**Системная и программная инженерия**

Лабораторная работа №4 «Знакомство с RabbitMQ»

**Введение**

**RabbitMQ** – это брокер сообщений с открытым исходным кодом, который маршрутизирует собщения, отправленные приложением-продюсером (**producer**) согласно принципам протокола AMQP (Advanced Message Queuing Protocol). RabbitMQ не просто принимает сообщение от продюсера, а доставляет его приложению-потребителю (**consumer**). Более того, RabbitMQ работает по модели «умный сервер, тупой клиент», позволяя реализовать сложную логику маршрутизации сообщения благодаря нескольким типам обменников (***exchanger***), которые отправляют сообщения в очереди (**queue**).

**Базовые определения**

* AMQP (Advanced Message Queuing Protocol) — открытый протокол для передачи сообщений между компонентами системы. Основная идея состоит в том, что отдельные подсистемы (или независимые приложения) могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет маршрутизацию, возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений.
* Producer (Поставщик) – Источник, посылающий сообщение в очередь RabbitMQ
* Consumer (Подписчик) – Программа, принимающая сообщения. Такие программы подписываются на события поступления сообщения в очередь, и всегда находятся в ожидании новых сообщений.
* exchange (обменник или точка обмена) — в неё отправляются сообщения. Обменник **распределяет сообщение** в одну или несколько очередей. Он **маршрутизирует сообщения в очередь** на основе созданных связей (binding) между ним и очередью
* queue (очередь) — структура данных на диске или в оперативной памяти, которая **хранит ссылки на сообщения и отдает копии сообщений consumers** (потребителям). Одна очередь может использоваться несколькими потребителями
* binding (привязка) — правило, которое **сообщает точке обмена в какую из очередей эти сообщения должны попадать**. Обменник и очередь могут быть связаны несколькими привязками

**Алгоритм работы**

1. Приложение-продюсер отправляет сообщение конкретному обменнику;
2. Обменник маршрутизирует полученное сообщение его в одну или несколько очередей, в зависимости от типа самого обменника, правилам его привязки к очереди (***binding***) и значению ключа маршрутизации в сообщении (***routing*** ***key***);
3. Очередь хранит ссылку на полученное сообщение, которое физически может хранится в оперативной памяти или на диске, в зависимости от свойств самой очереди;
4. Когда приложение-потребитель готово получить сообщение из очереди, брокер копирует его по ссылке из очереди и отправляет копию сообщения потребителю;
5. Получив сообщение, приложение-потребитель отправляет брокеру подтверждение об успешном получении данных;
6. После получения подтверждения брокер удаляет копию сообщения из очереди;
7. Наконец, брокер удаляет само исходное сообщение из очереди, очищая место в оперативной памяти или на жестком диске.

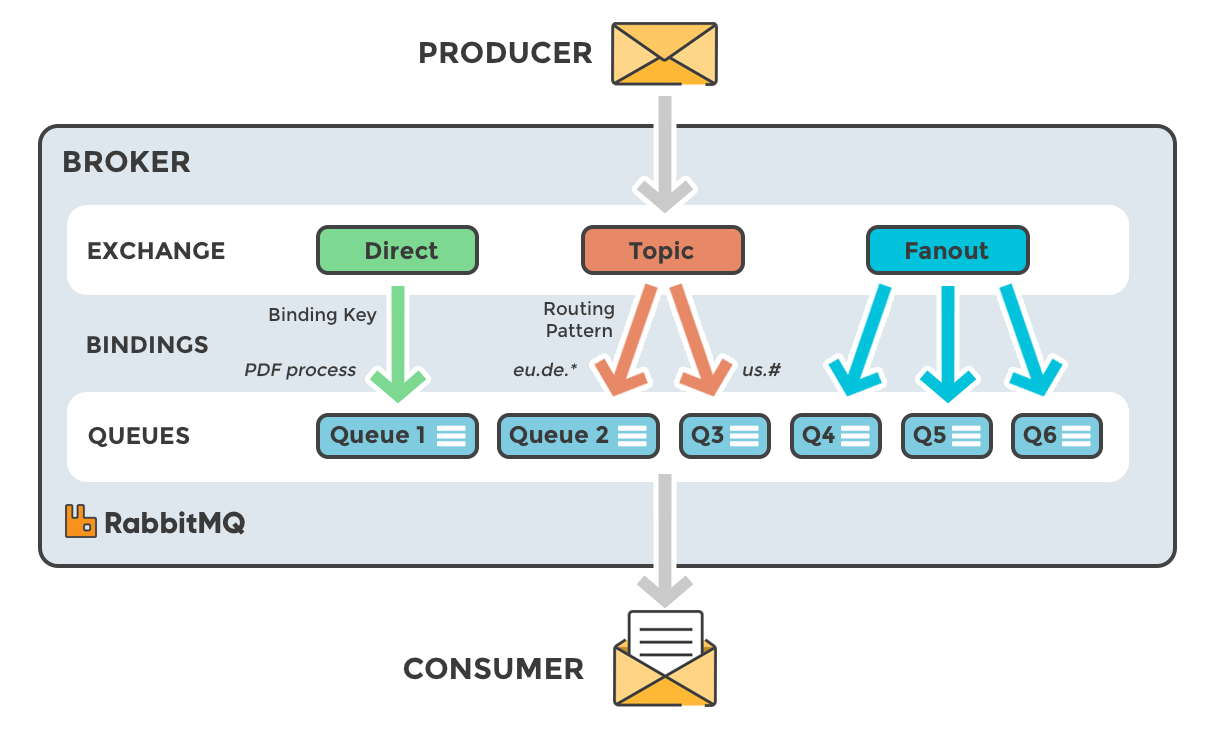


Рис. 1 – Схема работы RabbitMQ

**Виды обменников (Exchange)**

**Fanout Exchange.** Это «маршрутизация без маршрутизации» — то есть мы отправляем сообщения во все очереди.

Особенности:

* RabbitMQ **не работает с ключами маршрутизации и шаблонами** что положительно влияет на производительность. Это самый быстрый exchange;
* все потребители должны иметь возможность обрабатывать все сообщения;

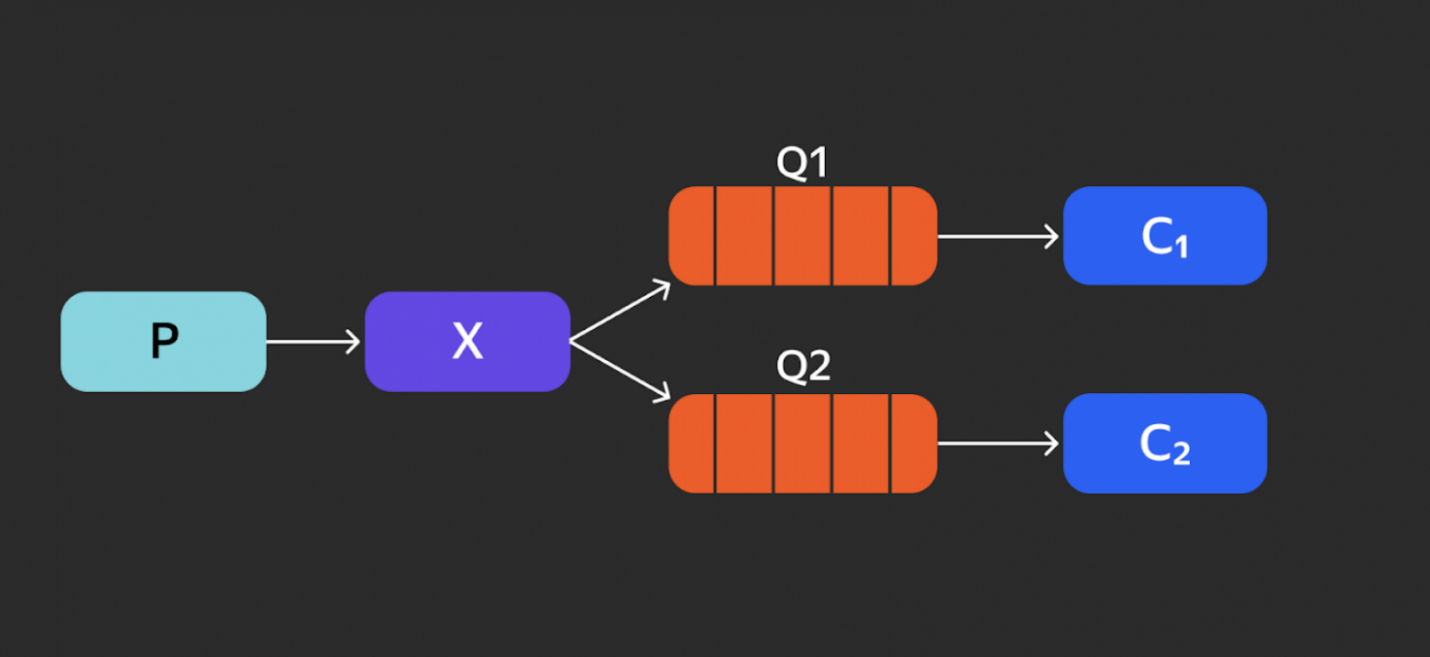


Рис. 2 – Схема работы Fanout exchange

**Direct Exchange.** Это маршрутизация по ключу. В RabbitMQ существует routing-key. Ключ маршрутизации — это категория сообщения. В данном случае orange, black и green. Они распределяются по разным очередям в зависимости от ключа. **Ключ маршрутизации — это строка**. Поиск соответствия происходит при помощи **проверки строк на эквивалентность**.

