




ОНЛАЙН-ОБРАЗОВАНИЕ

Онлайн-образование

Меня хорошо видно && слышно?

Ставьте  , если все хорошо
Напишите в чат, если есть проблемы

Проверить, идет ли запись!

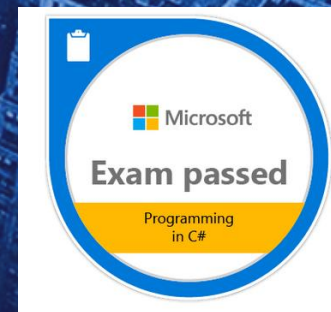


Брокеры сообщений на примере RabbitMQ

Приходько Роман

Приходько Роман

Старший .net разработчик В
к.т.н., доцент СевГУ



Маршрут вебинара

Общие понятия



RabbitMQ



Kafka

The background of the slide is a high-angle, blue-tinted aerial photograph of a city skyline, likely New York City, showing numerous skyscrapers and buildings. A semi-transparent blue band with a white geometric network pattern of dots and lines runs horizontally across the middle of the image, serving as a backdrop for the title.

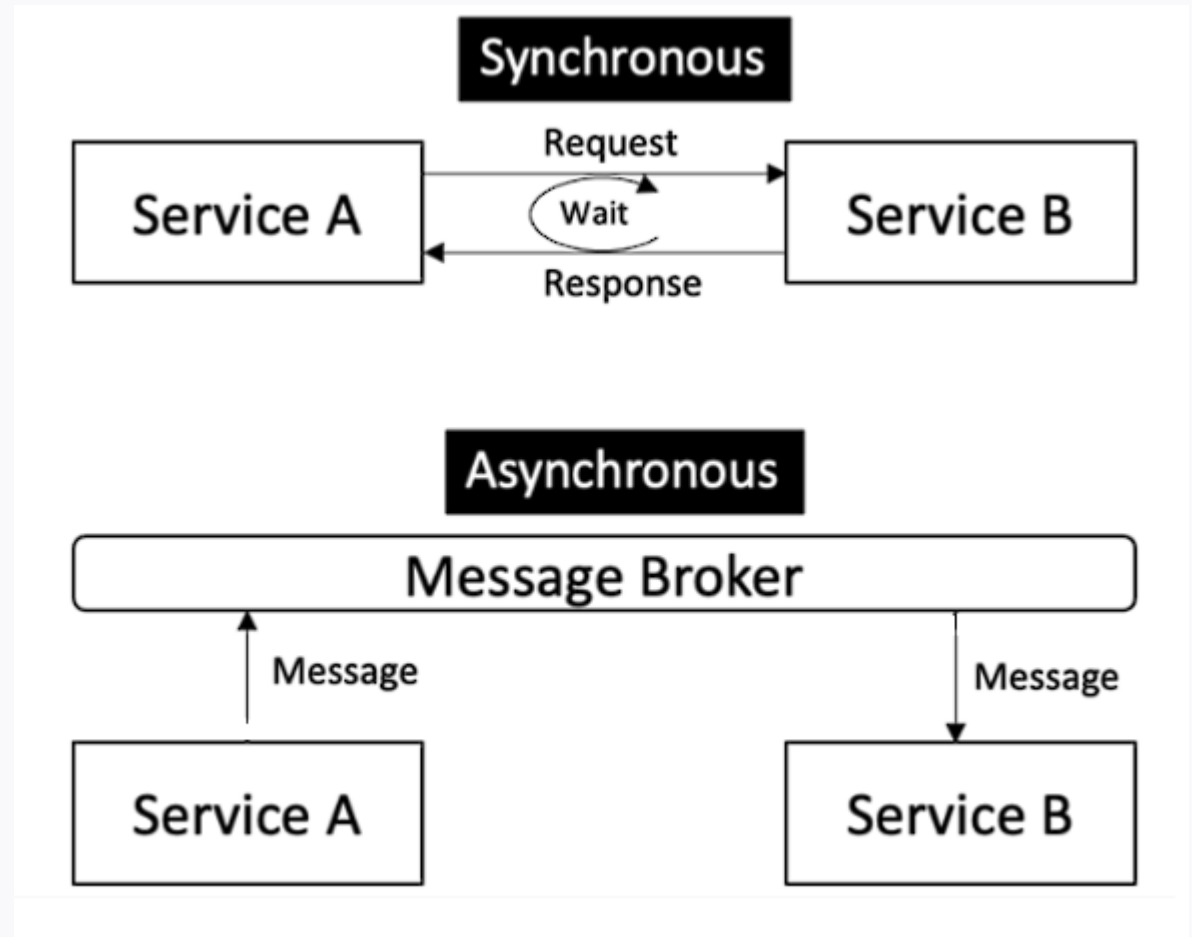
Общие понятия

Взаимодействия между микросервисами

Основные способы взаимодействий между сервисами в микросервисной архитектуре:

1. Синхронное взаимодействие
Отправитель ждет от получателя ответ на запрос.
Например, HTTP

2. Асинхронное взаимодействие
Отправитель не ждет от получателя ответ на запрос.
AMQP (Advanced Message Queuing Protocol)

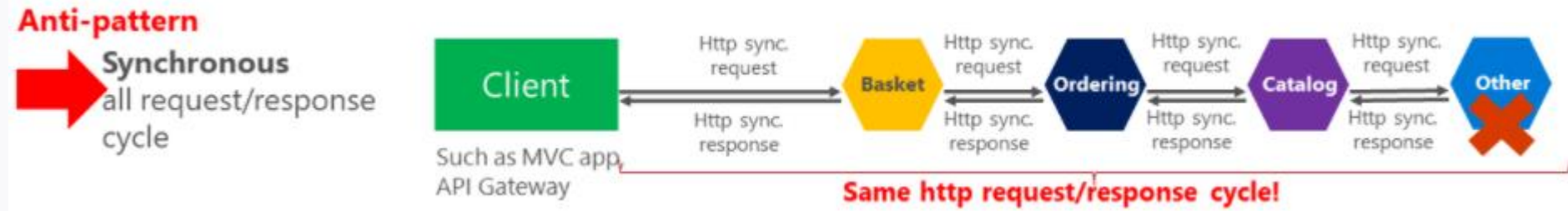


Взаимодействия между микросервисами

1. Синхронное взаимодействие

Отправитель ждет ответ на запрос

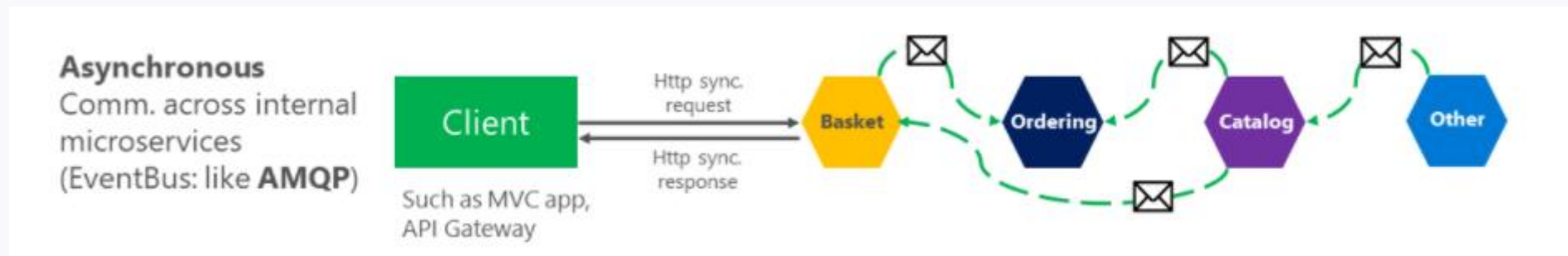
Например, HTTP



2. Асинхронное взаимодействие

Отправитель не ждет ответа на запрос.

AMQP (Advanced Message Queuing Protocol)



Брокер сообщений

Сервис, который принимает сообщения от клиента и осуществляет их маршрутизацию и постановку во временное хранилище (очередь/лог) по принципу FIFO.

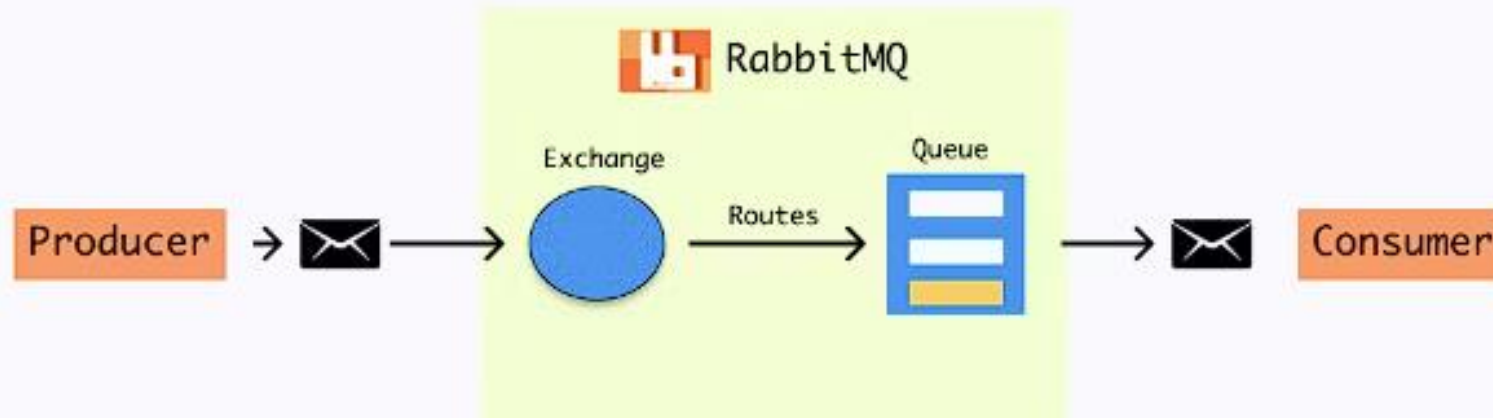
AMQP

AMQP – протокол, позволяющий приложениям обмениваться сообщениями через специальные сервисы – посредники, называемые брокерами сообщений.

