The RabbitMQ Message Broker

Martin Toshev

Agenda

- Messaging Basics
- RabbitMQ Overview
- Messaging Patterns
- Administration

Agenda

- Scalability and High Availability
- Integrations
- Security

Messaging Basics

Messaging Basics

- 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**

Messaging Basics

- Use cases include:
  - Log aggregation between systems
  - Event propagation between systems
  - Offloading long-running tasks to worker nodes

Messaging Basics

- Messaging solutions implement different protocols for transferring of messages such as AMQP, XMPP, MQTT and many others

- The variety of protocols implies vendor lock-in when using a particular messaging solution (also called a messaging broker)

Messaging Basics

- A variety of messaging brokers can be a choice for applications …

Messaging Basics

- Messaging solutions provide means for:
  - securing message transfer, authenticating and authorizing messaging endpoints
  - routing messages between endpoints
  - subscribing to the broker

Messaging Basics

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

RabbitMQ Overview

RabbitMQ Overview

- An open source message broker written in Erlang
- Implements the AMQP Protocol (Advanced Message Queueing Protocol)
- Has a pluggable architecture and provides extension for other protocols such as HTTP, STOMP and MQTT

RabbitMQ Overview

- AMQP is a binary protocol that aims to standardize middleware communication

- The AMQP protocol derives its origins from the financial industry - processing of large volumes of financial data between different systems is a classic use case of messaging

RabbitMQ Overview

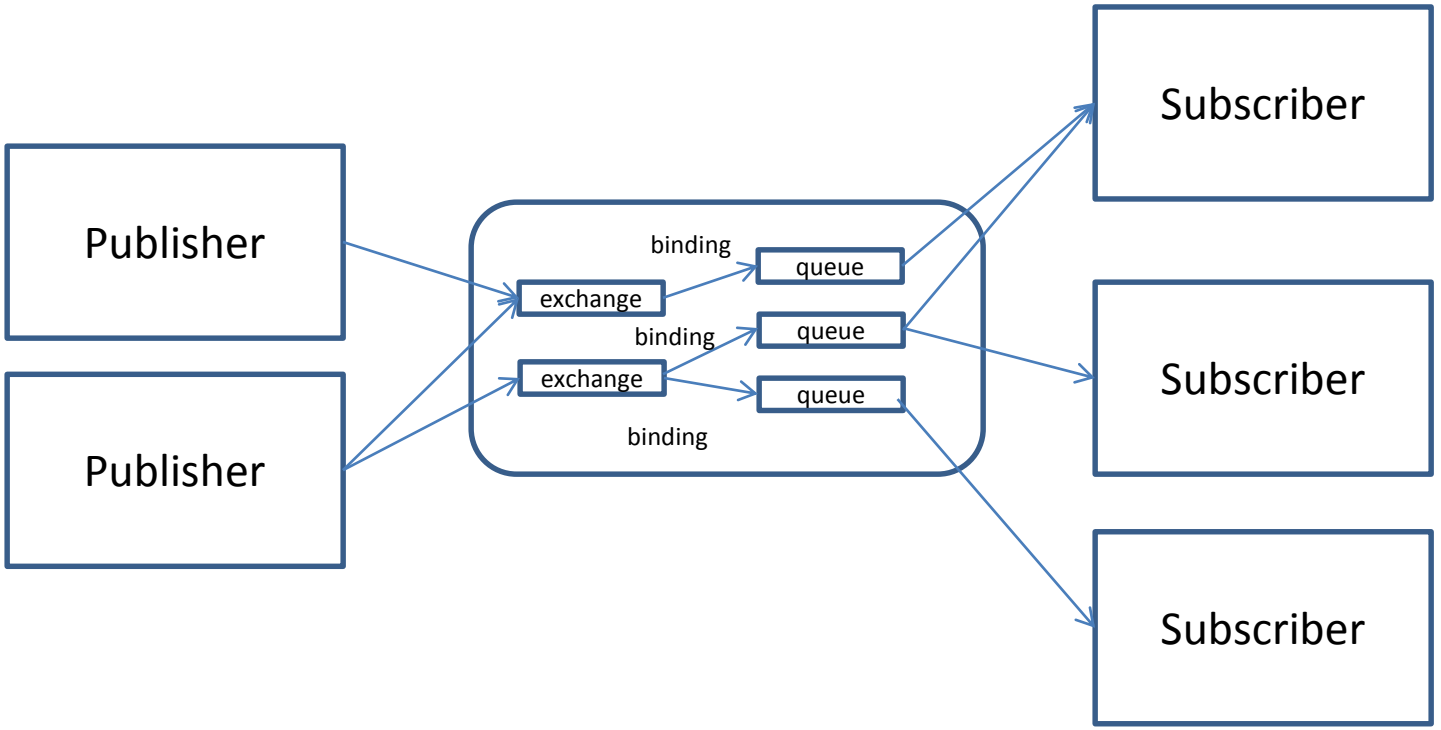
- The AMQP protocol defines:
  - **exchanges** – the message broker endpoints that receive messages
  - **queues** – the message broker endpoints that store messages from exchanges and are used by subscribers for retrieval of messages
  - **bindings** – rules that bind exchanges and queues

