Advanced topics

# 1. Messaging systems

## Intro

* Messaging provides a mechanism for loosely-coupled integration of systems
* The central unit of processing in a message is a message which typically contains a **body** and a **header**
* Use cases include:
  + Log aggregation between systems
  + Event propagation between systems - някакви събития се fire-ват
  + Offloading log-running tasks to worker nodes - the result of the task then to be sent to a third systems for example
* Messaging solutions implement different protocols for transferring of messages such as **AMQP** (binary protocol), XMPP, MQTT and many more like XML, JSON, etc.
* The variety of protocols implies vendor lock-in when using a particular messaging solution (also called a messaging broker) - ако е специфичен протокола лошо. Т.е. протокола е добре да е такъв, че да може да се използва от различни message broker systems

* Message brokers
  + ActiveMQ - using JMS (Java Messaging System) Java EE
  + RabbitMQ
  + Qpid
  + TIBCO
  + WebSphere MQ
  + Msmq

* Messaging solutions provide means for:
  + Securing message transfer, authenticating and authorizing messaging endpoints
  + Routing messages between endpoints
  + Subscribing to the broker

* An **enterprise service bus (ESB)** is one layer of abstraction above a messaging solution that further provides:
  + Adapters for different messaging protocols
  + Translation of messages between the different types of protocols

## I. RabbitMQ

### Info

* An open source message broker written in Erlang
  + Заема малко ранм памет и процесор
  + при Erlang няма context switching като при JVM
  + reliability - дете ако не се изпълни, то родителят му го пуска за изпълнение наново
* **Implements the AMQP protocol** (Advanced Message Queueing Protocol)
* Has a pluggable architecture and provides extension for other protocols such as HTTP, STOMP and MQTT

* AMQP is a binary protocol that aims to standardize middleware communication
* The AMQP protocol derives its origins form the financial industry - processing of large volumes of financial data between different systems is a classic use case of messaging
* The AMQP protocol defines:
  + **Exchanges** - the message broker endpoints that receive messages
  + **Queues** - the message broker endpoints that store messages from exchanges and are used by subscribers for retrieval of messages. The Queue can also be persistent - messages can be saved.
  + **Bindings** - rules that bind exchanges and queues
* The AMQP protocol is programmable - which means that the above entities can be created/ modified/ deleted by applications

* The AMQP protocol defines multiple connection channels inside a single TCP connection in order to remove the overhead of opening a large number of TCP connections to the message broker

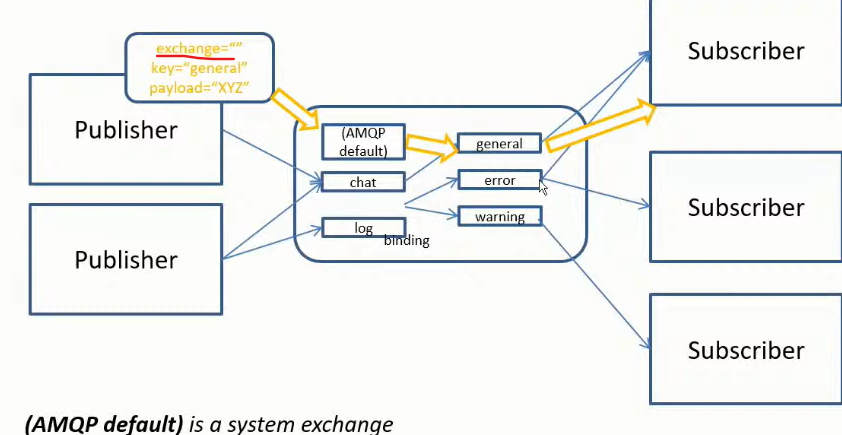
* Each message can be published with a **routing key**
* Each binding between an exchange and a queue has a **binding key**
* Routing of messages is determined based on matching between the **routing key** and the **binding key**

### Messaging patterns with RabbitMQ

* Different types of messaging patterns are implemented by means of different types of exchanges
* RabbitMQ provides the following types of exchanges:
  + default - без име като търси съвпадение на **routing key** с **binding key**
  + direct - има име като търси съвпадение на **routing key** с **binding key**
  + fanout
  + topic
  + headers- на база мачинг по хедъри също

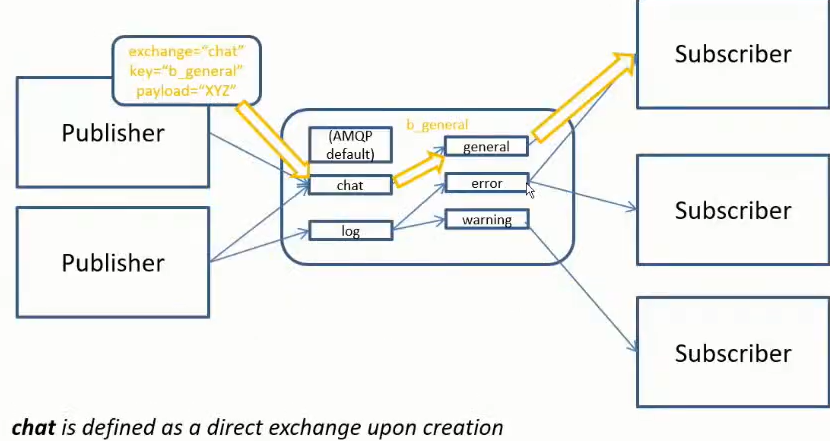
#### Default exchange

* А default exchange has **the empty string as a name** and routes messages to a queue if the routing key of the message matches the queue name (no binding needs to be declared between a default exchange and a queue)
* Default exchanges are suitable for point-to-point communication between endpoints