<https://www.rabbitmq.com/tutorials/amqp-concepts.html>

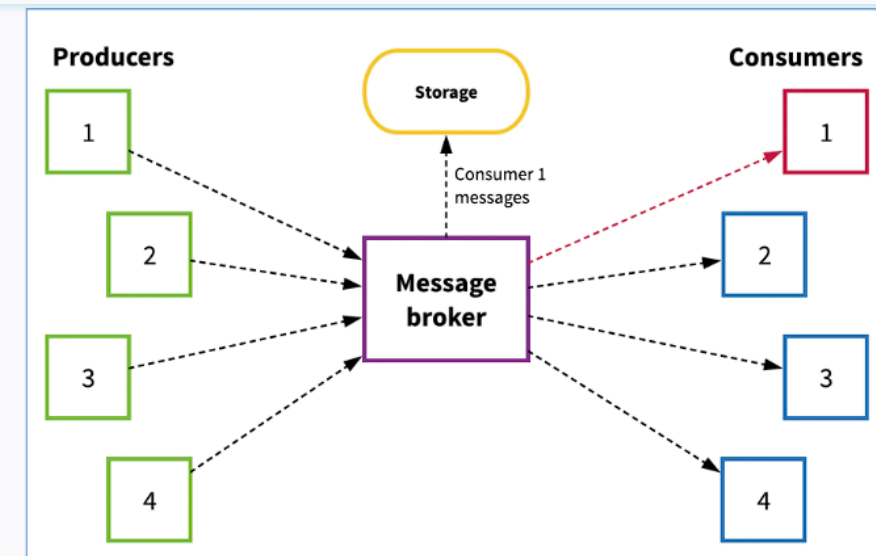
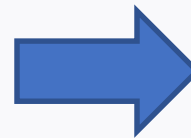
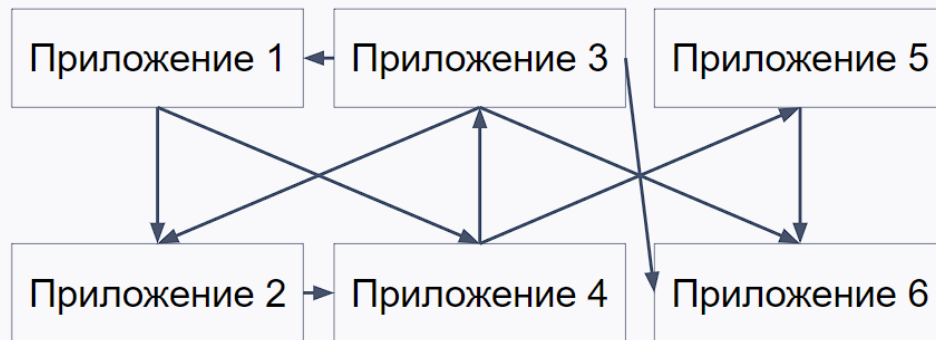
Основные функции брокера

1. Обеспечение возможности асинхронной обработки сообщений, достигающееся наличием очередей



Основные функции брокера

2. Централизованная маршрутизация



PUSH и PULL архитектуры брокеров

Получатель

PUSH Architecture

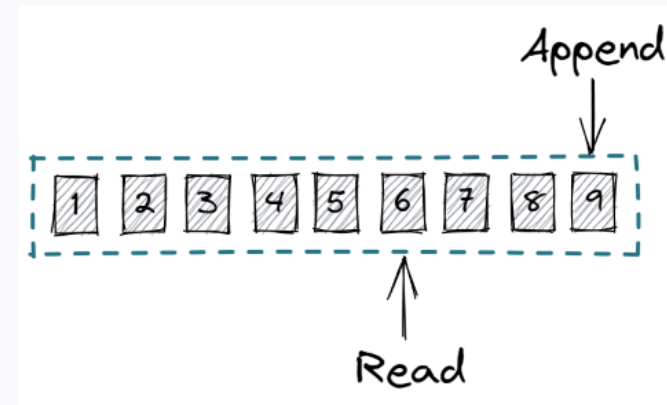
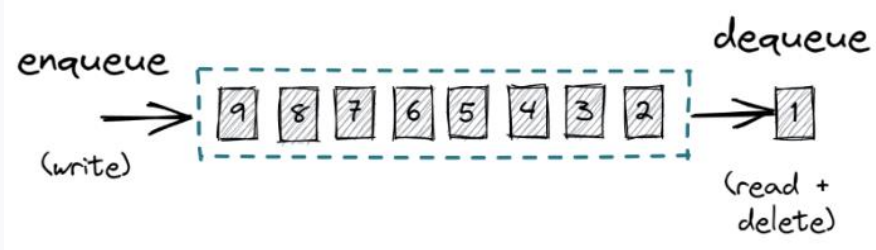


Получатель

PULL Architecture



Очередь и лог



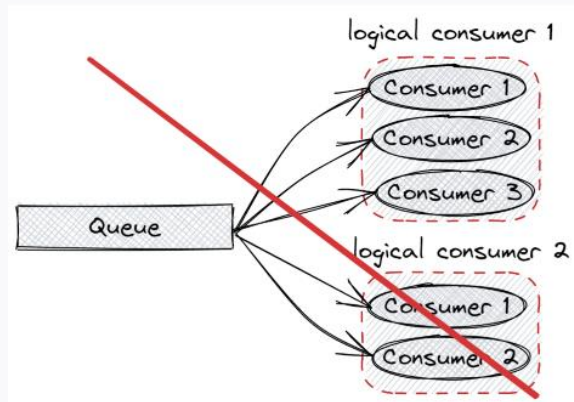
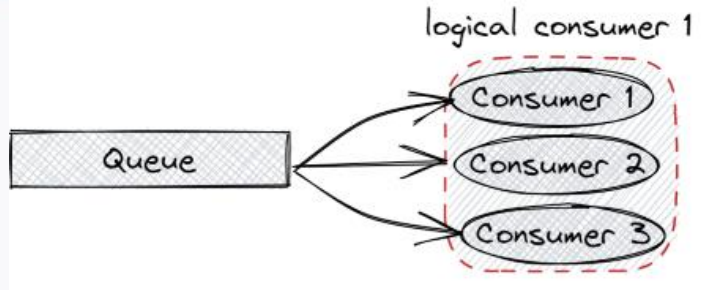
Очередь является временным хранилищем, лог - постоянным

Лог может только дополняться

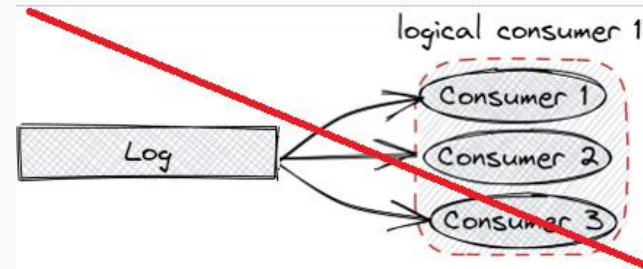
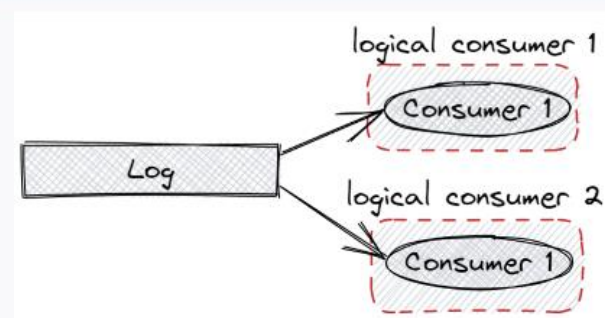
Сообщения персистентны

Сообщения неизменяемы

Не очередь, а лог

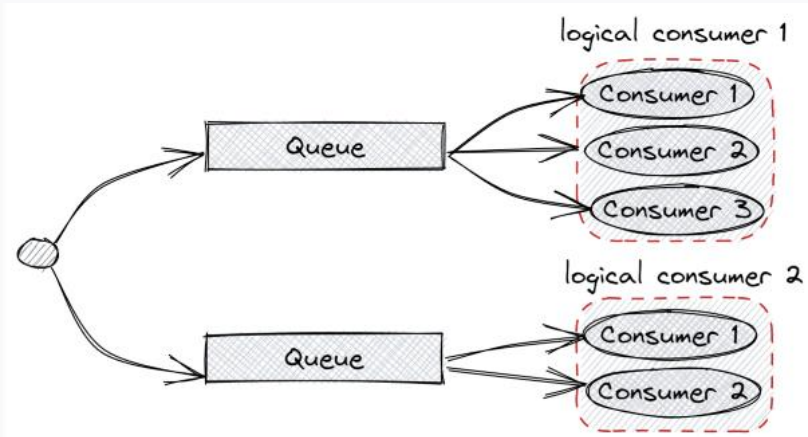


Очередь могут читать несколько физических консьюмеров, но не могут несколько логических