- The AMQP protocol is programmable – which means that the above entities can be created/modified/deleted by applications

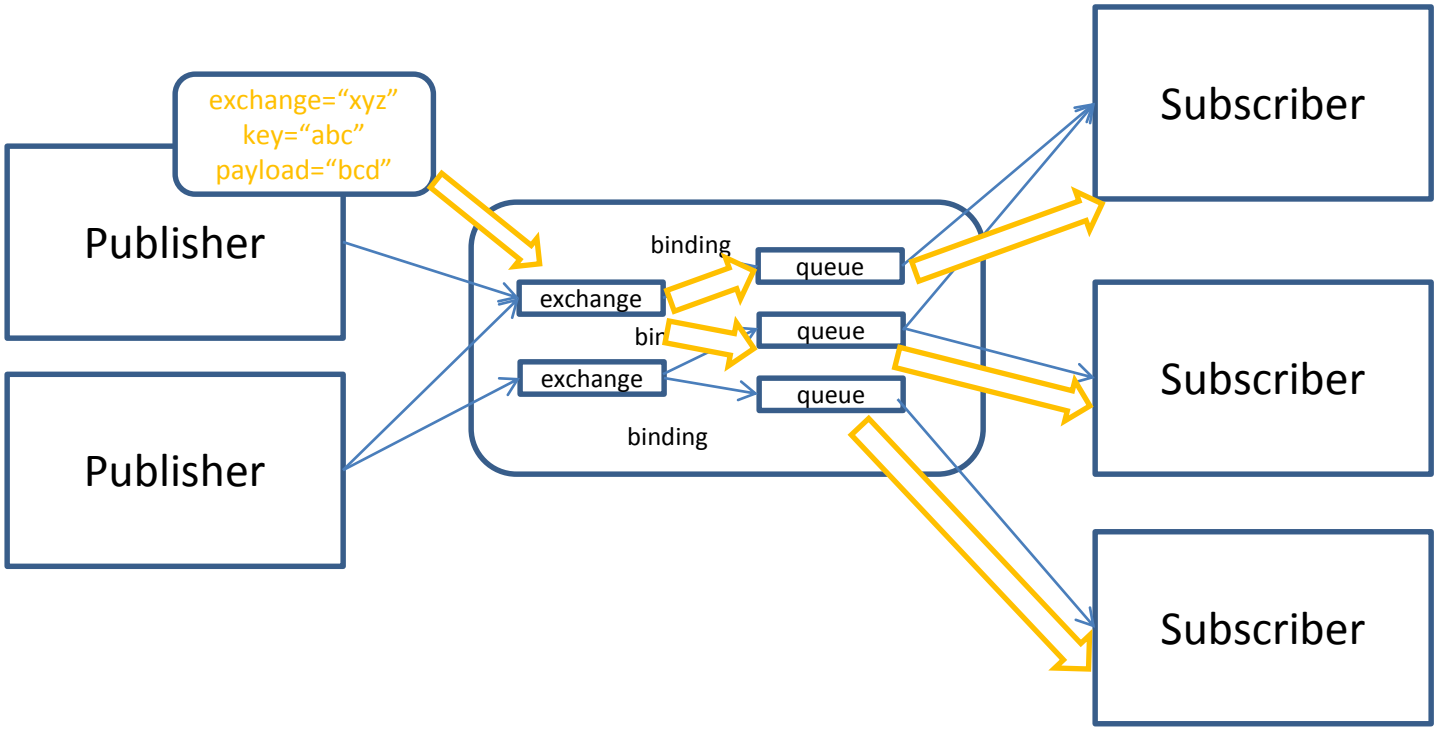
RabbitMQ Overview

- 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

RabbitMQ Overview



RabbitMQ Overview



RabbitMQ Overview

- 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 and binding keys

Messaging Patterns in RabbitMQ

Messaging Patterns with RabbitMQ

- Different types of messaging patterns are implemented by means of different types of exchanges

