as e:
142            logger.error(f"Unexpected error updating product: {e}", exc_info=True)
```

Реализована логика подписчиков (subscribers) для работы с продукцией. Подписчик на создание продукции (`subscribe_product_create`) слушает очередь `product`, принимает схему `ProductCreate` и использует `ProductService` для записи данных в БД, логируя результаты операций. Второй подписчик (`subscribe_product_update`) обрабатывает сообщения из очереди `product_update`, используя новую схему `ProductUpdateMessage` для валидации данных. Важной особенностью обновления является проверка наличия товара: если `stock_quantity` падает до нуля, система не только обновляет данные, но и логирует соответствующее предупреждение. Весь функционал поддерживается настроенным Dependency Injection для внедрения сессий и сервисов.

5. Реализация обработки сообщений о заказах

```
37 class OrderUpdate(BaseModel):
38     """Схема для обновления заказа. Все поля опциональные."""
39
40     status: str | None = Field(None, description="Статус заказа")
41
42
43 class OrderUpdateMessage(BaseModel):
44     """Схема для сообщения обновления заказа через RabbitMQ."""
45
46
47     order_id: int = Field(..., gt=0, description="ID заказа для обновления")
48     order_data: OrderUpdate = Field(..., description="Данные для обновления заказа")
49
50
51
52 Cursor File Edit Selection View Go Run Terminal Window Help
53
54 Agents Editor
55
56 rabbitmq_consumer.py M
57 lab2 > app > rabbitmq_consumer.py > ...
58
59 166     # Использует методом subscribe
60 167     # подписчиком сообщений о заказах
61 168     @broker.subscribe("order")
62 169     async def subscribe_order_create(
63 170         order_data: OrderCreate,
64 171         session: Annotated[AsyncSession, Depends(get_db_session)],
65 172         order_service: Annotated[OrderService, Depends(get_order_service)],
66 173         product_service: Annotated[ProductService, Depends(get_product_service)],
67 174     ) -> None:
68 175         ...
69 176
70 177     # Обработчик создания заказа через RabbitMQ.
71 178
72 179     Проверяет наличие всех товаров перед созданием заказа.
73 180     Если хотя бы один товар закончился (stock_quantity == 0), заказ не создается.
74 181
75 182     Args:
76 183         order_data: Данные для создания заказа
77 184         session: Сессия базы данных
78 185         order_service: Сервис для работы с заказами
79 186         product_service: Сервис для работы с продуктами
80 187
81 188     try:
82 189         logger.info(f"Received order create request: User ID={order_data.user_id}")
83 190
84 191         # Проверяет наличие всех товаров перед созданием заказа
85 192         out_of_stock_products = []
86 193         for item in order_data.items:
87 194             product = await product_service.get_by_id(session, item.product_id)
88 195             if not product:
89 196                 logger.error(
90 197                     f"Product with ID={item.product_id} not found, order rejected", item.product_id
91 198             )
92 199             return
93 200         if product.stock_quantity == 0:
94 201             out_of_stock_products.append(
95 202                 {"product_id": product.id, "name": product.name}
96 203             )
97 204
98 205         # Если есть товары, которые закончились, отклоняем заказ
99 206         if out_of_stock_products:
100 207             logger.warning(
101 208                 f"Order rejected: products out of stock: {out_of_stock_products}"
102 209             )
103 210             return
104 211
105 212         # Создаем заказ (OrderService.create уже проверяет достаточность количества)
106 213         order = await order_service.create(session, order_data)
107 214         logger.info(
108 215             f"Order created successfully: ID={order.id}, User ID={order.user_id}, Total={order.total}"
109 216             , order.id, order.user_id,
110 217             )
111
112
113 Problems Output Debug Console Terminal Ports
114 [WARNING] Unstaged files detected.
115 [INFO] Stashing unstaged files to /Users/zmediniray/codex/pre-commit/patch1764589587-9187.
116 [INFO] Black
117 [INFO] isort
118 [INFO] pycodestyle
119 [INFO] flake8
120 [INFO] [87fa18c] feat(rabbitmq): реализация обработки сообщений о продажах
121 [INFO] 5 files changed, 166 insertions(+)
122 [INFO] create mode 100644 lab2/rabbitmq_consumer.py
123 [INFO] (lab2) x 1 lab2 git:(main) x [1]
124
125 Timeline
126 main* 0-4 1+ App_Dev_sem_1 0 0 0
127
128 Cursor Tab -> Not Committed Yet 0 Ln 167, Col 1 Spaces: 4 UTF-8 LF () Python 3.13.9 ({venv}:venv) Go Live 0
```

Для обработки заказов была разработана схема `OrderUpdateMessage` с валидацией через Pydantic и настроены провайдеры DI для репозитория и сервиса заказов. Подписчик на создание заказа (`subscribe_order_create`), слушающий очередь

`order`, реализует сложную логику проверки: перед созданием он убеждается в существовании каждого товара и наличии достаточного количества на складе (`stock_quantity > 0`). Если хотя бы одна позиция отсутствует, заказ отклоняется еще до обращения к сервису создания, что дополняется вторичной проверкой внутри `OrderService` и полным логированием процесса. Также реализован подписчик `subscribe_order_update` для обновления статусов заказов через очередь `order_update`.

6: Интеграция RabbitMQ consumer с приложением