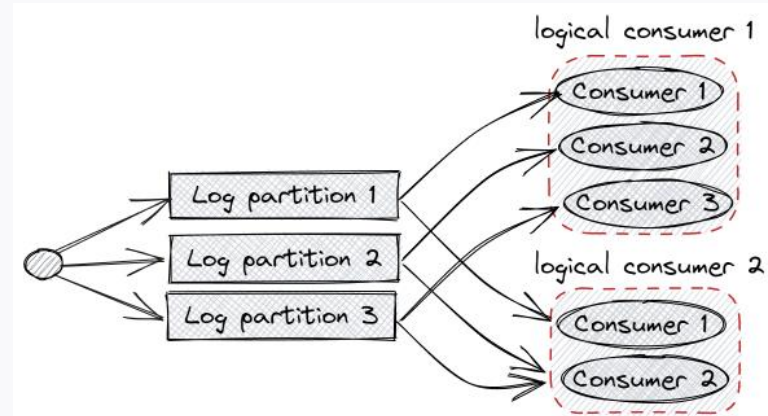
Лог могут читать несколько логических консьюмеров, но не могут несколько физических

Не очередь, а лог



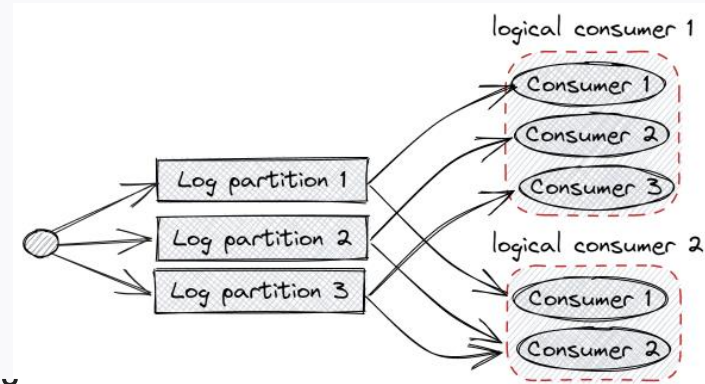
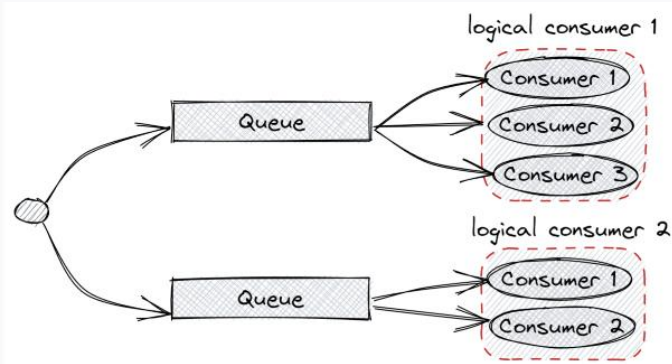
Решение

Решением для очереди является создание копии очереди для каждого логического консьюмера



Решением для лога является разбиение лога на части (НЕ копии) называемые партициями

Не очередь, а лог



Порядок сообщений

В очереди сообщения идут по порядку, а при обработке физическими консьюмерами порядок нарушается

Решение – синхронизация в логическом консьюмере (например, стейтмашина)

То есть встроенного решения нет

В логе сообщения идут по порядку, но при отправке в разные партиции порядок нарушается

Решение – отправлять сообщения, относящиеся к одной и той же сущности, в одну партицию

То есть встроенное решение есть

Некоторые реализации брокеров сообщений

1.RabbitMQ



2.Kafka



3.ZeroMQ



...

Клиентские библиотеки для работы с брокерами

Библиотеки для работы в .Net



- RabbitMQ.Client
- MassTransit.RabbitMQ
- NServiceBus
- EasyNetQ
- ...



- Confluent Kafka Client
- ...

Виды гарантий доставки

0.No guarantee – нет гарантий доставки

1.At most once (0 или 1 раз) – есть вероятность что сообщение будет потеряно

2.At least once (1 и более раз) – есть гарантия доставки (самый распространенный), но есть вероятность дублирования

3.Exactly once (строго 1 раз) – есть гарантия доставки и отсутствия дублирования. Сложно гарантировать, обеспечивается в основном с помощью паттернов инбокс, аутбокс и тд

Виды гарантий доставки

1. No guarantee

Реализуется установкой автоподтверждения доставки в true.

Сохранили в бд, не закоммитили, испытали сбой – обрабатывается дважды

Не сохранили в бд, закоммитили, испытали сбой – не обрабатывается

Недетерминированное поведение системы

2. At most once

Реализуется установкой автоподтверждения доставки в false, подтверждение делается **перед** логической обработкой.

Закоммитили, сохранили в бд – обрабатывается один раз

Закоммитили, испытали сбой, не сохранили в бд – не обрабатывается

Виды гарантий доставки

3. At least once

Реализуется установкой автоподтверждения доставки в false, подтверждение делается **после** обработки.

Сохранили в бд, закоммитили – обрабатывается один раз

Сохранили в бд, испытали сбой, не закоммитили, повторили – обрабатывается больше одного раза

4. Exactly once

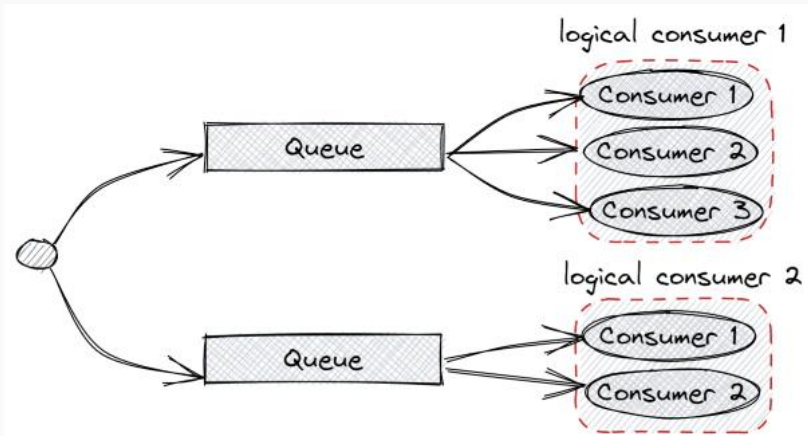
Реализуется созданием кастомного хранилища офсетов (Custom offset manager)

The background of the image is an aerial photograph of a city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that features a network of white lines and dots, resembling a mesh or a data network. The text "RabbitMQ" is centered in the middle of the image, in a white, sans-serif font.

RabbitMQ

Основа

RabbitMQ реализован на очередях



Как будем использовать

1. Развернутый в облаке

<https://www.cloudamqp.com/>

2. Развернутый в контейнере

Подключение помощью RabbitMQ.Client



Nuget: RabbitMQ.Client

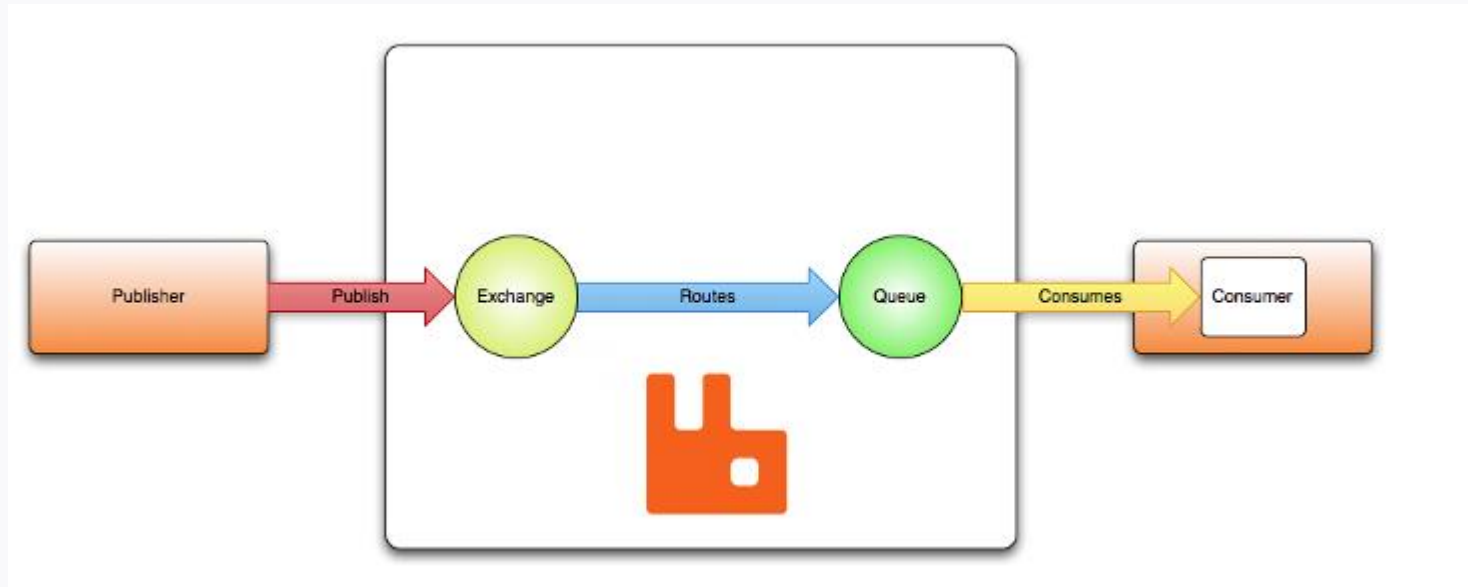
```
var cf = new ConnectionFactory();
var conn = cf.newConnection();

// the .NET client calls channels "models"
var ch = conn.CreateModel();

// do some work

// close the channel when it is no longer needed
ch.Close();
```