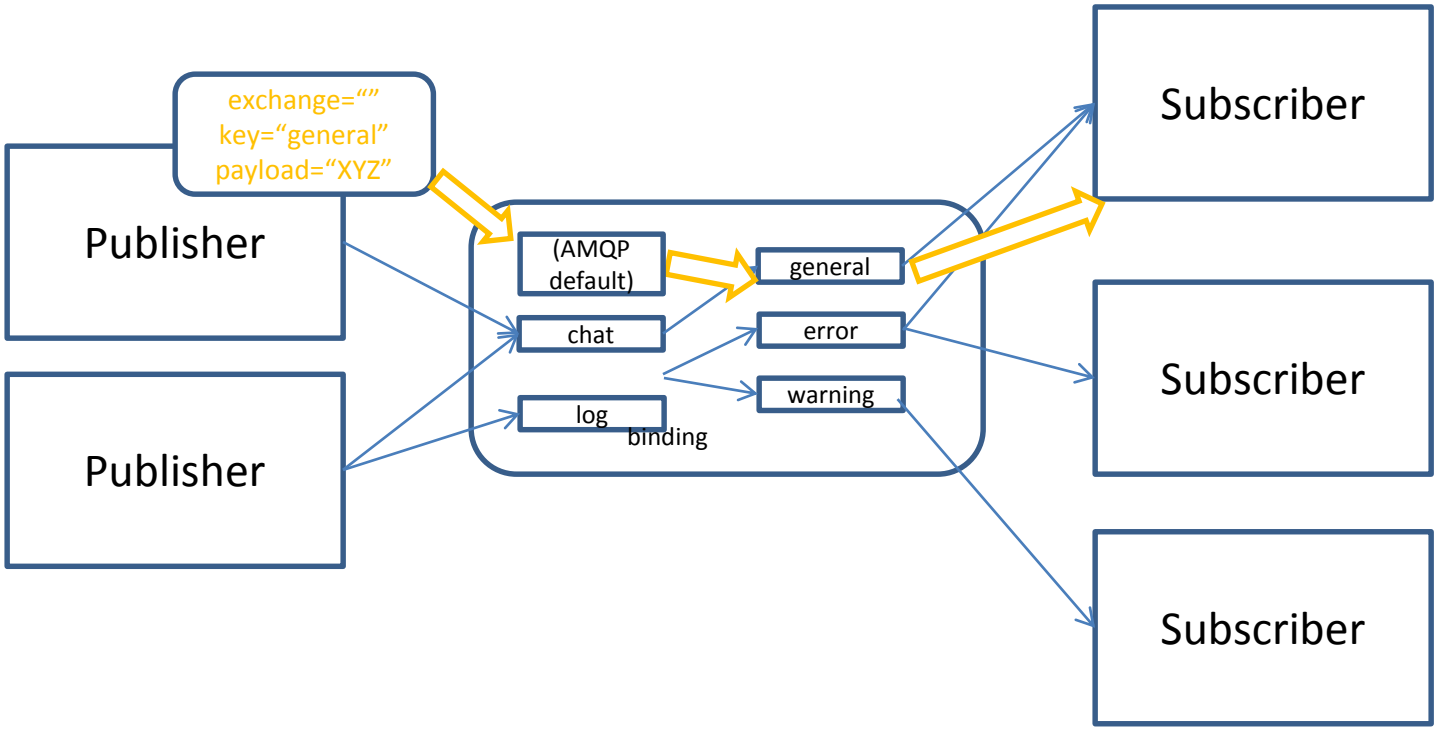
- RabbitMQ provides the following types of exchanges:
  - default
  - direct
  - fanout
  - topic
  - headers