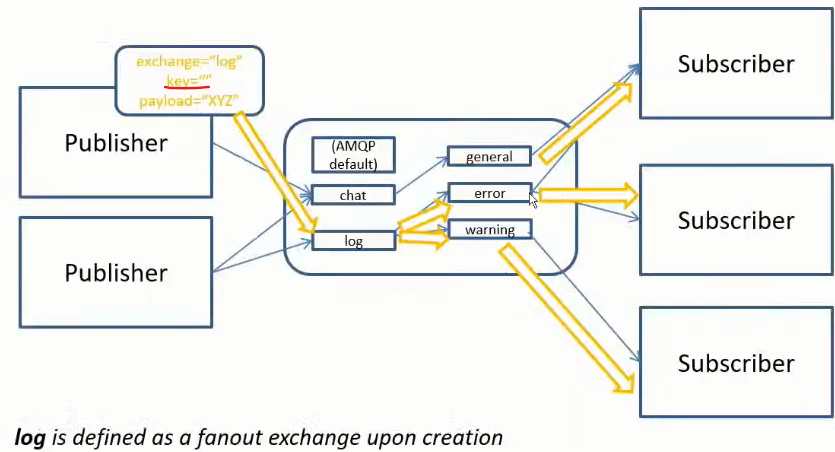
#### Direct exchange

* А direct exchange routes messages to a queue if the routing key of the message matches the binding key between the direct exchange and the queue
* Direct exchanges are suitable for point-to-point communication between endpoints
* Binding key should be defined here mandatory



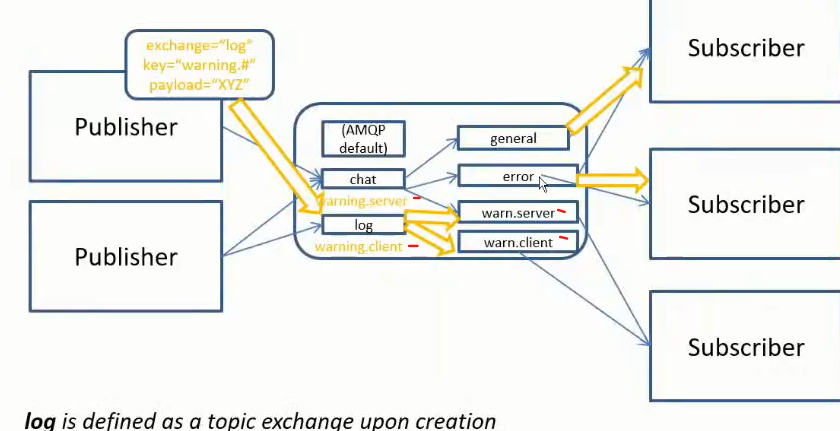
#### Fanout exchange

* А fanout exchange routes (broadcasts) messages to all queues that are bound to it (the binding key is not used)
* Fanout exchanges are suitable for publish-subscribe communication between endpoints



#### Topic exchange

* А topic exchange routes (multicasts) messages to all queues that have a binding key (can be a pattern) that matches the routing key of the message
* Topic exchanges are suitable for routing messages to different queues based on the type of message
* Диеса след *warning.* може да е всякаква дума

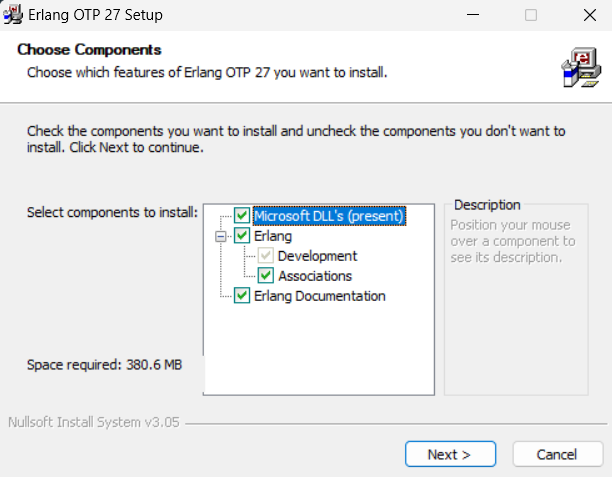


#### Headers exchange

* А headers exchange routes messages based on a custom message header
* Header exchanges are suitable for routing messages to different queues based on more than one attribute

### Installation of the RabbitMQ server

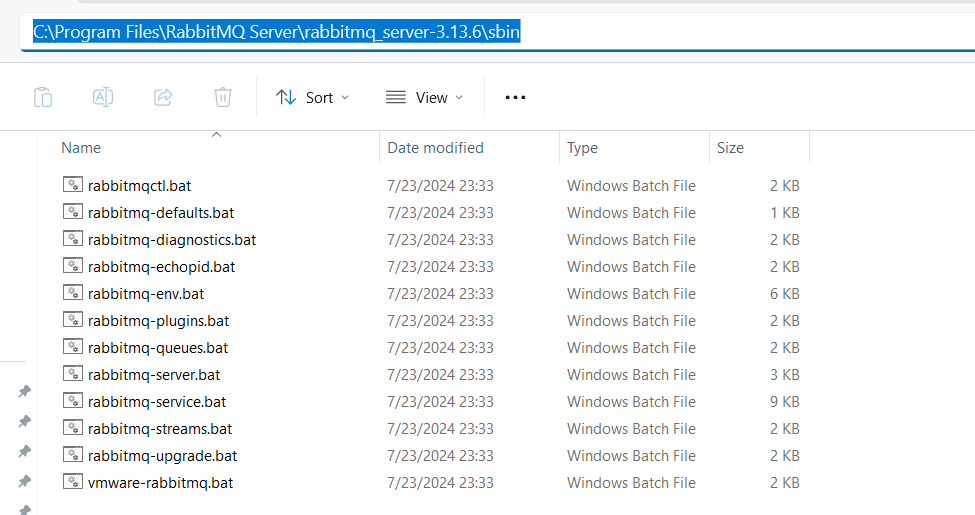
First install the Erlang - <https://www.erlang.org/downloads>



Then install the RabbitMQ server - <https://www.rabbitmq.com/docs/download>

rabbitmq-service.bat

rabbitmq-service.bat



C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.13.6\sbin>**rabbitmq-plugins.bat enable** rabbitmq\_management

Enabling plugins on node rabbit@SVILKATA:

rabbitmq\_management

The following plugins have been configured:

  rabbitmq\_management

  rabbitmq\_management\_agent

  rabbitmq\_web\_dispatch

Applying plugin configuration to rabbit@SVILKATA...