Основные понятия



Publisher (отправитель) – сервис, отправляющий сообщение

Consumer (потребитель) – сервис, принимающий сообщение

Exchange (обменник) – маршрутизатор, определяющий, в какую очередь должно быть отправлено сообщение

Queue (очередь) – место, где хранятся уже отправленные но еще не принятые сообщения по принципу FIFO

Адресно-маршрутная информация сообщения

1. Наименование обменника (exchange)
2. Маршрутный ключ (routing key)
3. Хедеры (headers)

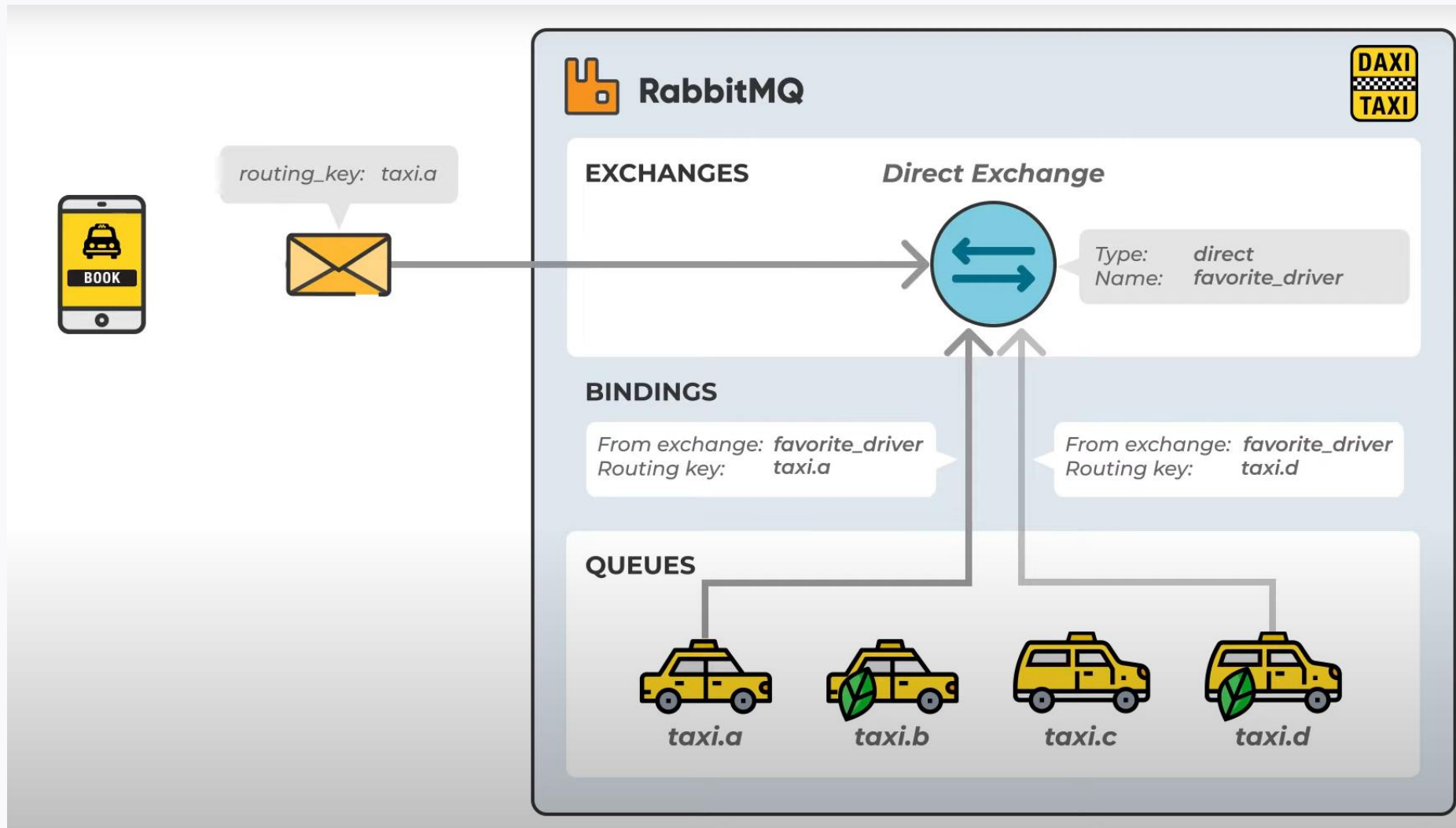
Nuget: **RabbitMQ.Client**

```
channel.BasicPublish(exchange: exchangeName,  
    routingKey: "cars.2",  
    basicProperties: null,  
    body: body);
```