```

rabitmq_worker.py - App_Dev_sem_1
lab2 > rabbitmq_worker.py > ...
1 """Скрипт для запуска RabbitMQ consumer в отдельном процессе."""
2
3 import asyncio
4 import logging
5
6 from app.rabbitmq_consumer import app
7
8 # Настройка логирования
9 logging.basicConfig(
10     level=logging.INFO,
11     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
12 )
13 logger = logging.getLogger(__name__)
14
15
16 async def main() -> None:
17     """Главная функция для запуска RabbitMQ consumer."""
18     logger.info("Starting RabbitMQ worker...")
19     try:
20         await app.run()
21     except KeyboardInterrupt:
22         logger.info("RabbitMQ worker stopped by user")
23     except Exception as e:
24         logger.error("Error running RabbitMQ worker: %s", e, exc_info=True)
25     raise
26
27
28 if __name__ == "main__":
29     asyncio.run(main())
30
31

```

```

docker-compose.yml - App_Dev_sem_1
lab2 > docker-compose.yml > ...
1 services:
2     app:
3         depends_on:
4             - db
5         command: ["uvicorn", "app.main:app"]
6         restart: unless-stopped
7
8     rabbitmq_worker:
9         build:
10            context: .
11            dockerfile: Dockerfile
12            container_name: rabbitmq_worker_lab2
13         environment:
14             - DATABASE_URL=postgresql+asyncpg://$POSTGRES_USER:$POSTGRES_PASSWORD:$admin@$db:5432/$POSTGRES_DB:-lab_db2
15             - DB_HOST=$DB_HOST:-db
16             - DB_PORT=$DB_PORT:-5432
17             - RABBITMQ_HOST=$RABBITMQ_HOST:-rabbitmq
18             - RABBITMQ_PORT=$RABBITMQ_PORT:-5672
19             - RABBITMQ_VHOST=$RABBITMQ_VHOST:-local
20             - RABBITMQ_USER=$RABBITMQ_USER:-guest
21             - RABBITMQ_PASSWORD=$RABBITMQ_PASSWORD:-guest
22         volumes:
23             - ./app:/app
24             - ./rabbitmq_worker.py:/app/rabbitmq_worker.py
25         networks:
26             - pg_network_lab2
27         depends_on:
28             - db:
29                 condition: service_healthy
30             rabbitmq:
31                 condition: service_started
32             command: ["uvicorn", "app.main:app"]
33             restart: unless-stopped
34
35 volumes:
36     db_data_lab2:
37     pgadmin_data_lab2:
38     rabbitmq_data:
39
40 networks:
41     pg_network_lab2:
42         driver: bridge