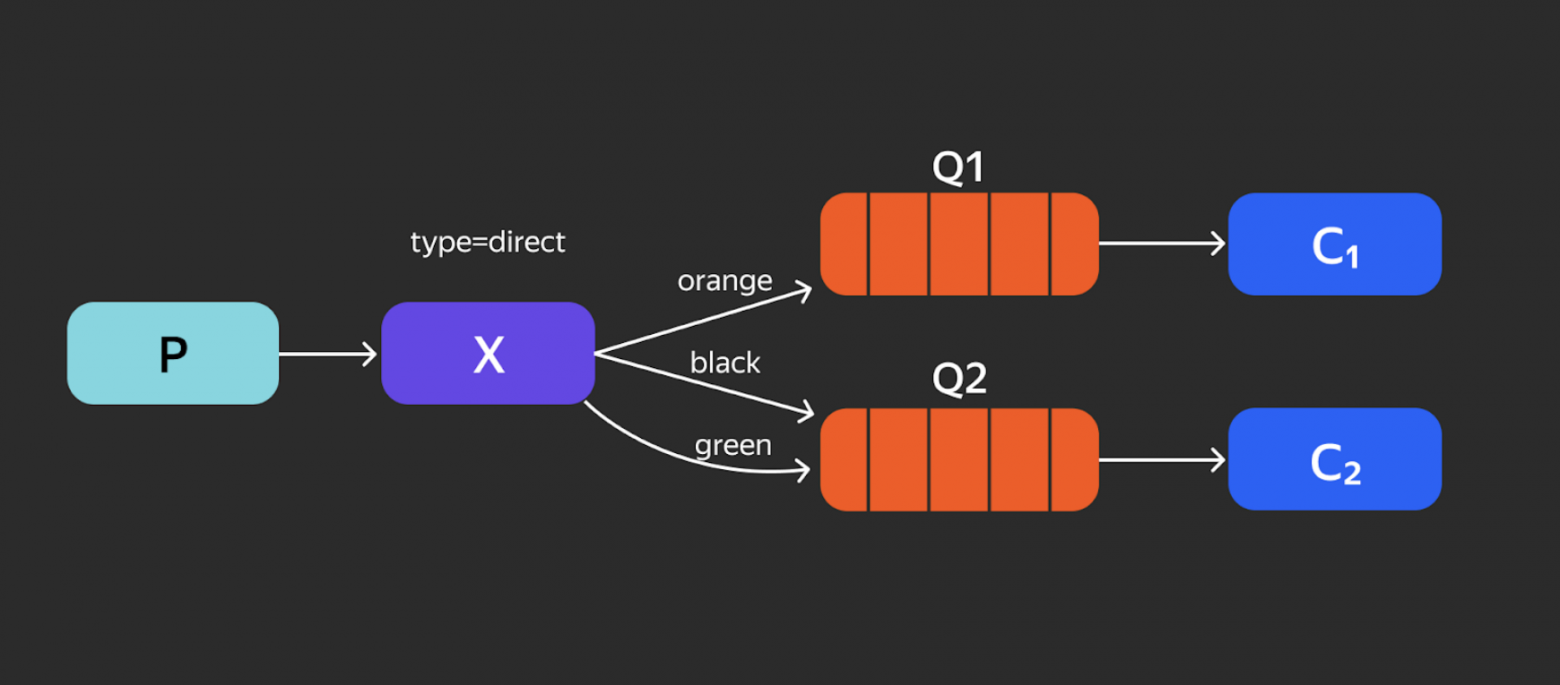


Рис. 3 – Схема работы Direct exchange

**Total Exchange.** Это уже более сложный Binding. Сообщения, не попадающие под ключ, удаляются, так как никому не нужны. То, что остаётся, передаётся в сами очереди.

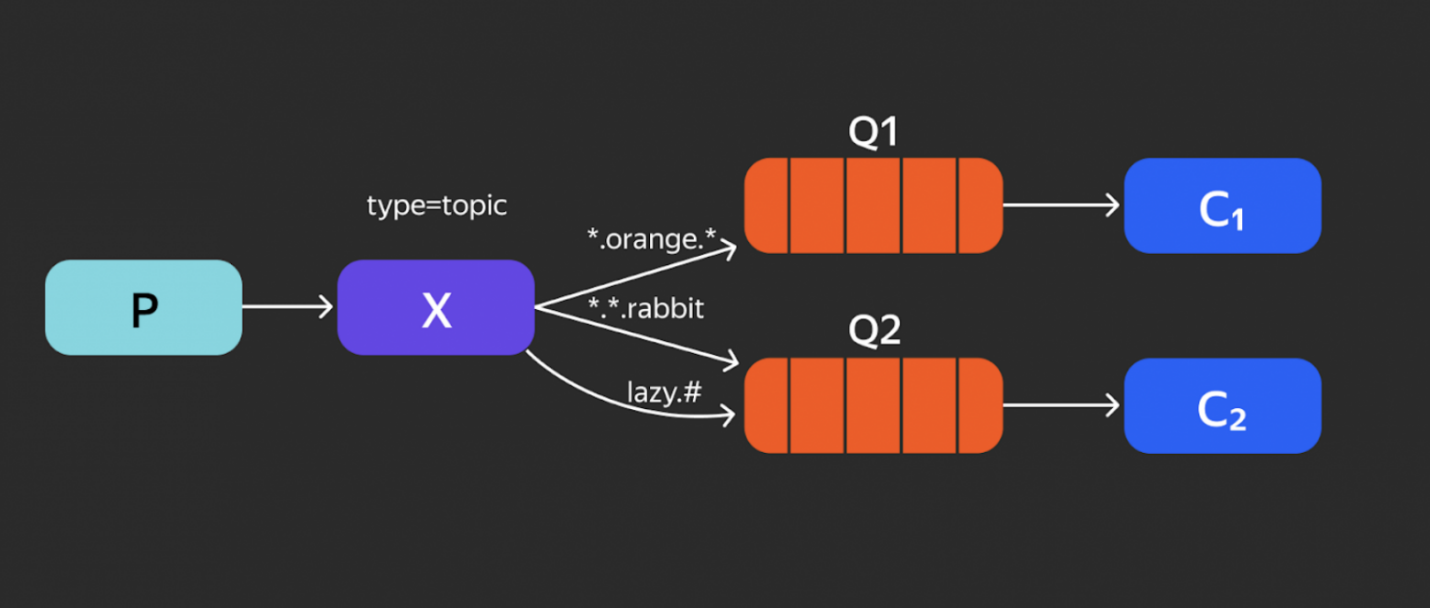


Рис. 4 – Схема работы Total exchange

**Плюсы RabbitMQ**

* Лёгкость разработки. У RabbitMQ есть библиотеки клиента для большинства современных языков и открытый исходный код, чтобы в нём разбираться.
* Простое администрирование. У RabbitMQ удобная админка, где вы можете в режиме реального времени разбираться с тем, что происходит. Роутинги можно настраивать в процессе, переключая нагрузку и меняя правила обработки.
* Тонкая настройка. Многие параметры можно менять, чтобы подстроить систему под свои нужды. Особенно это касается очередей.

**Минусы RabbitMQ**

* Работа при высокой нагрузке. У RabbitMQ есть сложности при горизонтальном масштабировании в кластере. Приходится добавлять настройки кластеризации над очередями, а они сложные и работают плохо. Приходится выбирать, чем пожертвовать — гибкостью или скоростью.

### В каких случаях использовать RabbitMQ

* Когда важна гибкость маршрутизации сообщений внутри системы. В таком случае он предоставляет инструменты для построения путей доставки данных и способен решить самые хитрые сценарии в организации потоков событий.
* Если вам важен сам факт доставки сообщений и порядок их получения.