The following plugins have been enabled:

  rabbitmq\_management

  rabbitmq\_management\_agent

  rabbitmq\_web\_dispatch

set 3 plugins.

Offline change; changes will take effect at broker restart.

**През CommandPrompt като администратор:**

C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.13.6\sbin>rabbitmq-plugins.bat list

Listing plugins with pattern ".\*" ...

 Configured: E = explicitly enabled; e = implicitly enabled

 | Status: [failed to contact rabbit@SVILKATA - status not shown]

 |/

[  ] rabbitmq\_amqp1\_0              3.13.6

[  ] rabbitmq\_auth\_backend\_cache   3.13.6

[  ] rabbitmq\_auth\_backend\_http    3.13.6

[  ] rabbitmq\_auth\_backend\_ldap    3.13.6

[  ] rabbitmq\_auth\_backend\_oauth2  3.13.6

[  ] rabbitmq\_auth\_mechanism\_ssl   3.13.6

[  ] rabbitmq\_consistent\_hash\_exchange 3.13.6

[  ] rabbitmq\_event\_exchange       3.13.6

[  ] rabbitmq\_federation           3.13.6

[  ] rabbitmq\_federation\_management 3.13.6

[  ] rabbitmq\_jms\_topic\_exchange   3.13.6

**[\*] rabbitmq\_management           3.13.6**

**[\*] rabbitmq\_management\_agent     3.13.6**

[  ] rabbitmq\_mqtt                 3.13.6

[  ] rabbitmq\_peer\_discovery\_aws   3.13.6

[  ] rabbitmq\_peer\_discovery\_common 3.13.6

[  ] rabbitmq\_peer\_discovery\_consul 3.13.6

[  ] rabbitmq\_peer\_discovery\_etcd  3.13.6

[  ] rabbitmq\_peer\_discovery\_k8s   3.13.6

[  ] rabbitmq\_prometheus           3.13.6

[  ] rabbitmq\_random\_exchange      3.13.6

[  ] rabbitmq\_recent\_history\_exchange  3.13.6

[  ] rabbitmq\_sharding             3.13.6

[  ] rabbitmq\_shovel               3.13.6

[  ] rabbitmq\_shovel\_management    3.13.6

[  ] rabbitmq\_stomp                3.13.6

[  ] rabbitmq\_stream               3.13.6

[  ] rabbitmq\_stream\_management    3.13.6

[  ] rabbitmq\_top                  3.13.6

[  ] rabbitmq\_tracing              3.13.6

[  ] rabbitmq\_trust\_store          3.13.6

**[\*] rabbitmq\_web\_dispatch         3.13.6**

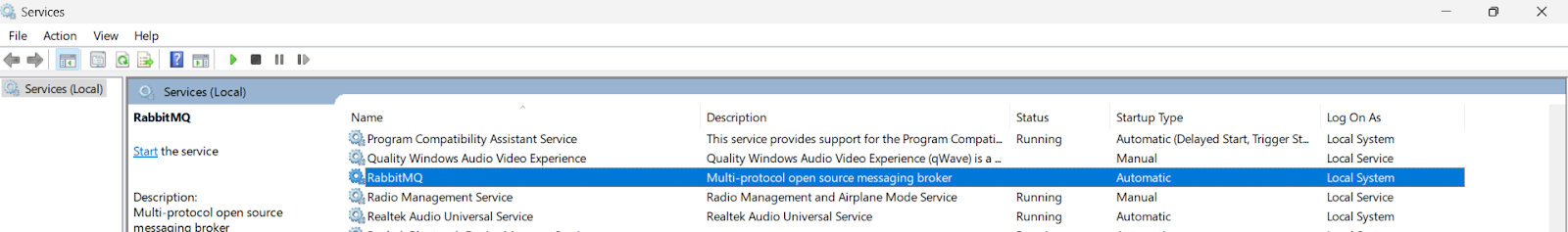
[  ] rabbitmq\_web\_mqtt             3.13.6

[  ] rabbitmq\_web\_mqtt\_examples    3.13.6

[  ] rabbitmq\_web\_stomp            3.13.6

[  ] rabbitmq\_web\_stomp\_examples   3.13.6

Type services.msc



C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.13.6\sbin>**rabbitmq-server.bat**

2024-07-31 11:52:47.571000+03:00 [warning] <0.134.0> Using RABBITMQ\_ADVANCED\_CONFIG\_FILE: c:/Users/svilk/AppData/Roaming/RabbitMQ/advanced.config

2024-07-31 11:52:52.145000+03:00 [notice] <0.45.0> Application syslog exited with reason: stopped

2024-07-31 11:52:52.145000+03:00 [notice] <0.213.0> Logging: switching to configured handler(s); following messages may not be visible in this log output

  ##  ##  RabbitMQ 3.13.6

  ##  ##

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

  ######  ##

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

  Erlang:  27.0.1 [jit]

  TLS Library: OpenSSL - OpenSSL 3.1.0 14 Mar 2023

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

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

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

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

  Monitoring:  https://www.rabbitmq.com/docs/monitoring

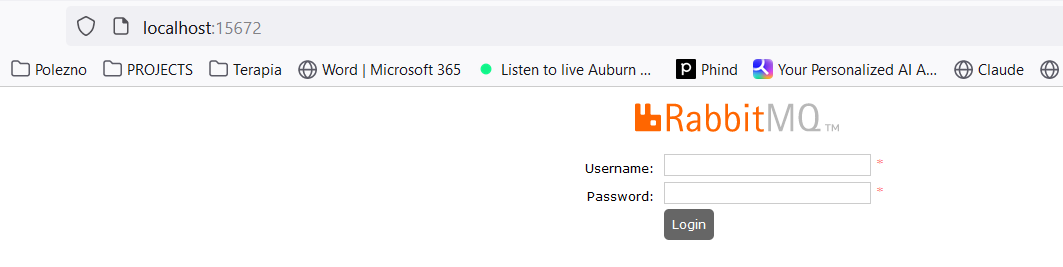
  Upgrading:   https://www.rabbitmq.com/docs/upgrade