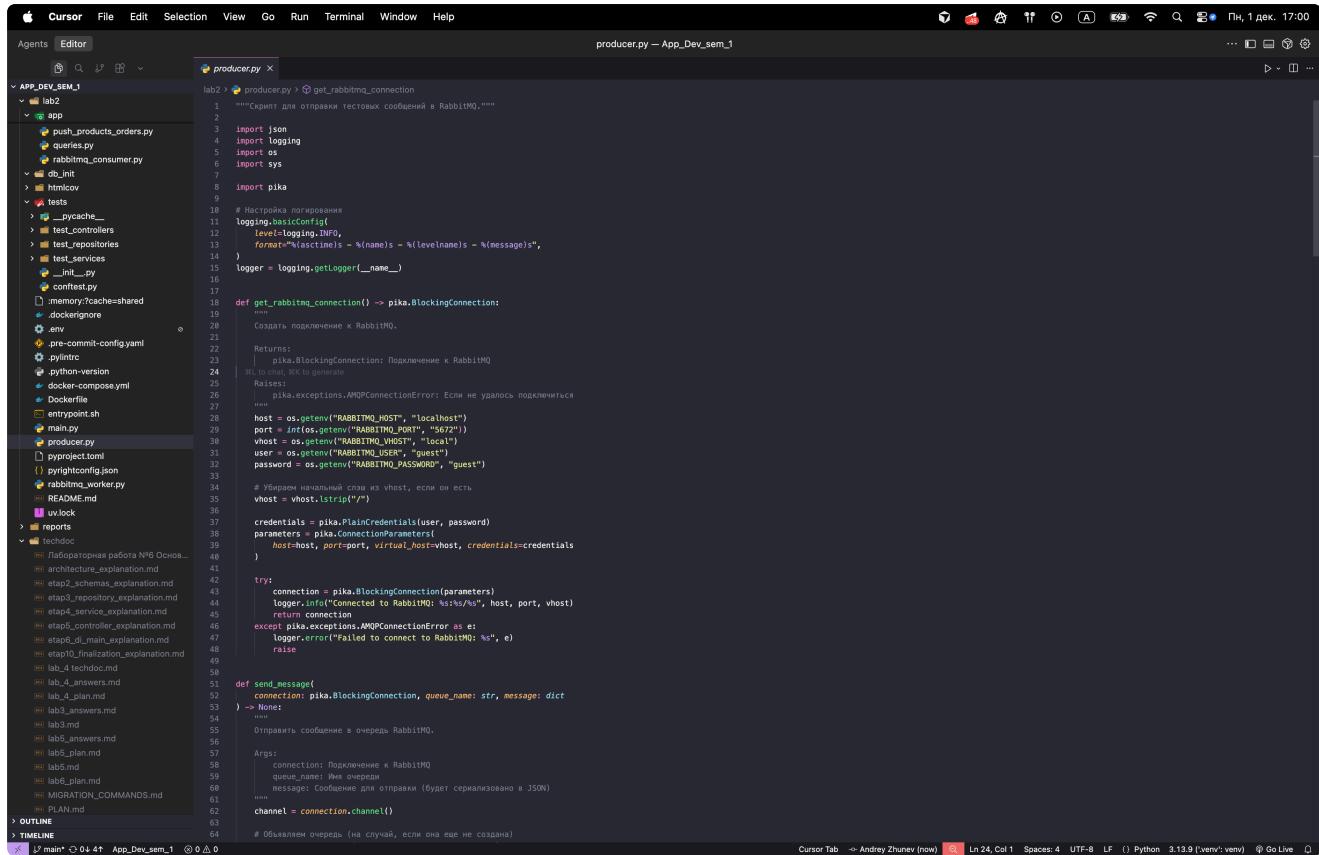
```

Для запуска потребителя сообщений был создан отдельный скрипт

`rabbitmq_worker.py`, который импортирует приложение из `app.rabbitmq_consumer` и запускает его через `asyncio.run()`, обеспечивая обработку прерываний и логирование. Файл `docker-compose.yml` был дополнен сервисом `rabbitmq_worker`, который использует общий Dockerfile и зависимости, но запускается командой `uv run python rabbitmq_worker.py` после готовности базы данных и брокера. Такая архитектура позволяет запускать REST API и обработчик очередей в изолированных

контейнерах, обеспечивая масштабируемость и независимое управление процессами.

7: Создание producer скрипта



```
#!/usr/bin/env python
# Скрипт для отправки тестовых сообщений в RabbitMQ.
# Использует пакет pika.

import json
import logging
import os
import sys
import pika

# Настройки подключения
logging.basicConfig()
level=logging.INFO,
format="%(asctime)s - %(name)s - %(levelname)s - %(message)s",
)
logger = logging.getLogger(__name__)

def get_rabbitmq_connection() -> pika.BlockingConnection:
    """
    Создать подключение к RabbitMQ.
    """
    Returns:
        | pika.BlockingConnection: Подключение к RabbitMQ
    Raises:
        |
            pika.exceptions.AMQPConnectionError: Если не удалось подключиться
    """
    host = os.getenv("RABBITMQ_HOST", "localhost")
    port = int(os.getenv("RABBITMQ_PORT", "5672"))
    vhost = os.getenv("RABBITMQ_VHOST", "/local")
    user = os.getenv("RABBITMQ_USER", "guest")
    password = os.getenv("RABBITMQ_PASSWORD", "guest")