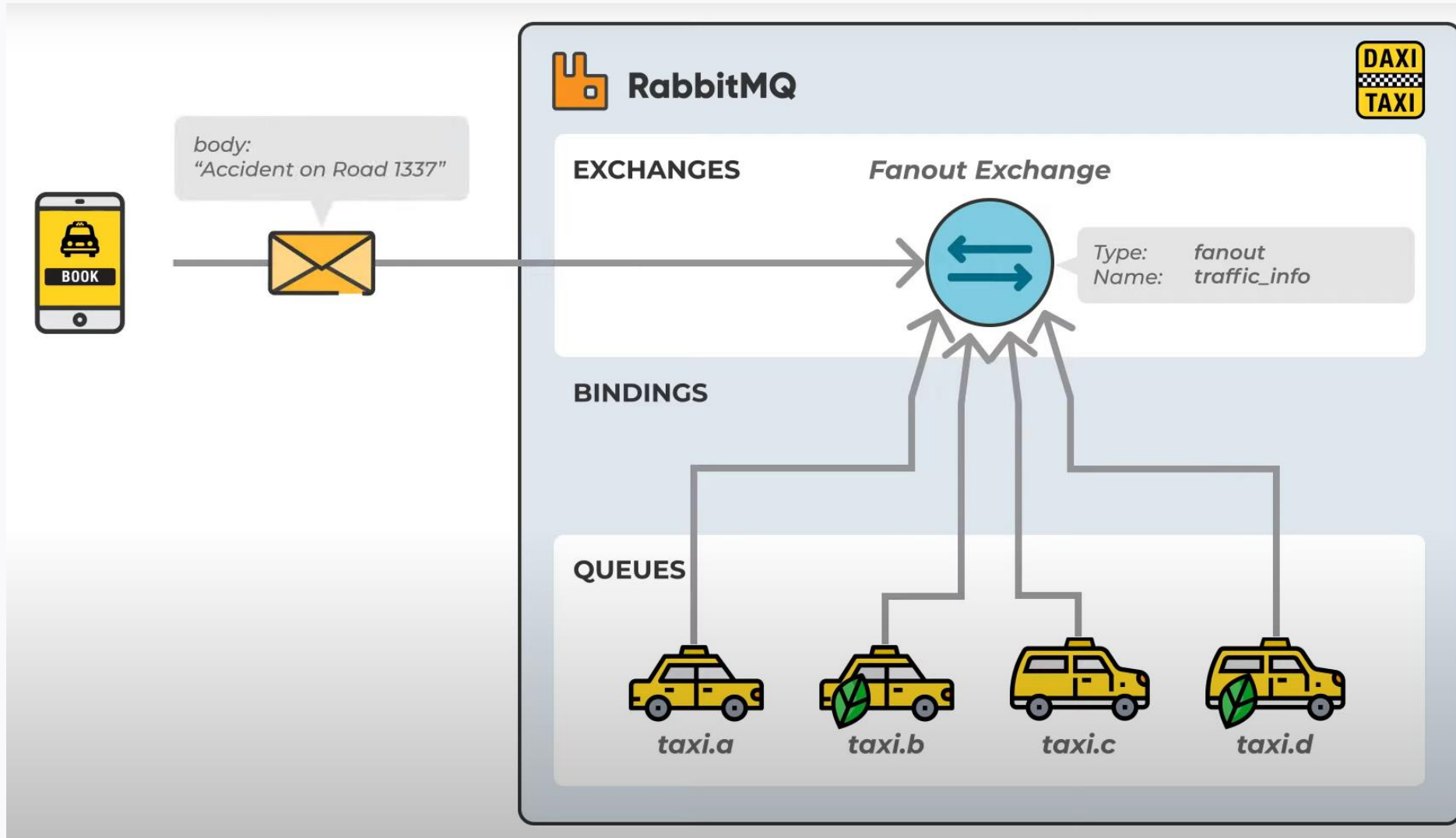

Виды Exchange

1. Direct
2. Fanout
3. Topic
4. Headers

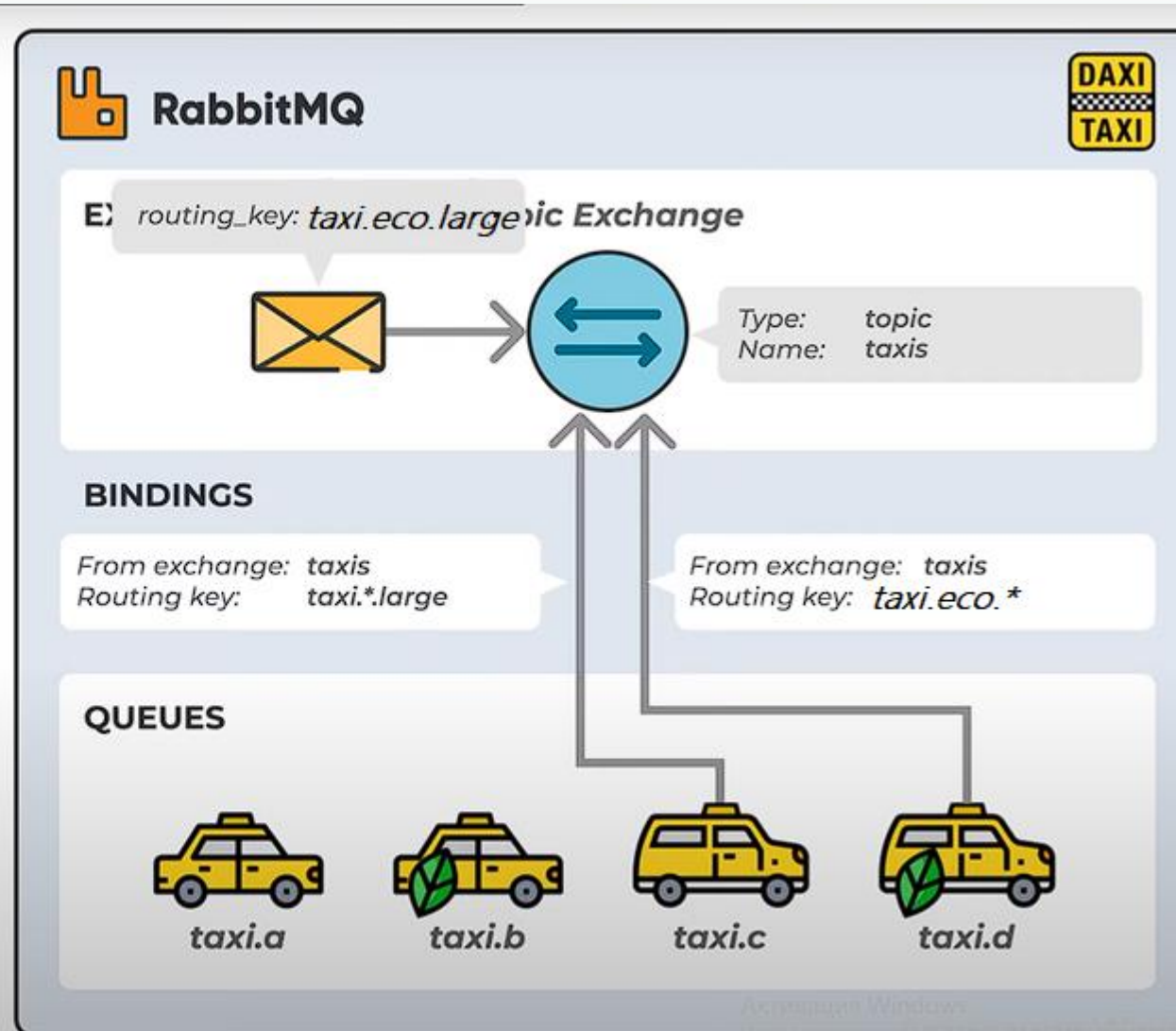
Direct Exchange



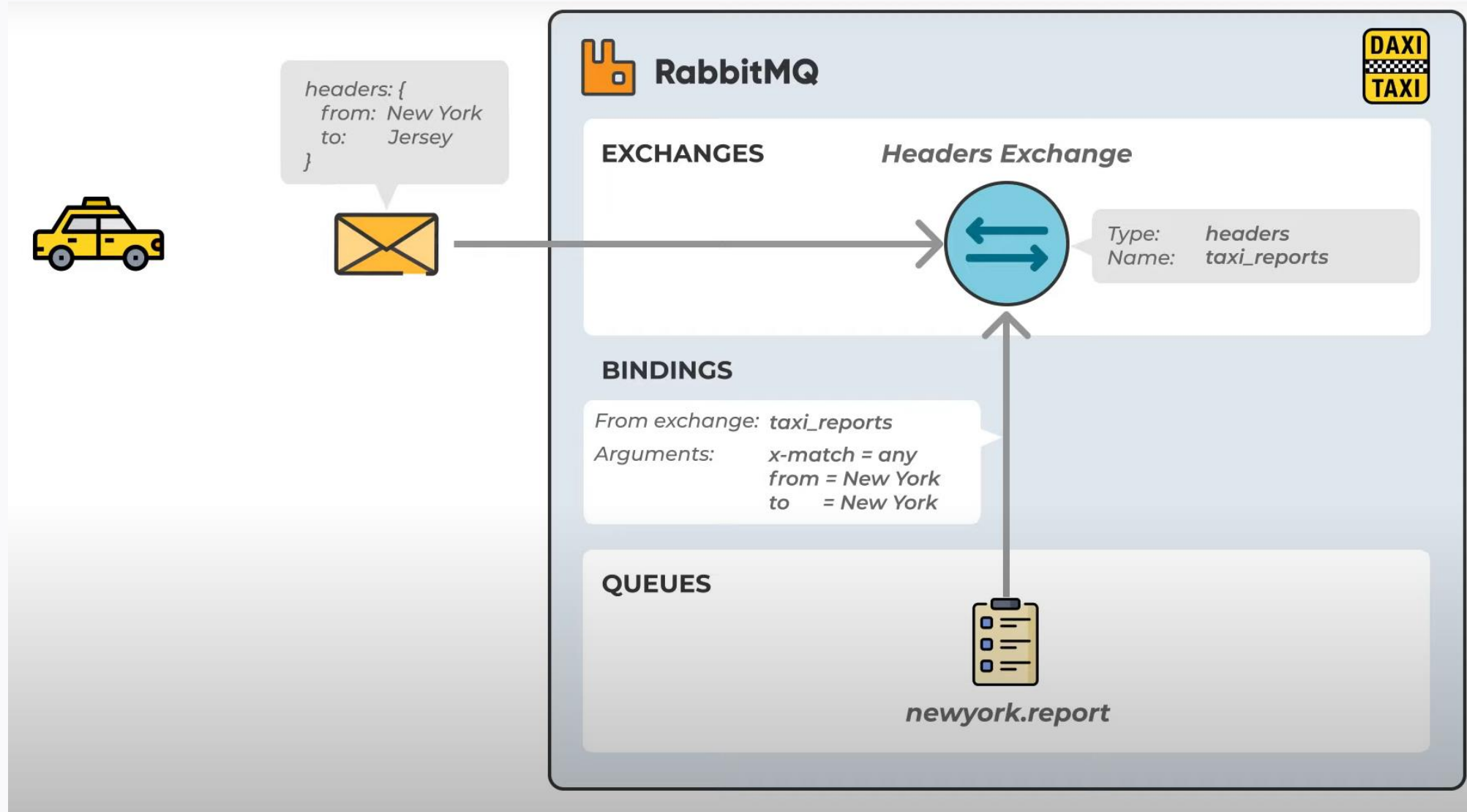
Fanout Exchange



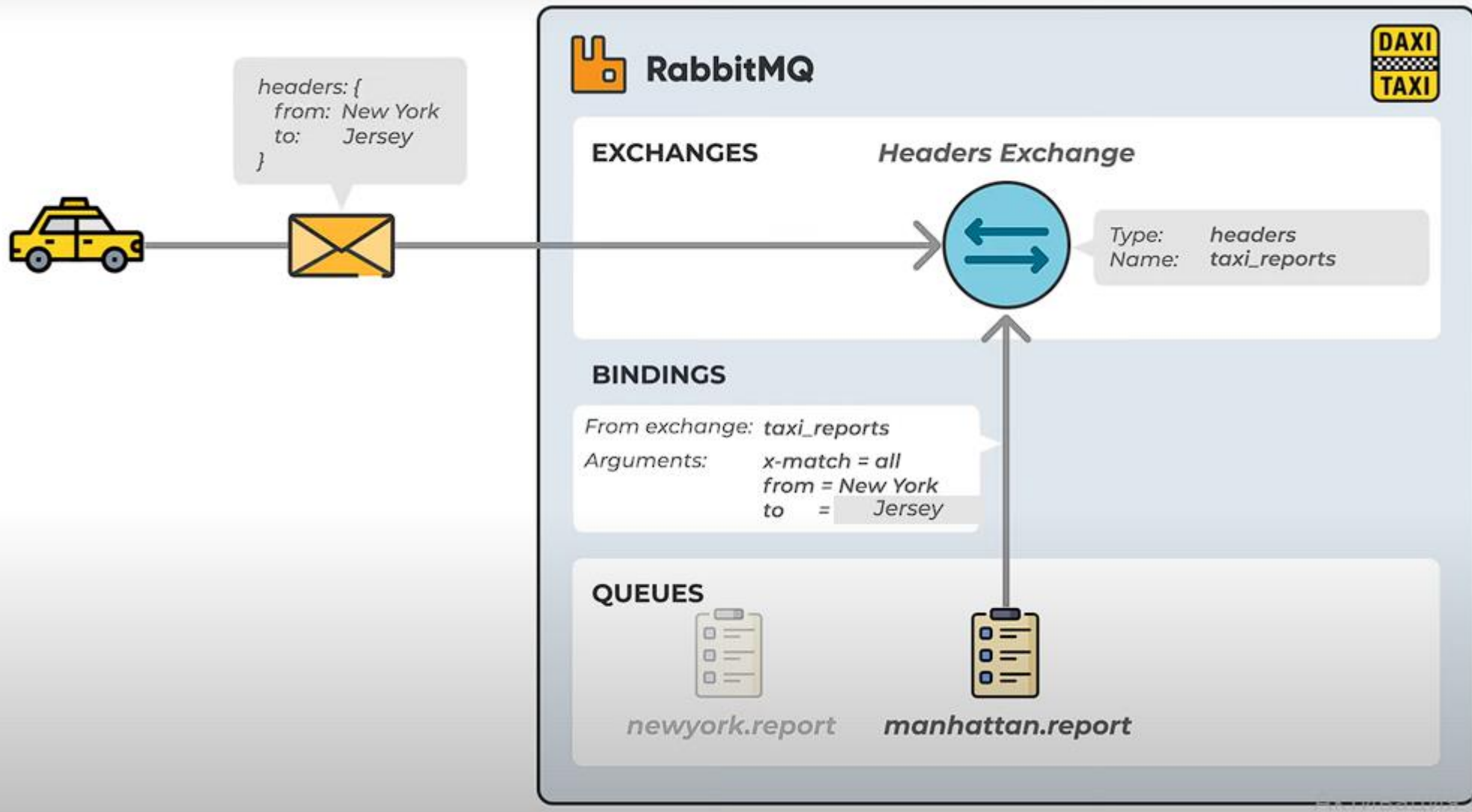
Topic Exchange



Headers Exchange




Headers Exchange



Панель администрирования

← ↻ 🌐 localhost:15672 RabbitMQ Management

 3.7.8 Erlang 22.0.1

Refreshed 2019-05-25 19:00:49 Refresh every 5 seconds

Virtual host All

Cluster rabbit@fixlix-VirtualBox


User admin Log out

OverviewConnectionsChannelsExchangesQueuesAdmin

Overview


Totals

Queued messages last minute ?



Ready 13
Unacked 0
Total 13

Message rates last minute ?



Publish 0.00/s
Publisher confirm 0.00/s
Deliver (manual ack) 0.00/s

Deliver (auto ack) 0.00/s
Consumer ack 0.00/s
Redelivered 0.00/s

Get (manual ack) 0.00/s
Get (auto ack) 0.00/s
Return 0.00/s

Disk read 0.00/s
Disk write 0.00/s

Global counts ?

Connections: 0Channels: 0Exchanges: 7Queues: 2Consumers: 0

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats	+/-
rabbit@fixlix-VirtualBox	36 65536 available	0 58890 available	392 1048576 available	73MB 1.5GB high watermark	16GB 48MB low watermark	4h 24m	basic disc 1 rss	This node All nodes	

Ports and contexts

Export definitions

Import definitions

HTTP APIServer DocsTutorialsCommunity SupportCommunity SlackCommercial SupportPluginsGitHubChangelog

Настройки количества неподтвержденных сообщений в очереди

PrefetchSize – объем (в байтах) сообщений

PrefetchCount – количество сообщений

Global – применяется ли ко всем консьюмерам или только к данному

Настройки очереди

```
/// <summary>
/// Declares a queue. See the <a href="https://www.rabbitmq.com/queues.html">Queues guide</a> to learn more.
/// </summary>
/// <param name="queue">The name of the queue. Pass an empty string to make the server generate a name.</param>
/// <param name="durable">Should this queue will survive a broker restart?</param>
/// <param name="exclusive">Should this queue use be limited to its declaring connection? Such a queue will be deleted when its declaring connection closes.
/// <param name="autoDelete">Should this queue be auto-deleted when its last consumer (if any) unsubscribes?</param>
/// <param name="arguments">Optional; additional queue arguments, e.g. "x-queue-type"</param>
[AmqpMethodDoNotImplement(namespaceName: null)]
❖ IL code
QueueDeclareOk QueueDeclare(string queue, bool durable, bool exclusive,
    bool autoDelete, IDictionary<string, object> arguments);
```

Durable – должна ли очередь сохраниться после перезапуска брокера

Exclusive – должна ли очередь удалиться после закрытия коннекта

Autodelete – должна ли очередь удалиться после отписки от нее консьюмера

Настройка контейнера с Rabbit

https://hub.docker.com/_/rabbitmq

1. Добавить в docker-compose в блок services:

rabbit:

image: rabbitmq:3-management

restart: always