  Logs: <stdout>

     c:/Users/svilk/AppData/Roaming/RabbitMQ/log/rabbit@SVILKATA.log

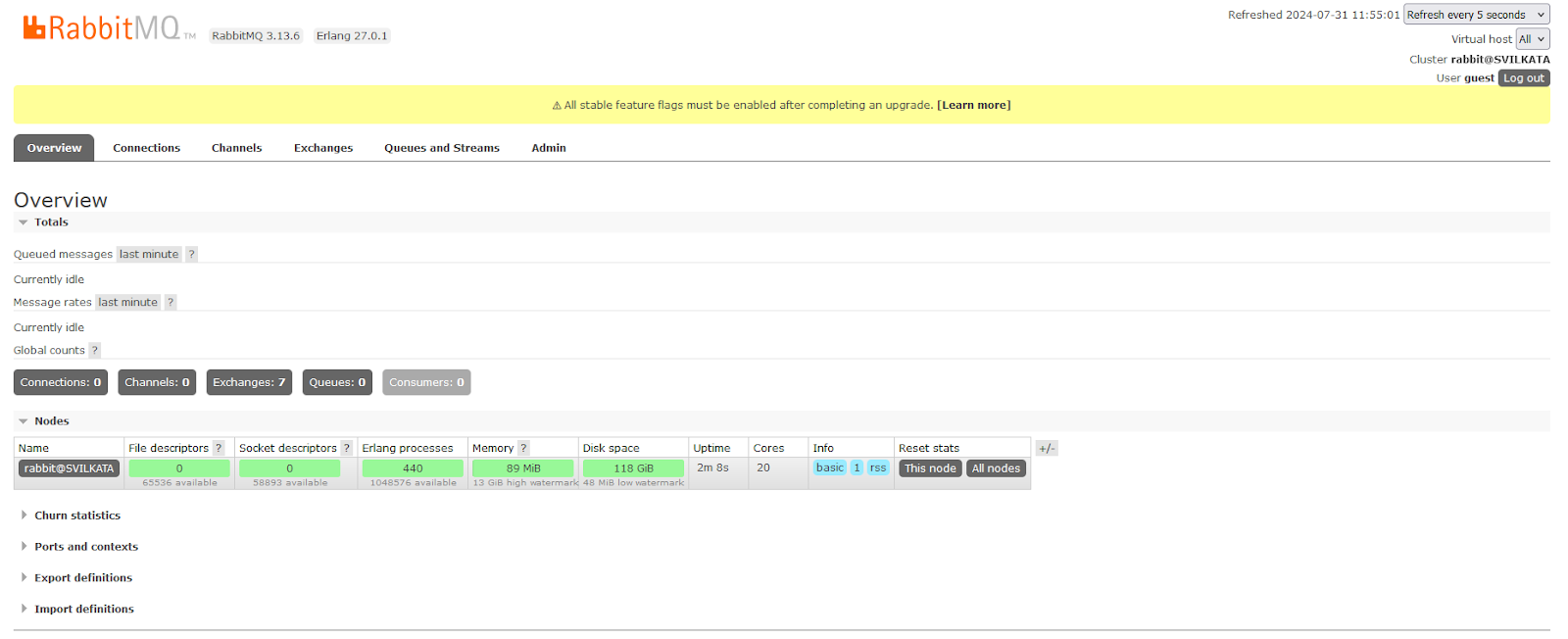
  Config file(s): c:/Users/svilk/AppData/Roaming/RabbitMQ/advanced.config

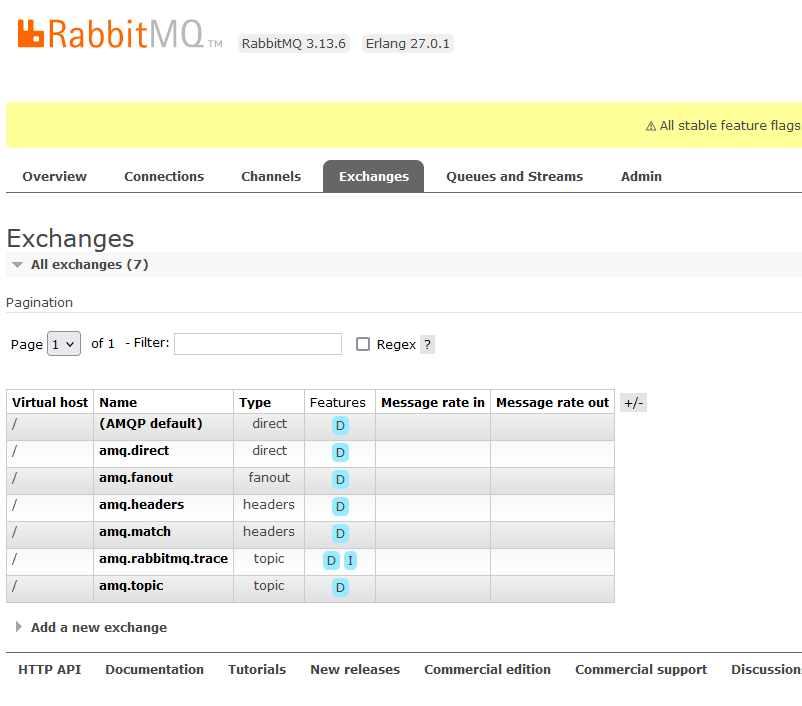
  Starting broker... **completed with 3 plugins.**