    # Убираем начальный слэш из vhost, если он есть
    vhost = vhost.lstrip("/")

    credentials = pika.PlainCredentials(user, password)
    parameters = pika.ConnectionParameters(
        host=host, port=port, virtual_host=vhost, credentials=credentials
    )
    try:
        connection = pika.BlockingConnection(parameters)
        logger.info(f"Connected to RabbitMQ: {host}:{port}, host, {vhost}")
        return connection
    except pika.exceptions.AMQPConnectionError as e:
        logger.error(f"Failed to connect to RabbitMQ: {e}")
        raise

def send_message(
    connection: pika.BlockingConnection, queue_name: str, message: dict
) -> None:
    """
    Отправить сообщение в очередь RabbitMQ.
    Args:
        connection: Подключение к RabbitMQ
        queue_name: Имя очереди
        message: Сообщение для отправки (будет сериализовано в JSON)
    """
    channel = connection.channel()

    # Объявляем очередь (на случай, если она еще не создана)
```

Для тестирования системы разработан скрипт `producer.py`, использующий библиотеку `pika` для синхронной отправки сообщений с гарантией доставки (`delivery_mode=2`) и предварительной проверкой подключения. Скрипт включает функцию `create_test_products`, генерирующую 5 тестовых товаров (ноутбук, мышь, клавиатура и др.) и отправляющую их в очередь `product`. Также реализована функция `create_test_orders`, создающая 3 заказа с несколькими позициями, используя ID пользователя и адреса из переменных окружения, при этом скрипт предусматривает задержки между операциями для корректной обработки данных консьюмером.

8: Тестирование интеграции RabbitMQ

В ходе тестирования все контейнеры были успешно запущены, а сервис RabbitMQ стал доступен на порту 15672 с корректно созданными очередями. Для проведения интеграционного теста были вручную созданы необходимые сущности (пользователь и адрес), после чего был запущен скрипт `producer.py`. Скрипт отработал без ошибок, что подтверждается появлением записей о товарах и заказах в базе данных, продемонстрированных на скриншотах.

```

=> [app] resolving provenance for metadata file
[+] Running 8/8
  ✓ lab2-rabbitmq_worker      Built
  ✓ lab2-app                  Built
  ✓ Volume lab2_rabbitmq_data Created
  ✓ Container rabbitmq_lab2   Started
  ✓ Container rabbitmq_worker_lab2 Started
  ✓ Container app_lab3        Started
  ✓ Container db_postgres_lab2 Healthy
  ✓ Container pgadmin4_lab2   Started
● (lab2) ➔ lab2 git:(main) ✘ docker compose ps
    NAME          IMAGE           COMMAND
    STATUS         PORTS
    app_lab3      lab2-app       "../entrypoint.sh uv ..."
    Up 7 seconds  0.0.0.0:8000->8000/tcp, [::]:8000->8000/tcp
    db_postgres_lab2 postgres:17.6-bookworm "docker-entrypoint.s..."
    Up 13 seconds (healthy)  0.0.0.0:5433->5432/tcp, [::]:5433->5432/tcp
    pgadmin4_lab2  dpage/pgadmin4:9.8.0  "/entrypoint.sh"
    Up 7 seconds  0.0.0.0:8081->80/tcp, [::]:8081->80/tcp