**Практическая часть**

**Установка**

В данной лабораторной работе для удобства развертывания RabbitMQ будет использоваться Docker. Предварительно необходимо скачать образ RabbitMQ (можно не использовать средства контейнеризации Docker, а установить RabbitMQ непосредственно на вашу рабочую машину)

Запустим контейнер с образом RabbitMQ командой:

docker run --rm -p 15672:15672 rabbitmq:3.10.7-management

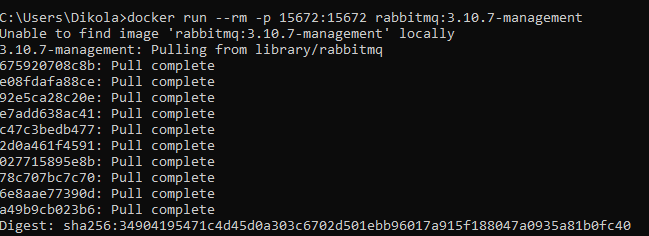


Рис. 4 – Схема работы Direct exchange

После этого можно открыть веб-интерфейс RabbitMQ в браузере по ссылке <http://127.0.0.1:15672/> Далее мы перейдем на страницу авторизации, войдем с систему используя стандартный логин guest и пароль guest.

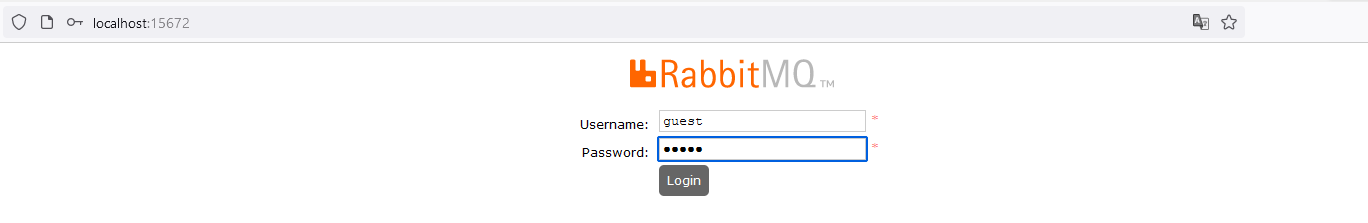


Рис. 4 – Страница авторизации RabbitMQ

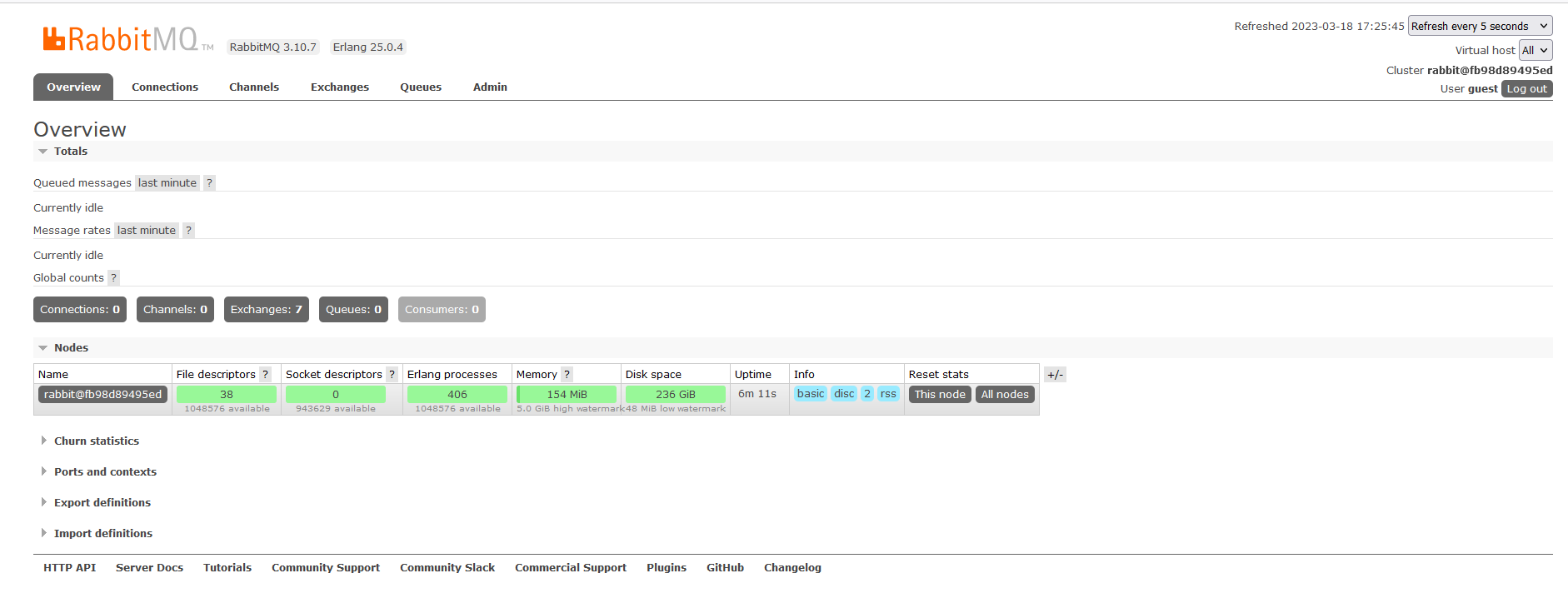


Рис. 5 – Главная страница RabbitMQ

**Создание** **Fanout обменника**

Далее мы реализуем Fanout обменник, а также создадим для него очереди, и отправим наше первое сообщение. Для начала нам необходимо перейти на вкладку Exchanges.