****

**username: guest**

**password: guest**

****

****

### Using the Java Client

<dependency>

 <groupId>com.rabbitmq</groupId>

 <artifactId>amqp-client</artifactId>

 <version>5.20.0</version>

</dependency>

import com.rabbitmq.client.Channel;

import com.rabbitmq.client.Connection;

import com.rabbitmq.client.ConnectionFactory;

import java.io.IOException;

import java.nio.charset.StandardCharsets;

import java.util.concurrent.TimeoutException;

public class Publisher {

   public static void main(String[] args) throws IOException, TimeoutException {

       Connection connection = null;

       Channel channel = null;

       try {

           ConnectionFactory connectionFactory = new ConnectionFactory();

           connectionFactory.setHost("localhost"); *//by default on port 15672*

connection = connectionFactory.newConnection();

           channel = connection.createChannel();

           channel.exchange**Declare**("name\_exchange", "direct"); *//created only once on the RabbitMQ server*

channel.queue**Declare**("name\_queue", false, false, false, null); *//created only once on the RabbitMQ server*

           channel.queueBind("name\_queue", "name\_exchange", "routing\_key\_test");

           channel.basicPublish("name\_exchange", "routing\_key\_test", null,

                   "Hello RabbitMQ from Java client".getBytes(StandardCharsets.*UTF\_8*));

       } finally {

           if (channel != null) {

               channel.close();

           }

           if (connection != null) {

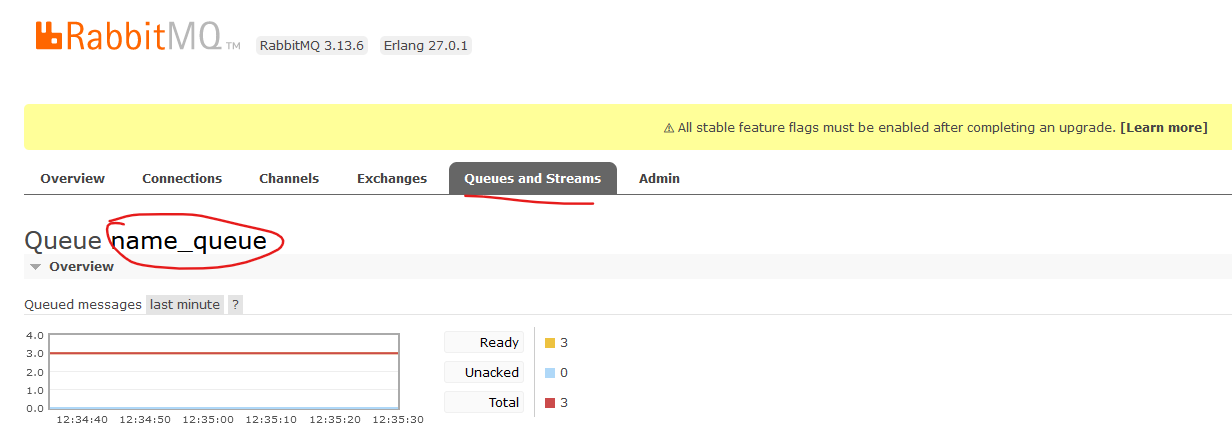
               connection.close();

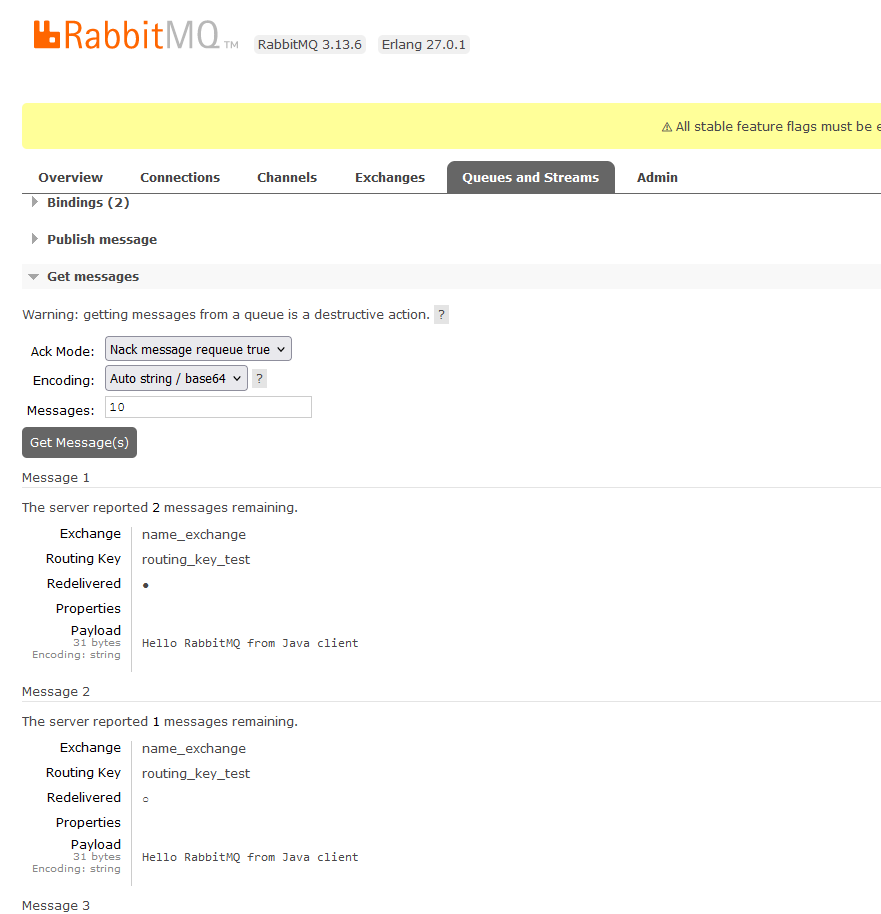
           }

       }

   }

}





import com.rabbitmq.client.AMQP;

import com.rabbitmq.client.Channel;

import com.rabbitmq.client.Connection;

import com.rabbitmq.client.ConnectionFactory;

import com.rabbitmq.client.DefaultConsumer;

import com.rabbitmq.client.Envelope;

import java.io.IOException;

import java.util.concurrent.TimeoutException;

public class Subscriber {

   public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {

       Connection connection = null;