Messaging Patterns with RabbitMQ

- A 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

RabbitMQ Overview



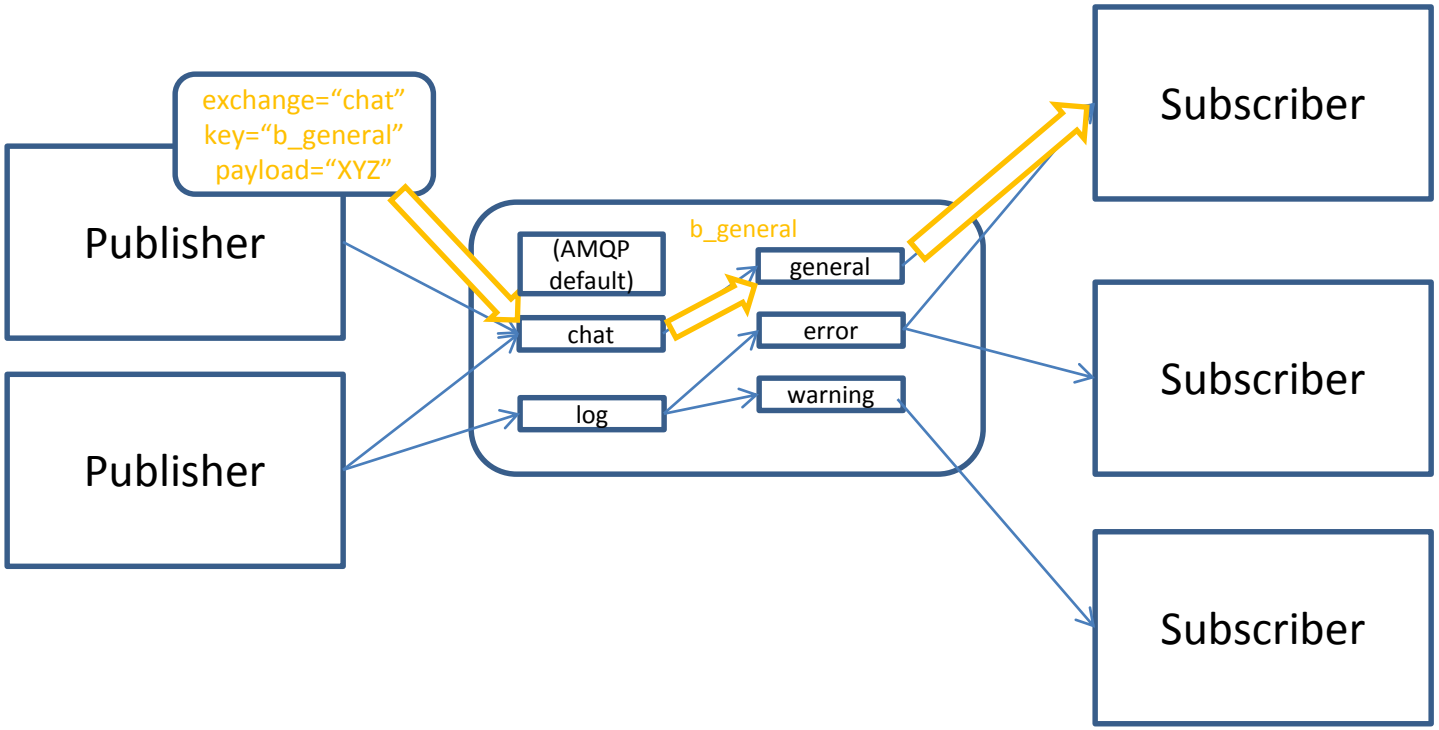
(AMQP default) *is a system exchange*

Messaging Patterns with RabbitMQ

- A 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

RabbitMQ Overview