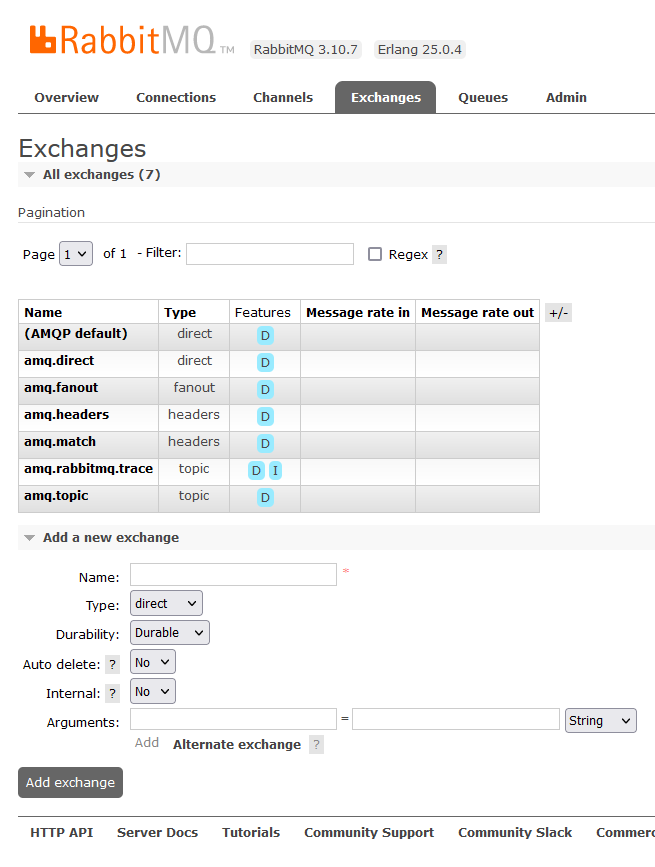


Рис. 6 – Вкладка Exchanges

На данной вкладке отображены все созданные обменники, а также указан их тип, и статистика по входящим и исходящим сообщениям. Так же ниже таблицы приведена вкладка для создания нового обменника. Для его создания укажем ему имя, а также зададим тип «Fanout».

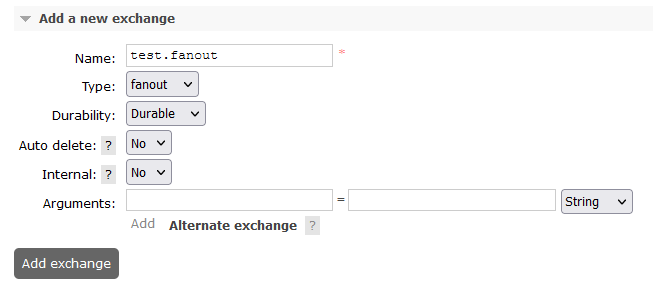


Рис. 7 – Создание fanout обменника

Далее нажимаем кнопку «Add exchange», и видим, как в таблице появился созданный нами обменник.

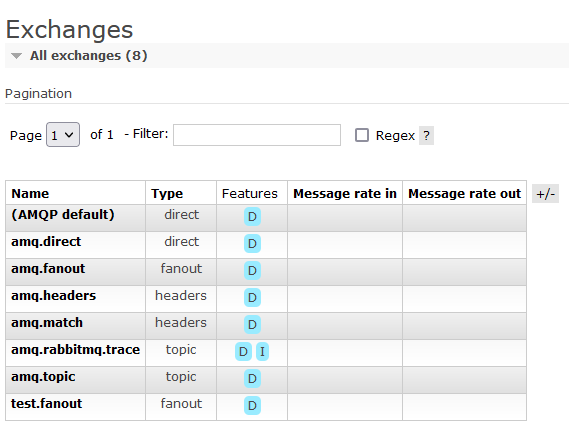


Рис. 8 – Создание fanout обменника

Далее нам необходимо создать очереди для нашего обменника, и связать их с ним. Для этого перейдем на вкладку «Queues». На ней будут отображены все наши существующие очереди.

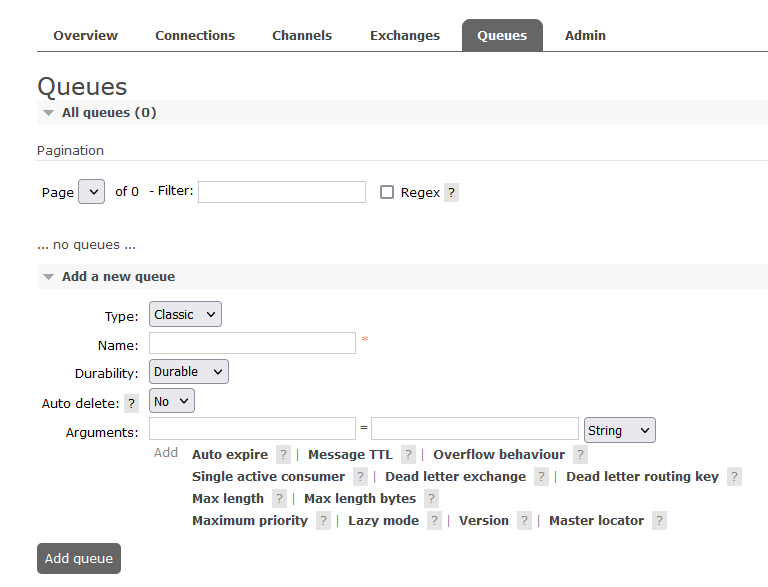


Рис. 9 – Вкладка Queues

На данный момент у нас не существует ни одной очереди, поэтому по аналогии с обменником нам необходимо её создать на вкладке «Add a new queue». Зададим имя для очереди, и создадим её.