hostname: rabbitmqhost

environment:

RABBITMQ_DEFAULT_USER:

guest

RABBITMQ_DEFAULT_PASS:

guest

volumes:

- rabbitmq_data:/var/lib/rabbitmq

ports:

- "5672:5672"

- "15672:15672"

2. Добавить в docker-compose в блок volumes:

rabbitmq_data:

3. Добавить в данные подключения в appsettings.json, например:

```
"RmqSettings": {  
  "Host": "localhost",  
  "VHost": "/",  
  "Login": "guest",  
  "Password": "guest"  
},
```

4. Считать данные из пункта 3 при инициализации клиентской библиотеки

The image features a high-angle, aerial view of a dense urban skyline, likely New York City, with numerous skyscrapers and buildings. The entire image is overlaid with a semi-transparent blue layer. A network of thin, light blue lines connects various points across the blue area, creating a web-like pattern that suggests connectivity or a transit system. The word "Masstransit" is centered in the middle of the image in a large, white, sans-serif font.

Masstransit

Masstransit как удобная надстройка для RabbitMQ

nuget:

MassTransit

MassTransit.RabbitMQ

<https://masstransit-project.com/getting-started/>



Поддерживает RabbitMQ, ActiveMQ, Azure Service Bus и т.д.

Masstransit как удобная надстройка для RabbitMQ

Что нужно знать чтобы отправить сообщение с помощью RabbitMQ.Client:

- Наименование обменника
- Маршрутный ключ

Что нужно знать чтобы отправить сообщение с помощью Masstransit:

- Имя очереди
- Тип сообщения

Способы передачи сообщений в Masstransit

Три способа передачи сообщений

1. Publish – асинхронное, его получает тот, кто подписан на данный **тип сообщений**
2. Send – асинхронное, его получает только тот, кто подписан на очередь с **данным наименованием**

Механизм повторной обработки (Retry) сообщений в Masstransit

```
e.UseMessageRetry(r =>
{
    r.Handle<ArgumentNullException>();
    r.Ignore(typeof(InvalidOperationException), typeof(InvalidCastException));
    r.Ignore<ArgumentException>(t => t.ParamName == "orderTotal");
});
```

События, не обработанные
повторно, попадают в
Fault - очереди

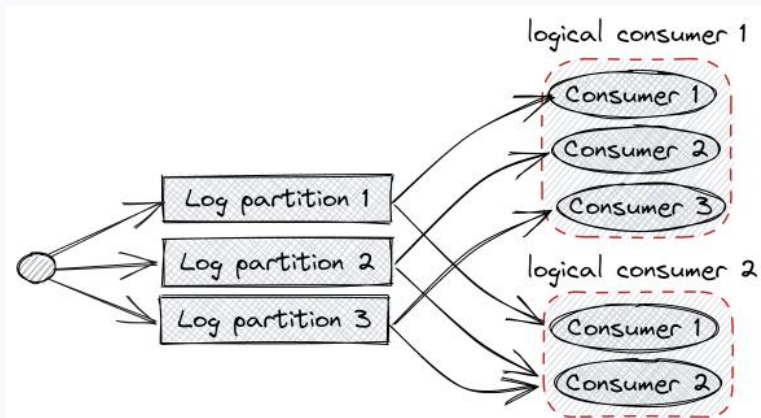
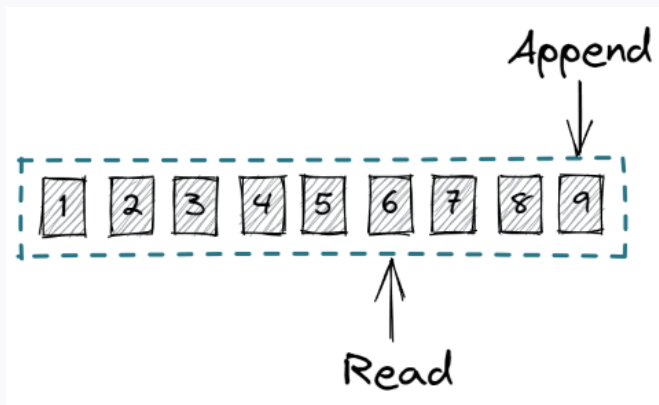
Policy	Description
None	No retry
Immediate	Retry immediately, up to the retry limit
Interval	Retry after a fixed delay, up to the retry limit
Intervals	Retry after a delay, for each interval specified
Exponential	Retry after an exponentially increasing delay, up to the retry limit
Incremental	Retry after a steadily increasing delay, up to the retry limit

The image features a high-angle, aerial view of a dense urban skyline, likely New York City, with numerous skyscrapers and buildings. The entire image is overlaid with a semi-transparent blue filter. A network of thin, light blue lines connects various points across the image, creating a web-like pattern that suggests connectivity and data flow. The word "Kafka" is centered in the middle of the image in a large, white, sans-serif font.