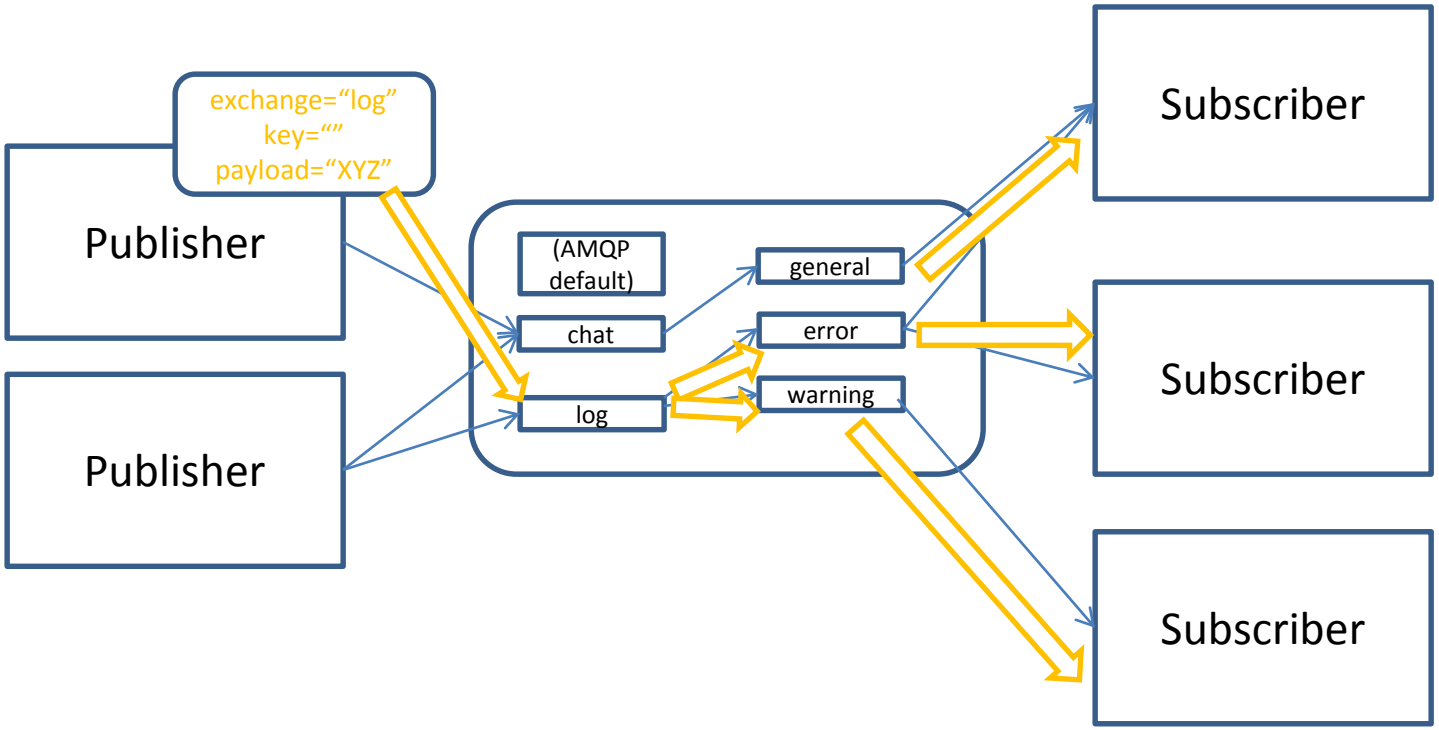
chat *is defined as a direct exchange upon creation*

Messaging Patterns with RabbitMQ

- A 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

RabbitMQ Overview



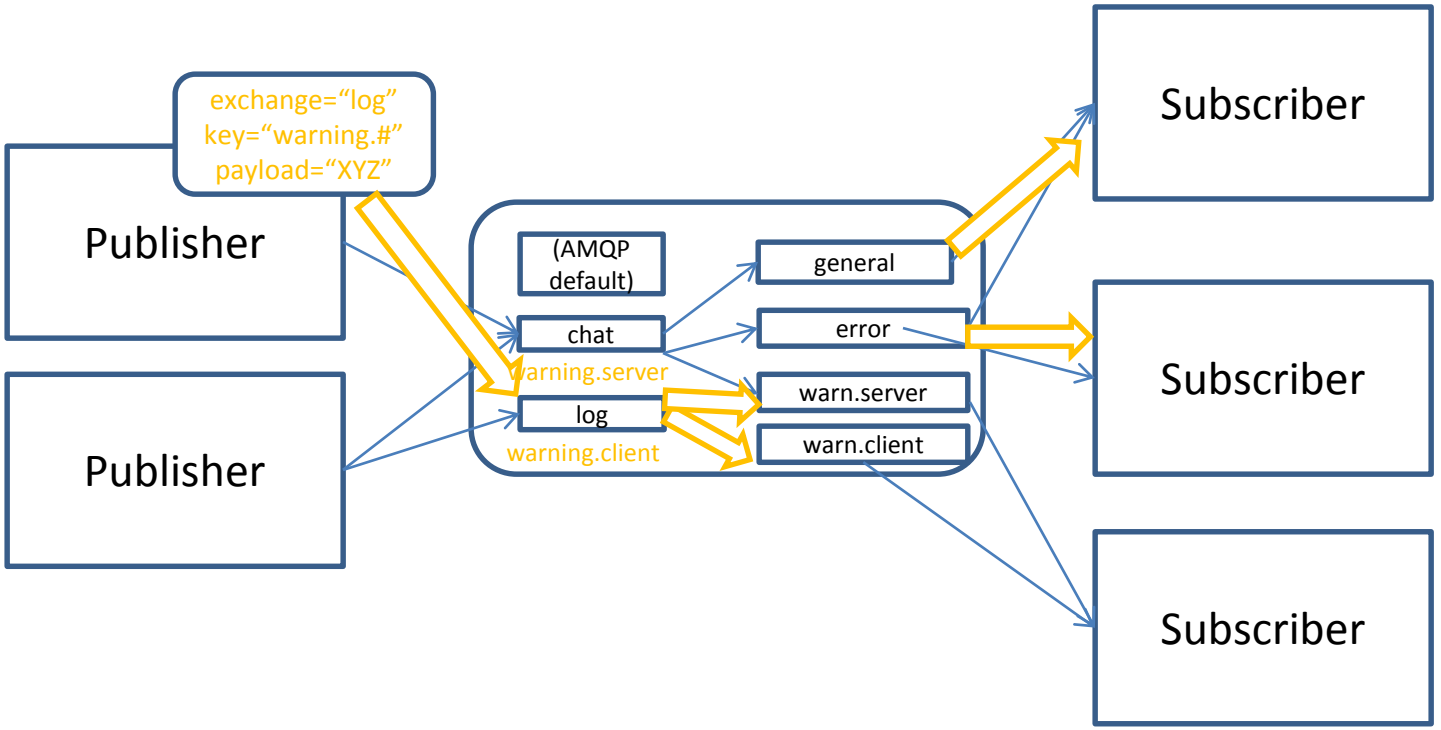
log *is defined as a fanout exchange upon creation*