       Channel channel = null;

       try {

           ConnectionFactory connectionFactory = new ConnectionFactory();

           connectionFactory.setHost("localhost"); *//by default on port 15672*

connection = connectionFactory.newConnection();

           channel = connection.createChannel();

           channel.exchangeDeclare("name\_exchange", "direct"); *//created only once on the RabbitMQ server*

channel.queueDeclare("name\_queue", false, false, false, null); *//created only once on the RabbitMQ server*

           channel.queueBind("name\_queue", "name\_exchange", "routing\_key\_test");

           while (true) {

               channel.basicConsume("name\_queue", true, new DefaultConsumer(channel) {

                   @Override

                   public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {

*// super.handleDelivery(consumerTag, envelope, properties, body);  no-op no work to do*

System.*out*.println(new String(body));

                   }

               });

               Thread.*sleep*(3000);

           }

       } finally {

           if (channel != null) {

               channel.close();

           }

           if (connection != null) {

               connection.close();

           }

       }

   }

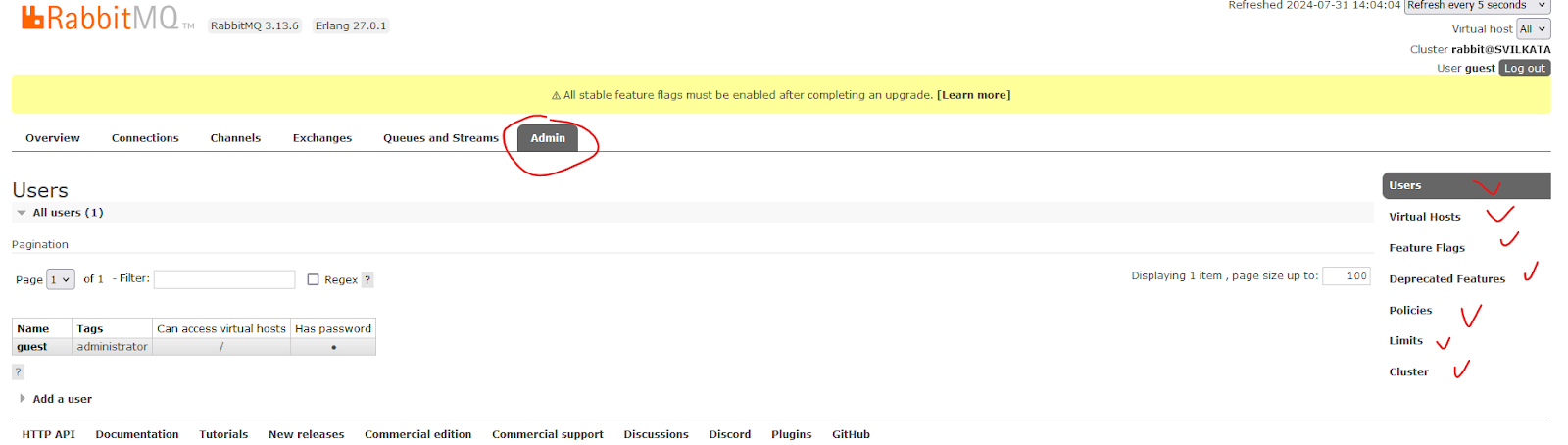
}

### Administration

* Administration of the broker includes a number of activities such as:
  + Updating the broker
  + Backing up the broker database
  + installing/uninstalling and configuring plug-ins
  + Configuring the various components of the broker

* Apart from queues, exchanges and bindings we can also manage the following types of components:
  + vhosts (virtual hosts) - for logical separation of broker components
  + users
  + Parameters - defining upstream links to another brokers
  + Policies - for queue mirroring

* Administration of single instance or an entire cluster can be performed in several ways:
  + Using the management Web interface



* Using the management HTTP API - rest API
* Using the **rabbitmq-admin.py / rabbitmqadmin.py** script - written on Python
* Using the **rabbitmqctl** utility

### Scalability and High Availability in RabbitMQ

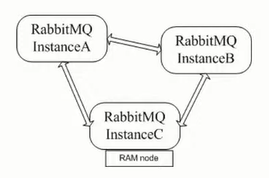
#### Basic default configuration

* RabbitMQ provides clustering support that allows new RabbitMQ nodes to be added on the fly
* Clustering by default does not guarantee that message loss may or may not occur - ако дадена инстанция примерно падне

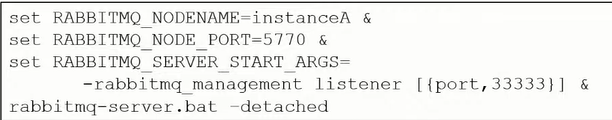
* Nodes in a RabbitMQ cluster can be:
  + DISK - data is persisted in the node database
  + RAM - data is buffered only in-memory - когато не е критично да се запазват данните след рестарт например
* **Nodes share only broker metadata - messages are not replicated among nodes!! - съобщението не се репликира в останалите node-ве**

**Example:**

А и B са DISK nodes, a C е Ram node.