Kafka

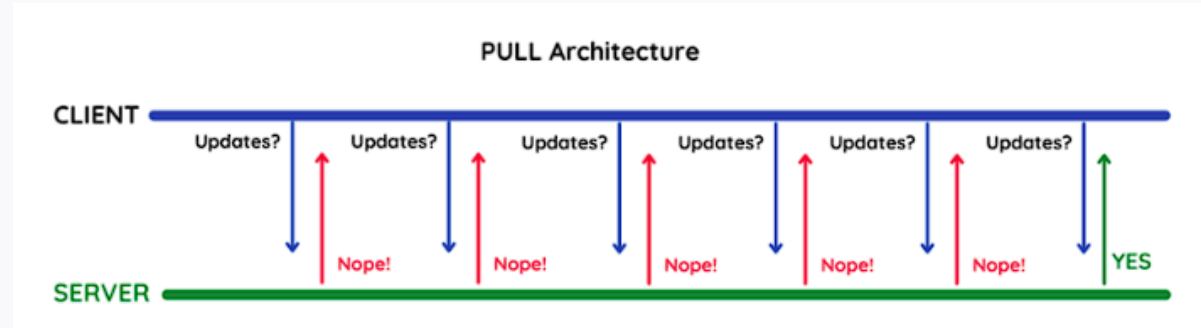
Основа

Kafka реализована на логах



Kafka PULL - модель

Почему выбрана PULL-модель



Механизм подтверждения получения (ask) подразумевает наличие статусов сообщения и их обработку, а это накладные расходы

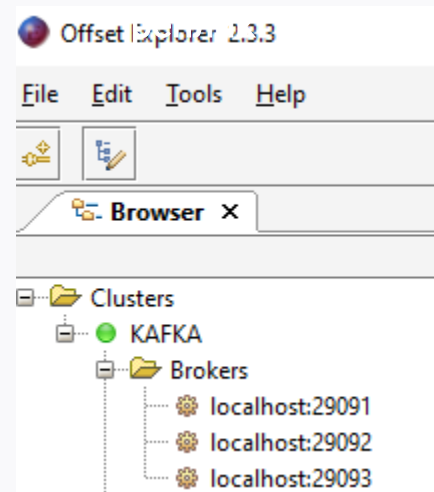
Недостаток PULL в том что потребитель вынужден постоянно запрашивать сообщения компенсируется наличием периода ожидания сообщения.

Инструменты работы с кафкой

1. Command line interface <https://docs.confluent.io/kafka/operations-tools/kafka-tools.html>
<https://docs.confluent.io/platform/current/installation/configuration/index.html>

```
[appuser@145d64181686 ~]$ kafka-topics --bootstrap-server localhost:9092 --topic order_events_3 --describe
Topic: order_events_3   TopicId: tv_-cebWTVwXVRt-roBA4g PartitionCount: 3      ReplicationFactor: 3    Configs:
Topic: order_events_3   Partition: 0    Leader: 1      Replicas: 1,2,3 Isr: 1,2,3
Topic: order_events_3   Partition: 1    Leader: 2      Replicas: 2,3,1 Isr: 2,3,1
Topic: order_events_3   Partition: 2    Leader: 3      Replicas: 3,1,2 Isr: 3,1,2
```

2. Клиентские приложения, например
Offset explorer <https://www.kafkatool.com/>
Kafka-ui <https://docs.kafka-ui.provectus.io/>



3. Клиентская библиотека Confluent-Kafka client
<https://github.com/confluentinc/confluent-kafka-dotnet>

Установка

Ссылка на образ <https://hub.docker.com/r/confluentinc/cp-kafka>

```
zookeeper:
  container_name: zookeeper
  image: confluentinc/cp-zookeeper:7.3.2
  ports:
    - "2181:2181"
  environment:
    ZOOKEEPER_SERVER_ID: '1'
    ZOOKEEPER_SERVERS: 'zookeeper:2888:3888'
    ZOOKEEPER_CLIENT_PORT: '2181'
    ZOOKEEPER_PEER_PORT: '2888'
    ZOOKEEPER_LEADER_PORT: '3888'
    ZOOKEEPER_INIT_LIMIT: '10'
    ZOOKEEPER_SYNC_LIMIT: '5'
    ZOOKEEPER_MAX_CLIENT_CONNS: '5'
```

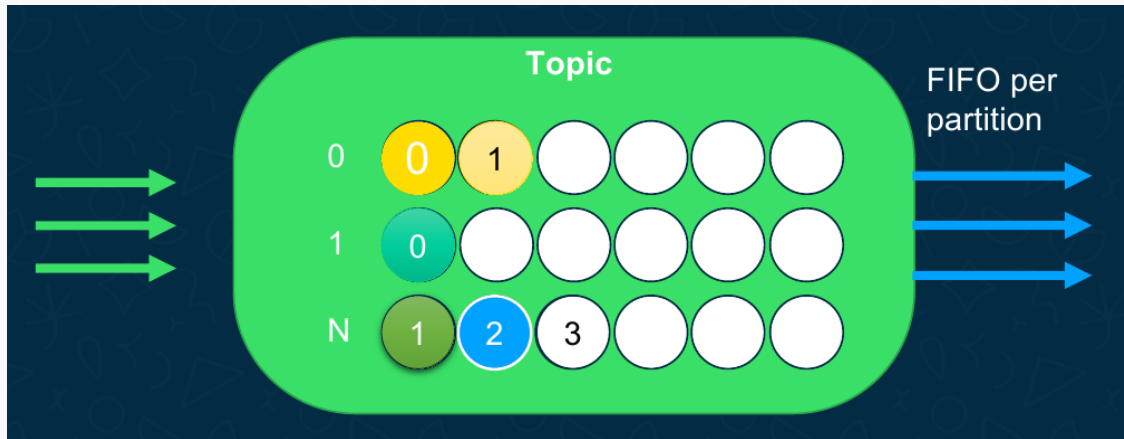
```
kafka-broker-1:
  image: confluentinc/cp-kafka:7.3.2
  container_name: kafka-broker-1
  depends_on:
    - zookeeper
  ports:
    - "29091:29091"
  environment:
    KAFKA_BROKER_ID: '1'
    KAFKA_BROKER_RACK: '1'
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_LISTENERS: INSIDE://0.0.0.0:9091,OUTSIDE://0.0.0.0:29091
    KAFKA_ADVERTISED_LISTENERS: INSIDE://kafka-broker-1:9091,OUTSIDE://localhost:29091
    KAFKA_INTER_BROKER_LISTENER_NAME: INSIDE
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INSIDE:PLAINTEXT,OUTSIDE:PLAINTEXT
    #KAFKA_CREATE_TOPICS: "my-topic:1:1"
    KAFKA_DEFAULT_REPLICATION_FACTOR: '2'
  volumes:
    - kafka_broker_1_data:/var/lib/kafka/data
    - kafka_broker_1_logs:/var/lib/kafka/logs
```

Данные для подключения

Перечисленные через запятую адреса брокеров

```
"kafkaOptions": {  
  "BootstrapServers" : "localhost:29091,localhost:29092,localhost:29093"  
}
```


Топик



Топик - именованный контейнер, содержащий похожие сообщения

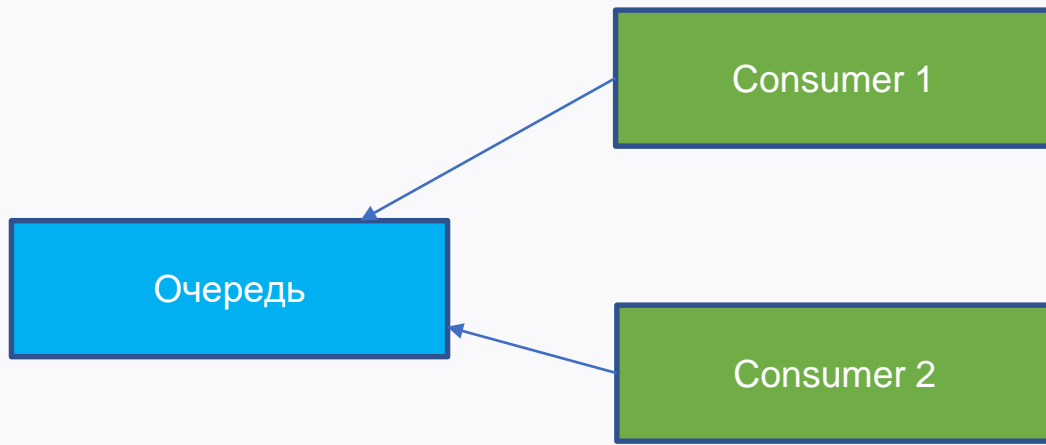
Содержит сообщения в виде логов, каждый отдельный лог называется партицией

Topic является логическим понятием, физически сообщения располагаются в партициях