    rabbitmq_lab2  rabbitmq:3-management "docker-entrypoint.s..."
    Up 13 seconds  0.0.0.0:5672->5672/tcp, [::]:5672->5672/tcp, 0.0.0.0:15672->15672/tcp, [::]:15672->15672/tcp
    rabbitmq_worker_lab2 lab2-rabbitmq_worker  "../entrypoint.sh uv ..."
    Up 7 seconds  8000/tcp
● (lab2) ➔ lab2 git:(main) ✘

```

⌘K to generate command

The screenshot shows the Docker Desktop application window. On the left is a sidebar with various project and learning center items. The main area displays a list of running Docker containers:

Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
lab2	-	-	-	0%	25 seconds ago	[Stop] [More] [Delete]
db_postgres_lab2	fc003a4fd062	postgres:1	5433:5432	0%	30 seconds ago	[Stop] [More] [Delete]
pgadmin4_lab2	2bfad2d8b4ff	dpage/pgadmin4:9.8.0	8081:80	0%	25 seconds ago	[Stop] [More] [Delete]
rabbitmq_lab2	4e70a77cd82	rabbitmq:3	15672:15672	0%	30 seconds ago	[Stop] [More] [Delete]
rabbitmq_worker_lab2	da45e05b27ff	lab2-rabbitmq_worker	8000:8000	0%	25 seconds ago	[Stop] [More] [Delete]
app_lab3	4ee14281bde2	lab2-app	8000:8000	0%	25 seconds ago	[Stop] [More] [Delete]

Below the container list, there are sections for 'Walkthroughs' and 'Multi-container applications'.

Все контейнеры поднялись.

Safari - Comet | Файл | Правка | Вид | Ассистент | История | Закладки | Профили | Вкладка | Окно | Справка

localhost / RabbitMQ Management

RabbitMQ TM RabbitMQ 3.13.7 Erlang 26.2.5.16

Overview Connections Channels Exchanges Queues and Streams Admin

Refreshed 2025-12-01 17:06:46 Refresh every 5 seconds Virtual host All Cluster rabbit@4e70a7cd82 User guest Log out

Overview

Totals

Queued messages last minute ?

Ready: 0 Unacked: 0 Total: 0

Message rates last minute ?

Disk read: 0.00/s Disk write: 0.00/s

Global counts ?

Connections: 1 Channels: 1 Exchanges: 7 Queues: 4 Consumers: 4

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Cores	Info	Reset stats	+/-
rabbit@4e70a7cd82	44 1048576 available	1 943629 available	466 1048576 available	162 MB 3.1 GB high watermark	207 GB 48 MB low watermark	1m 17s	10	basic 2 rss	This node All nodes	+/-

Churn statistics Ports and contexts Export definitions Import definitions

HTTP API Documentation Tutorials New releases Commercial edition Commercial support Discussions Discord Plugins GitHub

RabbitMQ TM RabbitMQ 3.13.7 Erlang 26.2.5.16

Overview Connections Channels Exchanges Queues and Streams Admin

Queues

All queues (4)

Pagination

Page 1 of 1 - Filter: Regex ?

Overview					Messages			Message rates			+/-
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
local	order	classic	Args	running	0	0	0				
local	order_update	classic	Args	running	0	0	0				
local	product	classic	Args	running	0	0	0				
local	product_update	classic	Args	running	0	0	0				

Add a new queue

HTTP API Documentation Tutorials New releases Commercial edition Commercial support Discussions Discord Plugins GitHub

Rabbitmq на порту 15672 тоже поднят корректно. Нужные очереди созданы