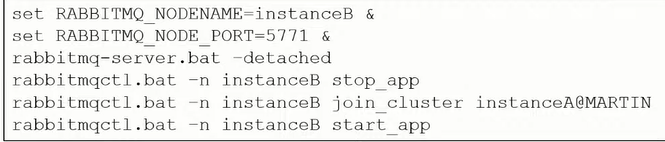
****

**Instance A node DISK**



**Instance B node DISK**

Пускаме инстанция B, спираме я, присъединяваме я след това



**Instance C node RAM**

Пускаме инстанция C, спираме я, присъединяваме я след това



* If a node that hosts a queue buffers unprocessed messages goes down, then messages are lost
* Default clustering mechanism provides scalability in terms of queues rather than high availability

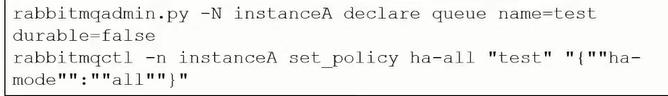
#### Mirrored queues

* **Mirrored queues** are an extension to the default clustering mechanism that can be used to establish **high availability** at the broker level
* Mirrored queues provide queue replication over different nodes that allows a message to survive node failure
* Queue mirroring is establishing by means of a mirroring policy that specifies:
  + Number of nodes to use for queue replication
  + Particular nodes designated by name for queue replication
  + All nodes for queue replication

* The node where the queue is created is the master node - all other nodes are slaves
* A new master node can be promoted in case the original one goes down
* A slave node is promoted to/as the new master in case it is fully synchronized with the old master

**Example:**

Let’s define the test queue in the cluster and mirror it over all other nodes:

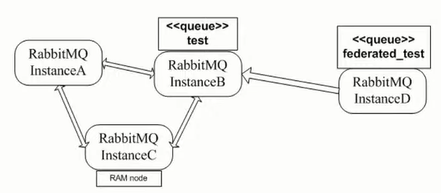


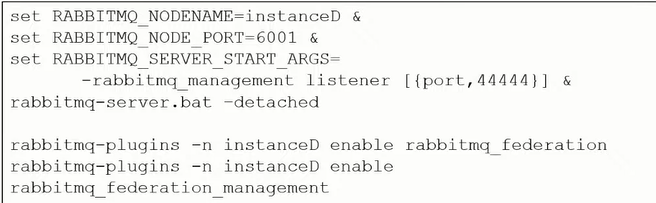
#### **Federation and Shovel plugins**

* The RabbitMQ clustering mechanism uses Erlang message passing along with a message cookie in order to establish communication between the nodes…….. which is **not reliable** over the Wide Area Networks!!
* In order to establish high availability among nodes in different geographic locations you can use the **federation, federation\_management** and **shovel** plug-ins
* The shovel plug-in works at a lower level than the federation plug-in

##### **Federation**

* Ако искаме да репликираме опашката на отдалечена инстанция D:

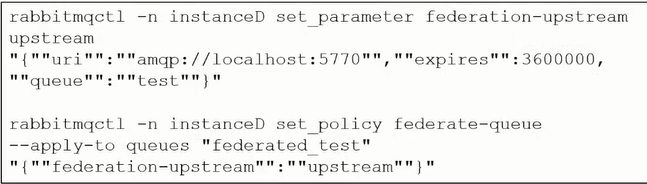




* Declare the **federated\_test** queue



Declare the upstream to the initial cluster and set a federation link to the **test** queue:



##### Shovel

The shovel plug-in provides two variants:

* **static** - all links between the source/destination nodes/clusters are defined statically in the RabbitMQ configuration file
* **dynamic** - all links between the source/destination nodes/clusters are defined dynamically via the RabbitMQ parameters



### Integrations

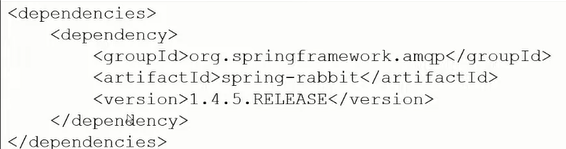
#### Info

* RabbitMQ provides integrations with other protocols such as STOMP, MQTT and LDAP by means of RabbitMQ plug-ins
* Using the Java Client - already discussed above
* The Spring framework provides integration with AMQP protocol and RabbitMQ in particular
* The **Spring AMQP framework** provides:
  + **RabbitAdmin** class for automatically declaring queues, exchanges and bindings
  + **Listener container** for asynchronous processing of inbound messages
  + **RabbitTemplate** class for sending and receiving messages

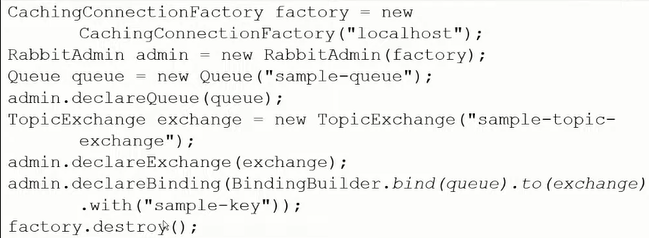
* Utilities of the Spring AMQP framework can be used directly in Java or preconfigured in the Spring configuration
* The **Spring Integration framework** to **Spring Boot** provides adapters for the AMQP protocol

* Integration with Quarkus framework

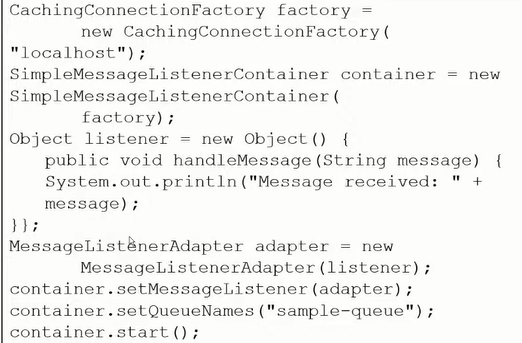
#### Spring AMQP framework



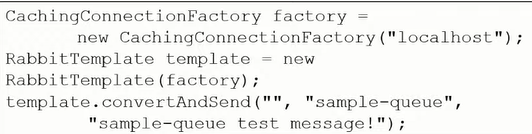
The **RabbitAdmin** class:



**Listener container**

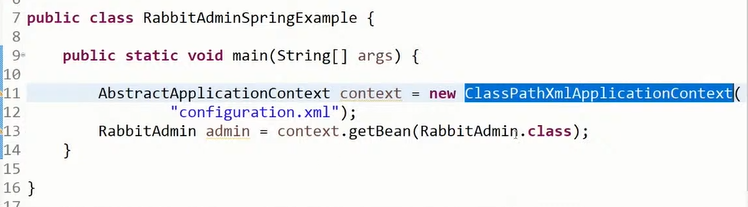
****

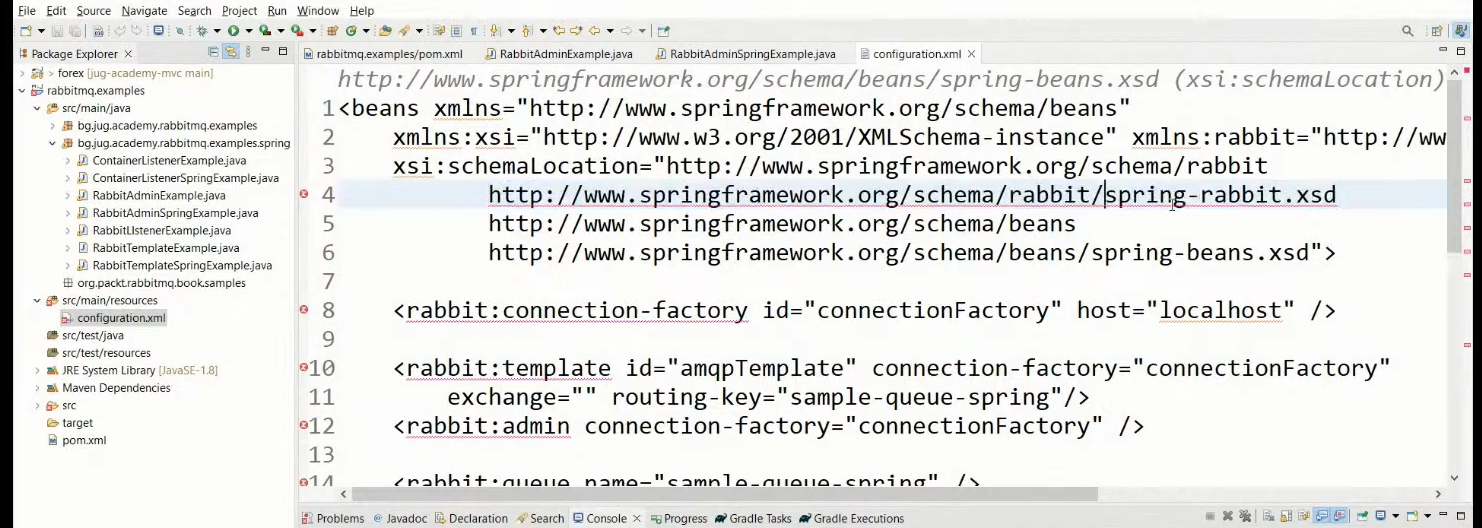
The **RabbitTemplate** class:



* All of the above Spring AMQP framework examples  can be configured using the Spring configuration - so that to be cleaner and to decouple RabbitMQ configuration from the business logic/Java code

For example within a **configuration.xml** file:





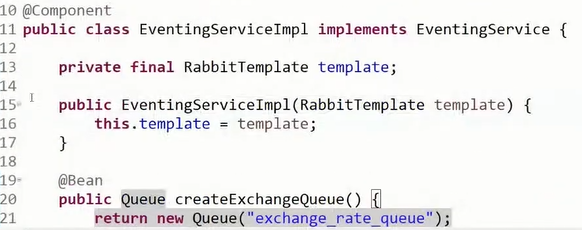
#### Spring Boot Starter AMQP - Spring Integration framework

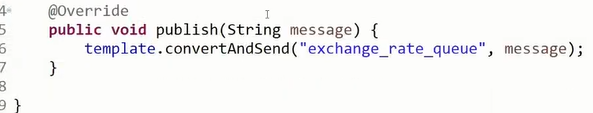
In gradle

Implementation ‘org.springframework.boot:**spring-boot-starter-amqp**’

Предоставя ни бийнове за  **RabbitAdmin** class, **Listener container** and **RabbitTemplate** class.

Посредством дефиниране на бийнове - можем да си декларираме **exchange, queue** или **queueBinding**





#### Quarkus framework

<dependency>

 <groupId>io.quarkus</groupId>

 <artifactId>quarkus-smallrye-reactive-messaging-rabbitmq</artifactId>

</dependency>

In application.properties file

mp.messaging.outgoing.bi.use-ssl=true

mp.messaging.outgoing.bi.connector=smallrye-rabbitmq

mp.messaging.outgoing.bi.exchange.type=direct

mp.messaging.outgoing.bi.port=5672

import io.smallrye.reactive.messaging.rabbitmq.**OutgoingRabbitMQMetadata**;

@Inject

@Channel("asd")  //from Microprofile

Emitter<String> emitter;  //from Microprofile

public void emmitMessage(RabbitMqPayload payload) {

   String messagePayload = JsonUtils.*toJsonString*(List.*of*(payload));

*LOGGER*.debugf("Emitting Bi message to RabbitMQ %s", messagePayload);

**OutgoingRabbitMQMetadata** metadata = new OutgoingRabbitMQMetadata.Builder()

**.withRoutingKey**(payload.routingKey())

           .build();

   Message<String> message = Message.of(messagePayload, Metadata.of(metadata));

   biEmitter.send(message); //from Microprofile

*LOGGER*.infof("Bi message emitted to route %s", payload.routingKey());

}

### Security

* RabbitMQ uses SASL Simple Authentication Security Layer for authentication (SASL PLAIN used by default)
* RabbitMQ uses access control lists (permissions) for authorization
* SSL/TLS support can be enabled for the AMQP communication channels
* SSL/TLS support can be enabled for node communication between nodes in a cluster
* SSL/TLS support can be enabled for the federation and shovel plug-ins

## II. Apache Kafka