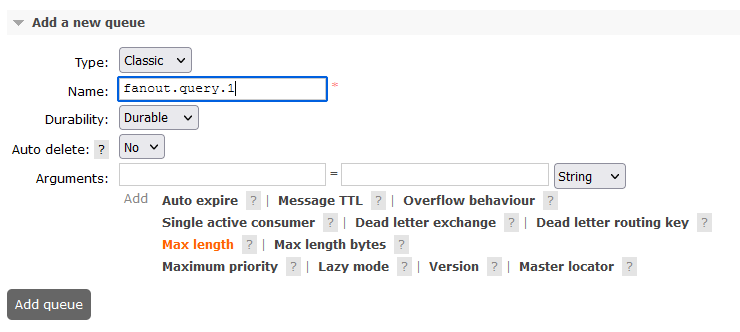


Рис. 10 – Создание очереди

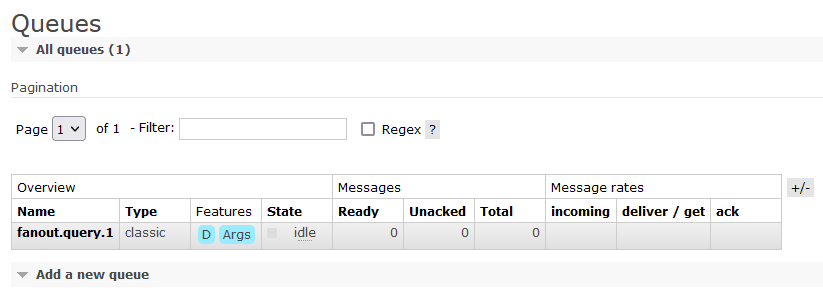


Рис. 11 – Вкладка Queues после создания новой очереди

После создания она отобразится в таблице очередей. Далее для того, чтобы связать её с обменником нам необходимо перейти в интерфейс очереди, кликнув по её названию в таблице очередей.

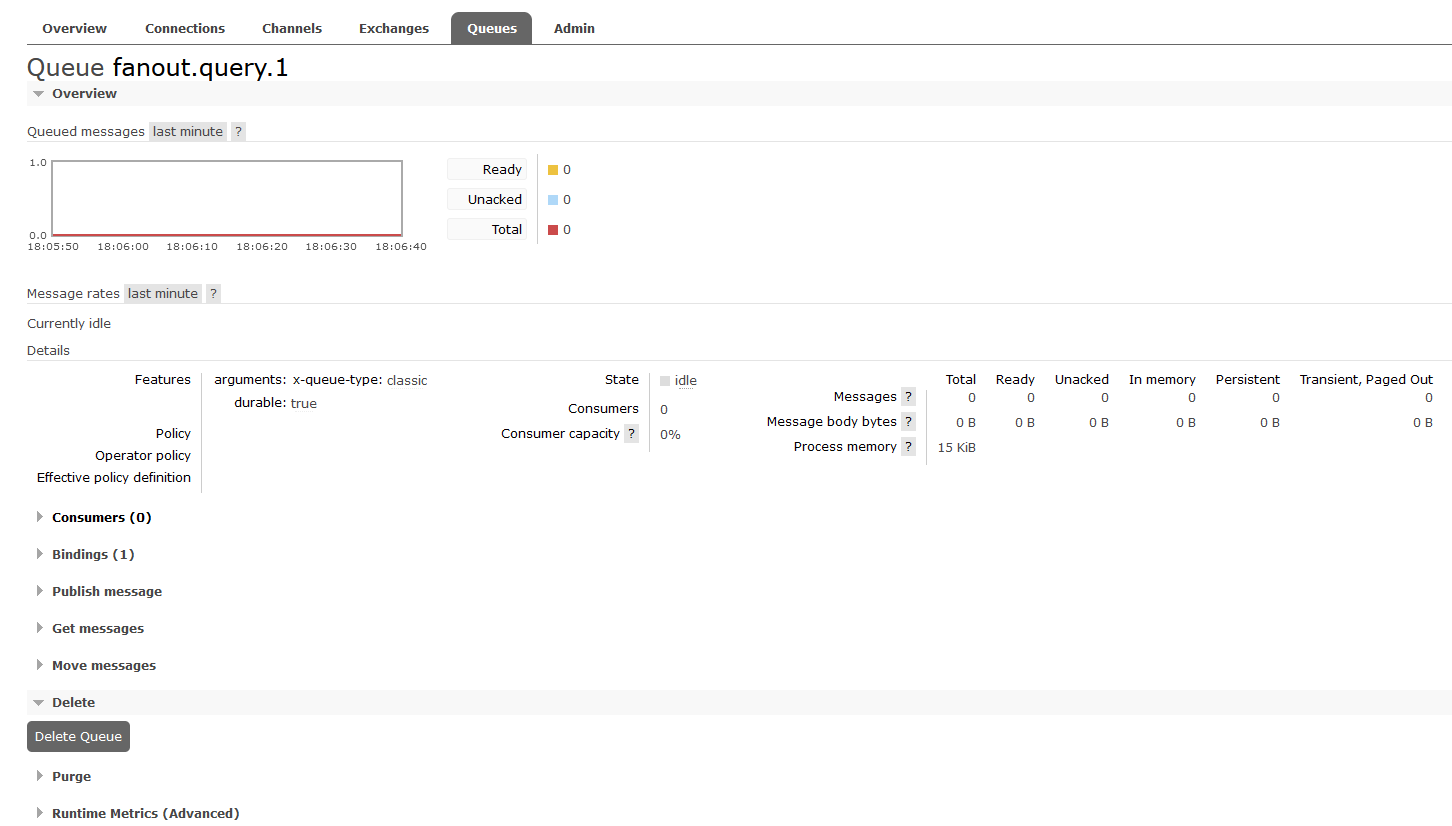


Рис. 12 – Интерфейс очереди fanout.query.1

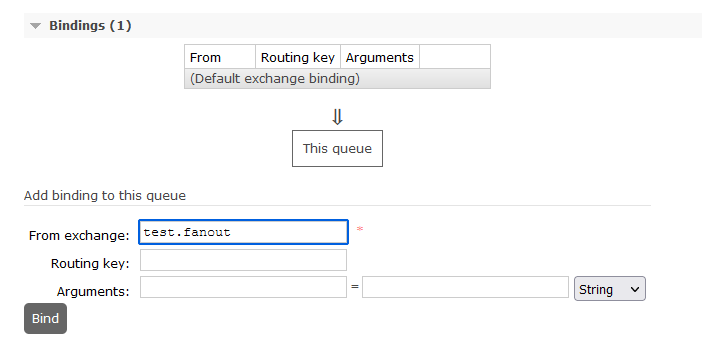
Далее необходимо открыть вкладку «Bindings», и в поле «From exchange» указать название обработчика. В нашем случае мы указываем «test.fanout». Далее мы нажимаем кнопку «Bind». 