```
        }
    {"status_code":404,"detail":"Not Found"}%
● (lab2) ✘ lab2 git:(main) ✘ docker compose exec db psql -U admin -d lab_db2 -c "SELECT id, user_id, street, city FROM addresses LIMIT 5;
"
   id | user_id | street | city
   +----+-----+-----+
(0 rows)

● (lab2) ✘ lab2 git:(main) ✘ docker compose exec db psql -U admin -d lab_db2 -c "INSERT INTO addresses (user_id, street, city, state, zip_code, country, is_primary, created_at, updated_at) VALUES (1, '123 Test Street', 'Test City', 'TS', '12345', 'Test Country', true, NOW(), NOW()) RETURNING id;"%
   id
   +----+
      1
(1 row)

INSERT 0 1
● (lab2) ✘ lab2 git:(main) ✘ docker compose exec db psql -U admin -d lab_db2 -c "SELECT id, user_id, street, city FROM addresses LIMIT 5;
"
   id | user_id | street | city
   +----+-----+-----+
      1 |       1 | 123 Test Street | Test City
(1 row)

○ (lab2) ✘ lab2 git:(main) ✘ █

```

Создал адрес и пользователя напрямую для тестирования

Запустил producer скрипт для создания заказов - скрипт отработал без ошибок

```
producer.py -- App_Dev_sem_1
producer.py M
lab2> producer.py > create_test_products
    78 def create_test_products(connection: pika.BlockingConnection) -> list[dict]:
    79     j
    80
    81     lab2 git:(main) x docker compose exec app uv run python producer.py
2025-12-01 12:17:04,119 - INFO - pika.adapters.utils.io_services_utils - INFO - Socket connected: <socket.socket fd=6, family=2, type=1, proto=6, laddr='('172.18.0.2', 5672)>
2025-12-01 12:17:04,119 - INFO - pika.adapters.utils.io_services_utils - INFO - Socket connected: <socket.socket fd=4, family=2, type=1, proto=6, laddr='('172.18.0.2', 5672)>
2025-12-01 12:17:04,119 - INFO - pika.adapters.utils.connection_workflow - INFO - Streaming transport linked up: <pika.adapters.utils.io_services_utils.AsyncPainlessTransport object at 0xffff8edc5160> params=<ConnectionParameters host=rabbitmq port=5672 virtualHost=/local ssl=False>
2025-12-01 12:17:04,121 - INFO - pika.adapters.utils.connection_workflow - INFO - AMQConnector - reporting success: <selectConnection OPEN transport=<pika.adapters.utils.io_services_utils._AsyncPainlessTransport object at 0xffff8edc5160> params=<ConnectionParameters host=rabbitmq port=5672 virtualHost=/local ssl=False>>
2025-12-01 12:17:04,121 - INFO - pika.adapters.utils.connection_workflow - INFO - AMQConnectionWorkflow - reporting success: <selectConnection OPEN transport=<pika.adapters.utils.io_services_utils._AsyncPainlessTransport object at 0xffff8edc5160> params=<ConnectionParameters host=rabbitmq port=5672 virtualHost=/local ssl=False>>
2025-12-01 12:17:04,121 - INFO - pika.adapters.blocking_connection - INFO - Connection workflow succeeded: <selectConnection OPEN transport=<pika.adapters.utils.io_services_utils._AsyncPainlessTransport object at 0xffff8edc5160> params=<ConnectionParameters host=rabbitmq port=5672 virtualHost=/local ssl=False>>
2025-12-01 12:17:04,121 - INFO - pika.adapters.blocking_connection - INFO - Connection created: <Channel(0) on<pika.adapters.blocking_connection>>
2025-12-01 12:17:04,121 - INFO - pika.adapters.blocking_connection - INFO - Channel(0) created: <Channel(0) on<pika.adapters.blocking_connection>>
2025-12-01 12:17:04,121 - INFO - pika.adapters.blocking_connection - INFO - Creating test products...
2025-12-01 12:17:04,121 - INFO - pika.adapters.blocking_connection - INFO - Creating 5 products...
2025-12-01 12:17:04,121 - INFO - pika.adapters.blocking_connection - INFO - Created channel=1
2025-12-01 12:17:04,122 - INFO - pika.adapters.blocking_connection - INFO - Message sent to queue 'product': {'name': 'Laptop Dell XPS 15', 'description': 'High-performance laptop with 16GB RAM and 512GB SSD', 'price': 1299.99, 'stock_quantity': 10}
2025-12-01 12:17:04,122 - INFO - pika.adapters.blocking_connection - INFO - Message sent to queue 'product': {'name': 'Wireless Mouse Logitech MX Master 3', 'description': 'Ergonomic wireless mouse with advanced tracking', 'price': 99.99, 'stock_quantity': 25}
2025-12-01 12:17:04,123 - INFO - pika.adapters.blocking_connection - INFO - Product created: Laptop Dell XPS 15
2025-12-01 12:17:04,123 - INFO - pika.adapters.blocking_connection - INFO - Created channels=3
2025-12-01 12:17:04,123 - INFO - pika.adapters.blocking_connection - INFO - Message sent to queue 'product': {'name': 'Mechanical Keyboard Keychron K8', 'description': 'Wireless mechanical keyboard with RGB backlight', 'price': 89.99, 'stock_quantity': 15}