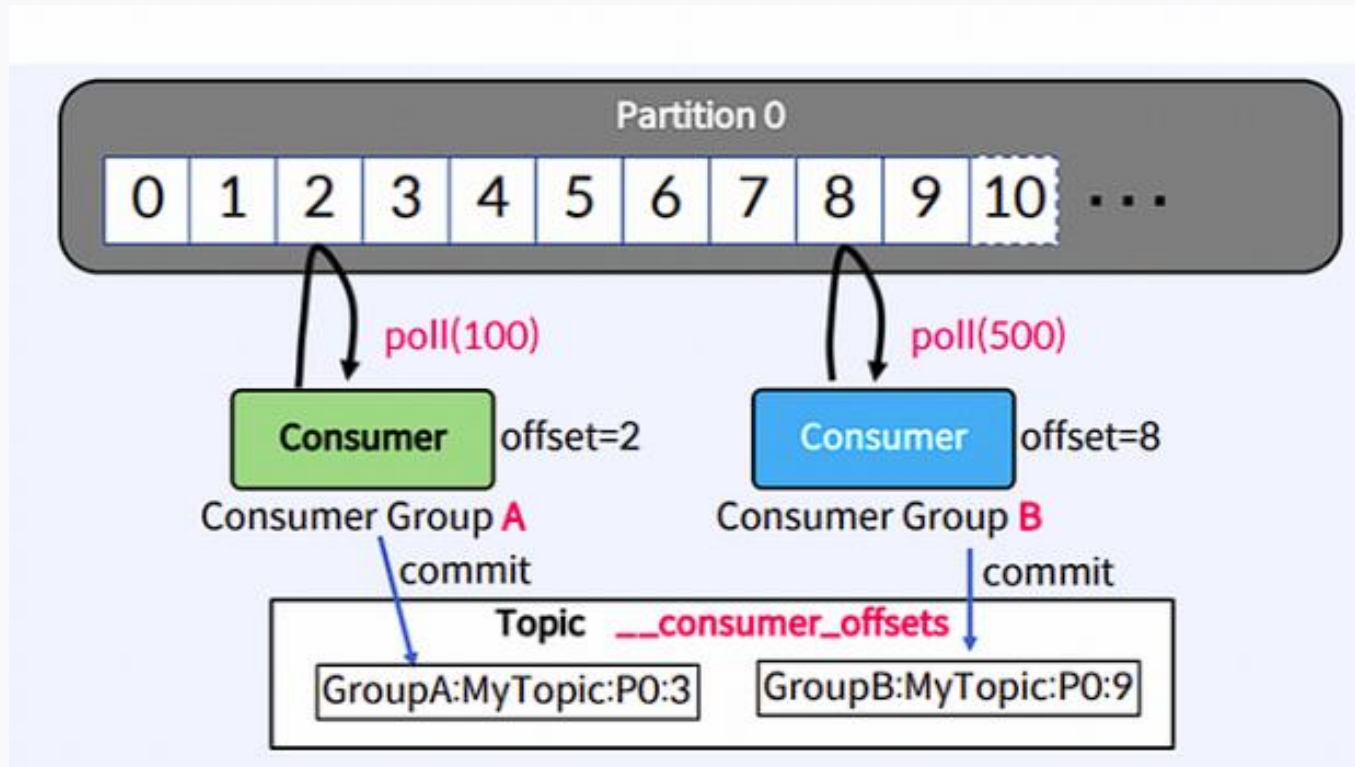
Топик заказов, топик отправок, топик уведомлений и т.п.

Чтение сообщений из очереди

Как консьюмер понимает, с какого сообщения начать чтение?

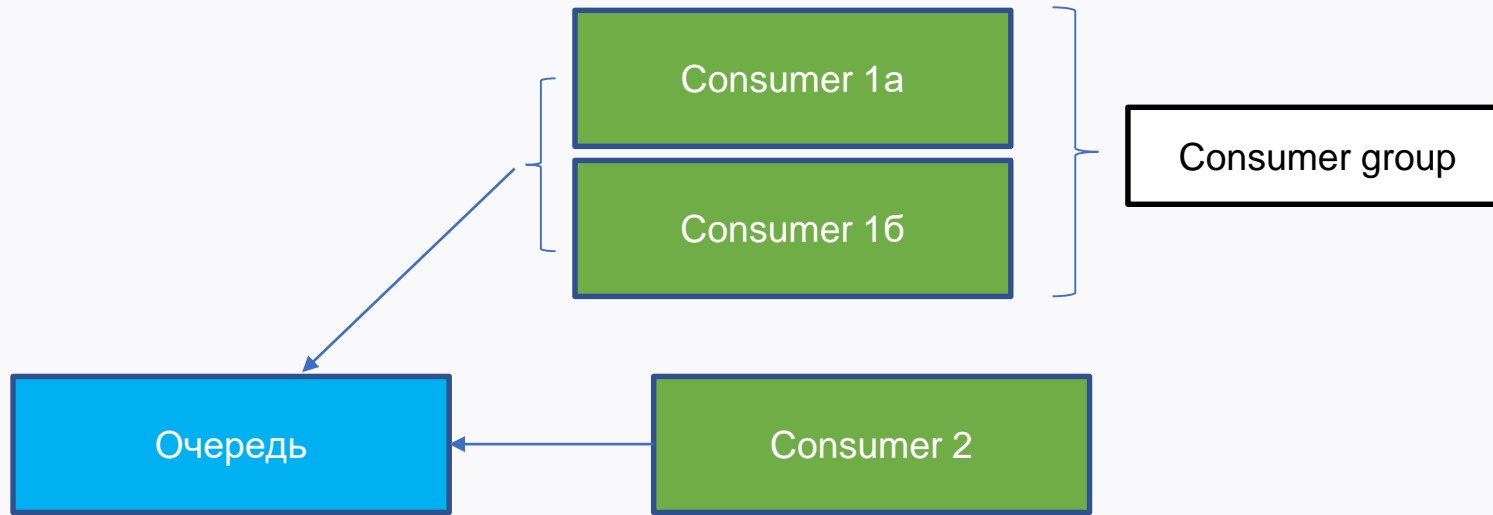


Offset



Offset показывает, с какой позиции данный Consumer Group должен читать сообщения

Consumer group



Консьюмеры с единым Consumer group действуют как реплики одного консьюмера

Стратегии партицирования с ключом

1. Round robin

Например, партии 1 и 2. Каждое приходящее сообщение распределяется по очереди в эти партии, т.е. 1,2,1,2,1,2...

2. Явное определение партии

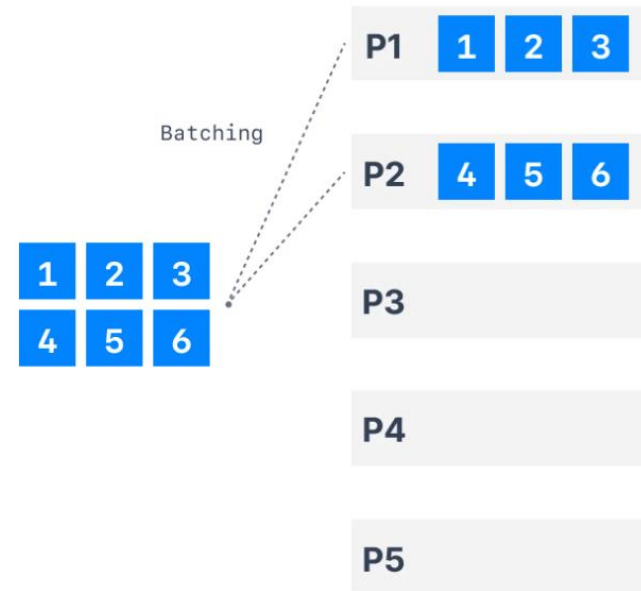
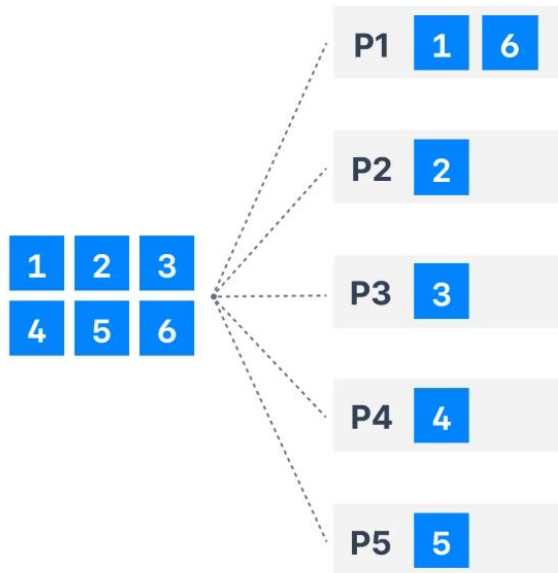
Например, имеется 100 отделов, сообщения нужно распределить по ним. Принимают стратегию если номер отдела < 50 , отправляют в партию 1, иначе – в 2

3. Key-defined ($\text{key-hash} \% n$) (по умолчанию) – вычисляется хэш ключа партия вычисляется как $\text{key-hash} \% n$. Стратегия подразделяется далее по виду алгоритма хэширования

Стратегии партицирования без ключа

Если сообщение не содержит ключа,

1. До 2.4 отправлялось через Round robin
2. Начиная с 2.4 отправляется также Round robin, но батчем - называется «sticky partitioner» (целевая партиция не меняется пока не наполнится батч или не исчерпается время `linger.ms`)



Kafka



The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent blue band that contains a white network diagram. This diagram consists of numerous small dots connected by thin white lines, creating a web-like structure that spans the width of the slide. Centered within this band is the word "Практика" in a large, white, sans-serif font.


Практика

The background of the entire image is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. A semi-transparent blue overlay covers the entire image. In the center, there is a network of white lines connecting various points, creating a web-like pattern. The text "Ответы на вопросы" is written in white, bold, sans-serif font across the middle of the image.

Ответы на вопросы

Список материалов для изучения

1. Об устройстве RMQ <https://www.rabbitmq.com/tutorials/amqp-concepts.html>
2. О библиотеке RMQ.Client <https://www.rabbitmq.com/dotnet-api-guide.html>
3. О кафке коротко <https://medium.com/codex/asynchronous-communication-why-doeskafka-use-a-pull-based-message-consumer-442c19a70f58>
4. Короткое но информативное видео о кафке <https://youtu.be/Ch5VhJzaoal?si=sNsRj72itoKdnCr>
5. Более длинное и сложное, но на русском <https://youtu.be/-AZOi3kP9Js?si=b8wylQ4Rtp4RC-Ro>
6. Библиотека Kafka.Confluent, примеры реализации. См.ReadMe снизу <https://github.com/confluentinc/confluent-kafka-dotnet>

The background of the image is an aerial photograph of a city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that features a white geometric network pattern of interconnected dots and lines. The text is centered within this blue layer.

Заполните, пожалуйста,
опрос о занятии по ссылке в чате