Рис. 13 – Привязка очереди к обработчику

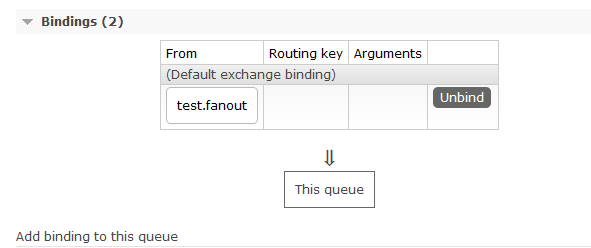
После привязки очереди к обработчику во вкладке «Bindings» отображается таблица, в которой указано к каким обработчикам привязана данная очередь. 

Рис. 14 – Таблица привязки очереди к обработчику

Далее вернемся на вкладку «Exchanges», и перейдем в интерфейс обменника «test.fanout». Во вкладке «Bindings» также отображается таблица, в которой указано к каким очередям привязана данный обработчик.

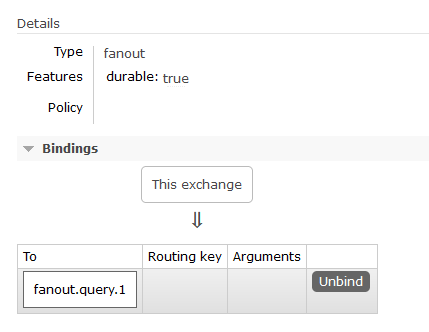
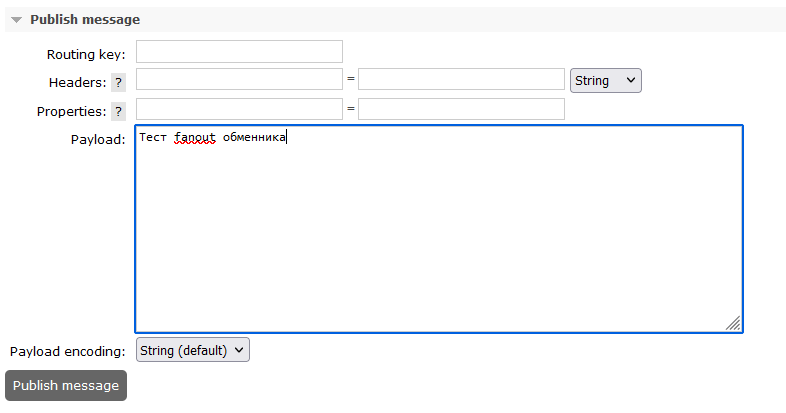


Рис. 15 – Таблица привязки обработчика к очереди

**Отправка сообщения**

Для того, чтобы отправить сообщение мы воспользуемся веб-интерфейсом RabbitMQ. Для этого необходимо в интерфейсе обменника перейти на вкладку «Publish message». Для fanout обменника нам необходимо лишь указать текст сообщения в поле «Payload», и нажать кнопку «Publish message». После чего сообщение будет отправлено в первую из доступных очередей.

 Рис. 16 – Отправка сообщения через fanout обменник.

Для того, чтобы проверить, что сообщение пришло, и для того чтобы прочитать его перейдем в интерфейс привязанного к обменнику очереди, и во вкладке «Get messages» нажать на кнопку «Get message(s)», после чего мы увидим текст сообщения, которое пришло к нам в очередь.

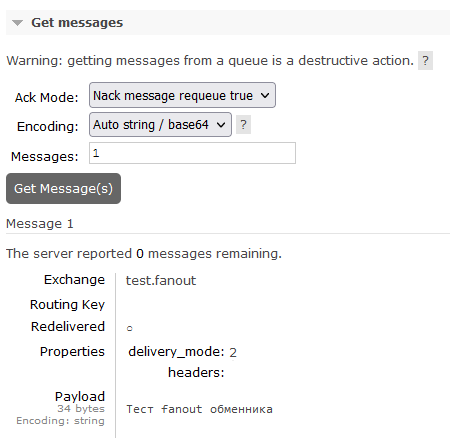


Рис. 17 – Чтение сообщения из очереди

**Создание** **Direct обменника**

Для начала необходимо создать direct обменник. Его создание происходит по аналогии с fanout обменником, однако в данном случае нам необходимо в поле «Type» указать «Direct».

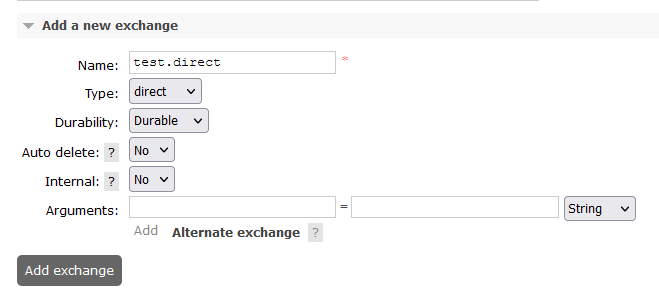


Рис. 18 – Создание direct обменника

Далее нам необходимо создать две очереди: direct.query.1 и direct.query.2 по аналогии с тем, как мы это делали для fanout обменника.