2025-12-01 12:17:04,123 - INFO - pika.adapters.blocking_connection - INFO - Product created: Mechanical Keyboard Keychron K8
2025-12-01 12:17:04,123 - INFO - pika.adapters.blocking_connection - INFO - Created channel=4
2025-12-01 12:17:04,123 - INFO - pika.adapters.blocking_connection - INFO - Message sent to queue 'product': {'name': 'USB-C Hub 7-in-1', 'description': 'Multi-port USB-C hub with HDMI, USB 3.0, and SD card reader', 'price': 49.99, 'stock_quantity': 30}
2025-12-01 12:17:04,124 - INFO - pika.adapters.blocking_connection - INFO - Product created: USB-C Hub 7-in-1
2025-12-01 12:17:04,124 - INFO - pika.adapters.blocking_connection - INFO - Created channel=5
2025-12-01 12:17:04,124 - INFO - pika.adapters.blocking_connection - INFO - Message sent to queue 'product': {'name': 'External SSD Samsung T7 1TB', 'description': 'Fast portable SSD with USB 3.2 Gen 2', 'price': 149.99, 'stock_quantity': 20}
2025-12-01 12:17:04,124 - INFO - pika.adapters.blocking_connection - INFO - Product created: External SSD Samsung T7 1TB
2025-12-01 12:17:04,124 - INFO - pika.adapters.blocking_connection - INFO - Successfully created 5 products
2025-12-01 12:17:06,125 - INFO - pika.adapters.blocking_connection - INFO - Waiting 2 seconds before creating orders...
2025-12-01 12:17:06,125 - INFO - pika.adapters.blocking_connection - INFO - Creating test orders...
2025-12-01 12:17:06,125 - INFO - pika.adapters.blocking_connection - INFO - Creating 3 orders...
2025-12-01 12:17:06,126 - INFO - pika.adapters.blocking_connection - INFO - Using user_id=1, delivery_address_id=1
2025-12-01 12:17:06,126 - INFO - pika.adapters.blocking_connection - INFO - Warning: Make sure user id=1 and delivery address id=1 exist in the database!
2025-12-01 12:17:06,126 - INFO - pika.adapters.blocking_connection - INFO - Created channel=6
2025-12-01 12:17:06,126 - INFO - pika.adapters.blocking_connection - INFO - Message sent to queue 'order': {'user_id': 1, 'delivery_address_id': 1, 'items': [{"product_id": 1, "quantity": 1}, {"product_id": 2, "quantity": 1}]}
2025-12-01 12:17:06,126 - INFO - pika.adapters.blocking_connection - INFO - Order 1 created with 2 items
2025-12-01 12:17:06,129 - INFO - pika.adapters.blocking_connection - INFO - Created channel=7
2025-12-01 12:17:06,130 - INFO - pika.adapters.blocking_connection - INFO - Message sent to queue 'order': {'user_id': 1, 'delivery_address_id': 1, 'items': [{"product_id": 3, "quantity": 2}, {"product_id": 4, "quantity": 1}]}
2025-12-01 12:17:06,130 - INFO - pika.adapters.blocking_connection - INFO - Created channel=8
2025-12-01 12:17:06,130 - INFO - pika.adapters.blocking_connection - INFO - Message sent to queue 'order': {'user_id': 1, 'delivery_address_id': 1, 'items': [{"product_id": 5, "quantity": 1}, {"product_id": 2, "quantity": 1}, {"product_id": 4, "quantity": 1}]}
2025-12-01 12:17:06,131 - INFO - pika.adapters.blocking_connection - INFO - Order 2 created with 2 items
2025-12-01 12:17:06,131 - INFO - pika.adapters.blocking_connection - INFO - Order 3 created with 3 items
2025-12-01 12:17:06,131 - INFO - pika.adapters.blocking_connection - INFO - Successfully created 3 orders
2025-12-01 12:17:06,131 - INFO - pika.adapters.blocking_connection - INFO - All test data sent successfully!
```

КАК ВИДИМ, ЭТО ПОДТВЕРЖДАЕТСЯ ТЕМ, ЧТО ТОВАРЫ ДОБАВЛЕНЫ В ТАБЛИЦУ

The screenshot shows the pgAdmin 4 interface. In the top navigation bar, there are tabs for Comet, Файл (File), Правка (Edit), Вид (View), Ассистент (Assistant), История (History), Закладки (Bookmarks), Профили (Profiles), Вкладка (Tab), Окно (Window), and Справка (Help). The current tab is 'File'. Below the navigation bar is a toolbar with various icons for file operations like Open, Save, Print, and Copy.

The main window has a title bar 'localhost / pgAdmin 4' and a tab 'pgAdmin 4'. The left sidebar, titled 'Object Explorer', shows the database structure. It includes sections for Servers (1), Databases (2), Schemas (1), and Tables (6). The 'Tables' section is expanded, showing tables like 'addresses', 'alembic_version', 'order_items', and 'orders'. The 'orders' table is selected, and its structure is displayed in the central pane.

In the central pane, there is a 'Query' editor containing the following SQL code:

```
1 SELECT id, name, price, stock_quantity
2 FROM products
3 ORDER BY id;
```

Below the query editor is a 'Data Output' pane displaying the results of the query:

	id [PK] integer	name character varying	price double precision	stock_quantity integer
1	1	USB-C Hub 7-in-1	49.99	29
2	2	Wireless Mouse Logitech MX Master	99.99	24
3	3	Laptop Dell XPS 15	1299.99	8
4	4	Mechanical Keyboard Keychron K8	89.99	13
5	5	External SSD Samsung T7 1TB	149.99	19

The bottom status bar indicates 'Total rows: 5' and 'Query complete 00:00:00.101'.

The screenshot shows a PostgreSQL database client interface. At the top, there's a toolbar with various icons for file operations, search, and navigation. Below the toolbar, a menu bar has 'Query' and 'Query History' selected. The main area contains a code editor with the following SQL query:

```
1 SELECT id, name, stock_quantity
2 FROM products
3 ORDER BY id;
```

Below the code editor is a 'Data Output' tab, which is currently active. It displays a table of product data:

	id [PK] integer	name character varying	stock_quantity integer
1	1	USB-C Hub 7-in-1	29
2	2	Wireless Mouse Logitech MX Maste...	24
3	3	Laptop Dell XPS 15	8
4	4	Mechanical Keyboard Keychron K8	13
5	5	External SSD Samsung T7 1TB	19

Ответы на вопросы

1. Что такое AMQP? Каковы его основные преимущества?

AMQP (Advanced Message Queuing Protocol) — это открытый и стандартизованный протокол, предназначенный для обмена сообщениями между различными приложениями в распределенной системе. Он не только определяет формат сообщений, но и строгие правила их передачи, что обеспечивает полную независимость от конкретного брокера или языка программирования. Ключевые преимущества AMQP включают его надежность которая обеспечивается через механизмы подтверждения (acknowledgments) доставки, и его гибкость в поддержке различных архитектур, таких как point-to-point и publish-subscribe. Благодаря этому, AMQP позволяет легко создавать масштабируемые и слабосвязанные системы, где

отправители и получатели могут работать, не зная напрямую о существовании друг друга.

2. В чем разница между очередями сообщений и шинами сообщений?

Основное различие между очередями и шинами сообщений лежит в их модели доставки. Очереди сообщений (Message Queues) работают по модели "point-to-point" (один-к-одному): каждое отправленное сообщение доставляется строго одному потребителю, после чего оно удаляется из очереди. Это идеально подходит для распределения задач между воркерами и обеспечения балансировки нагрузки. В отличие от них, Шины сообщений (Message Buses/Event Buses) используют модель "publish-subscribe" (один-ко-многим): одно отправленное сообщение (событие) доставляется всем активным подписчикам, которые заинтересованы в этом типе информации. Этот подход используется, когда необходимо уведомить множество независимых сервисов о произошедшем в системе событии.

3. Как обеспечить надежную доставку сообщений в RabbitMQ?

Для обеспечения надежной доставки сообщений в RabbitMQ необходимо задействовать несколько механизмов. Во-первых, нужно сделать Durable Queues (постоянные очереди) и Persistent Messages (постоянные сообщения): первые сохранят структуру очереди при перезапуске брокера, а вторые сохранят само сообщение на диске. Во-вторых, критически важно использовать Message Acknowledgments (Подтверждения): потребитель должен отправить явный `ack` после успешной обработки сообщения; если потребитель упадет, не отправив подтверждение, RabbitMQ вернет сообщение в очередь для повторной обработки другим воркером. Дополнительно можно использовать Publisher Confirms, чтобы отправитель получил подтверждение от брокера о том, что сообщение безопасно сохранено.

4. Что произойдет с сообщением, если consumer упадет во время обработки?

Поведение RabbitMQ полностью зависит от того, какой метод подтверждения используется. Если настроен Automatic Acknowledgment (`auto_ack=True`), сообщение считается обработанным сразу после получения. В этом случае, если потребитель упадет, сообщение будет безвозвратно потеряно, так как оно уже удалено из очереди. Однако, в большинстве надежных систем (например, в нашем проекте, где FastStream использует ручное подтверждение) применяется Manual Acknowledgment (`auto_ack=False`). Если потребитель падает до отправки `ack`, RabbitMQ обнаружит потерю соединения и, после таймаута, автоматически вернет

это сообщение обратно в очередь. Оно будет доставлено другому доступному потребителю, обеспечивая гарантию, что ни одно сообщение не будет потеряно, а обработка будет завершена.

5. Как обеспечить сохранность сообщений при перезапуске RabbitMQ?

Для обеспечения полной сохранности данных при перезапуске брокера необходимо сделать постоянными все три ключевых компонента: очереди, сообщения и обменики (exchanges). Очереди и обменики объявляются с параметром `durable=True`, что гарантирует сохранение их структуры на диске. Сообщения, в свою очередь, должны быть опубликованы с установкой свойства `delivery_mode=2` (Persistent), что заставляет брокер сохранить их содержимое на диске. Кроме того, в среде Docker необходимо использовать Volume для каталога данных RabbitMQ (`rabbitmq_data:/data`), чтобы само хранилище данных не терялось при перезапуске или обновлении контейнера.

6. Что такое TTL (Time To Live) для сообщений и как его настроить?

TTL (Time To Live) — это максимальное время, в течение которого сообщение может находиться в очереди до того, как оно будет автоматически удалено брокером. Это эффективный механизм для обработки временных данных и предотвращения застравания устаревших сообщений. Настроить TTL можно двумя основными способами: установить TTL для отдельного сообщения при его публикации (например, 60000 миллисекунд через `expiration` в свойствах) или установить TTL для всей очереди при ее создании через аргументы (`x-message-ttl`). Если указаны оба значения, RabbitMQ всегда использует наименьшее, чтобы гарантировать, что сообщение будет удалено не позднее заданного срока.