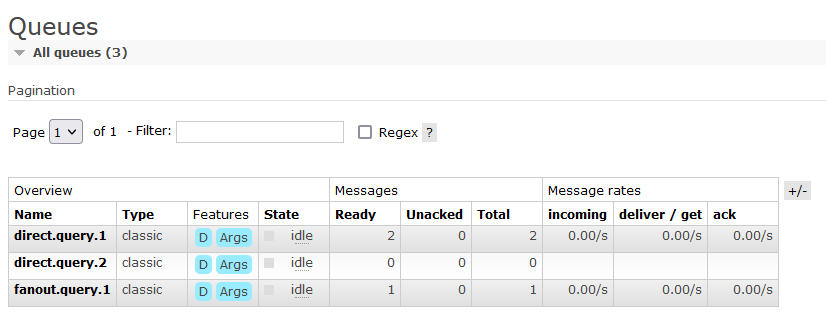


Рис. 19 – Создание двух очередей для direct обменника

Далее нам необходимо связать их с direct обменником, однако в этом случае мы также укажем Routing key для каждой из очередей, по которому сообщения, отправленные с данным параметром, будут отправлены в разные очереди.

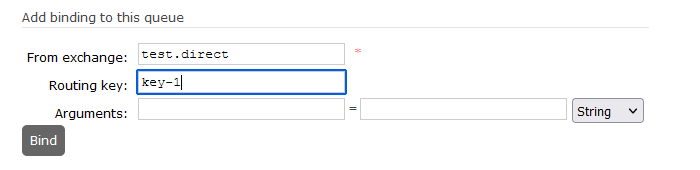


Рис. 19 – Создание связи для очереди direct.query.1

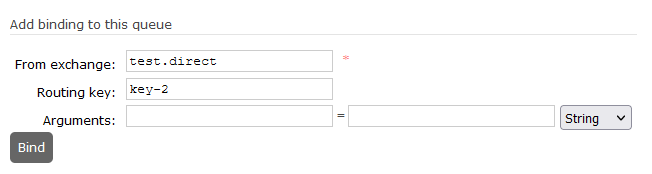


Рис. 20 – Создание связи для очереди direct.query.2

Далее перейдем в интерфейс direct обменника, и отправим два сообщения с ключами key-1 и key-2.

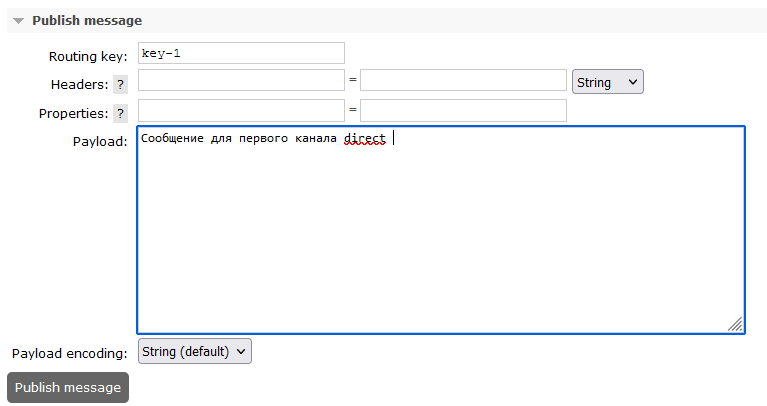


Рис. 21 – Отправка сообщения с ключом key-1

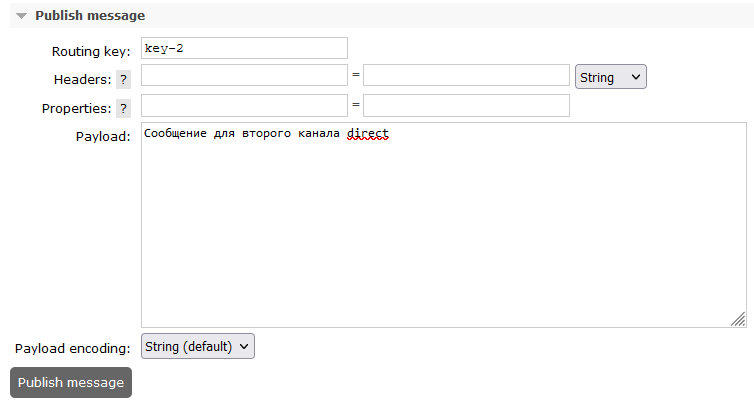


Рис. 22 – Отправка сообщения с ключом key-2

Далее перейдем в интерфейс direct.query.1, и попробуем прочитать входящие сообщения.

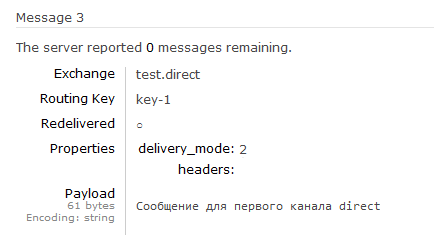


Рис. 23 – Чтение сообщения из direct.query.1

Как мы видим, в этот канал попало только сообщение, у которого был указан ключ key-1. Для того, чтобы убедиться, что второе сообщение попало в канал direcy.query.2, перейдем в его интерфейс, и прочитаем входящие в него сообщения.

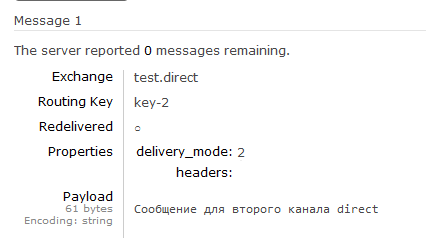


Рис. 24 – Чтение сообщения из direct.query.2

**Практическое задание**

1. Создать fanout обменник, в имени которого необходимо указать ФИО, и номер группы, создать очередь для него, связать их между собой, и отправить сообщение. Продемонстрировать отправку сообщения.
2. Создать direct обменник, в имени которого необходимо указать ФИО, и номер группы, создать две очереди для него, связать очереди с обменником по разным ключам, и отправить сообщения с разными ключами. Продемонстрировать отправку сообщения.

**Теоретические вопросы**

1. Что такое RabbitMQ? Для чего применяется?
2. Назовите и опишите основные элементы, участвующие в схеме работы RabbitMQ.
3. Опишите схему работы RabbitMQ.
4. Назовите и опишите основные виды обменников.
5. Что такое routing key, и для чего он применяется?