Messaging Patterns with RabbitMQ

- A 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

RabbitMQ Overview



log *is defined as a topic exchange upon creation*

Messaging Patterns with RabbitMQ

- A 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

Messaging Patterns in RabbitMQ

(demo)

Administration

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

Administration

- Apart from queues, exchanges and bindings we can also manage the following types of components:
  - vhosts – for logical separation of broker components
  - users
  - parameters (e.g. for defining upstream links to another brokers)
  - policies (e.g. for queue mirroring)

Administration

- Administration of single instance or an entire cluster can be performed in several ways:
  - using the management Web interface
  - using the management HTTP API
  - using the **rabbitmq-admin.py** script
  - using the **rabbitmqctl** utility

Administration

(demo)

Scalability and High Availability in RabbitMQ

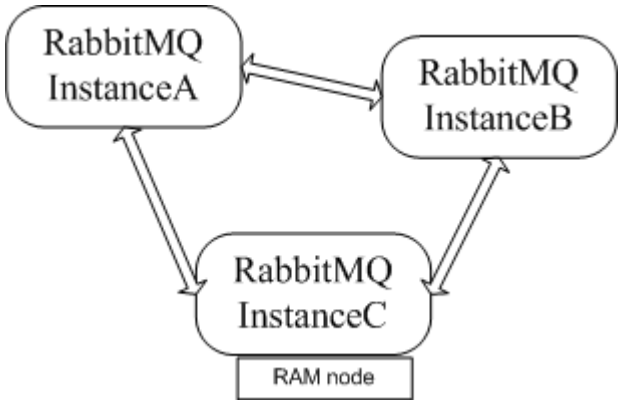
- RabbitMQ provides clustering support that allows new RabbitMQ nodes to be added on the fly
- Clustering by default does not guarantee that message loss may not occur

Scalability and High Availability in RabbitMQ

- Nodes in a RabbitMQ cluster can be:
  - DISK – data is persisted in the node database
  - RAM – data is buffered only in-memory
- Nodes share only broker metadata – messages are not replicated among nodes

Scalability and High Availability in RabbitMQ

- Let’s create the following three-node cluster:



Scalability and High Availability in RabbitMQ

- InstanceA node (root node):

```
set RABBITMQ_NODENAME=instanceA &
set RABBITMQ_NODE_PORT=5770 &
set RABBITMQ_SERVER_START_ARGS=
       -rabbitmq_management listener [{port,33333}] &
rabbitmq-server.bat -detached
```

Scalability and High Availability in RabbitMQ

- InstanceB node (DISK node):

```
set RABBITMQ_NODENAME=instanceB &
set RABBITMQ_NODE_PORT=5771 &
rabbitmq-server.bat -detached
rabbitmqctl.bat -n instanceB stop_app
rabbitmqctl.bat -n instanceB join_cluster instanceA@MARTIN
rabbitmqctl.bat -n instanceB start_app
```

Scalability and High Availability in RabbitMQ

- InstanceC node (RAM node):

```
set RABBITMQ_NODENAME=instanceC &
set RABBITMQ_NODE_PORT=5772 &
rabbitmq-server.bat -detached
rabbitmqctl.bat -n instanceC stop_app
rabbitmqctl.bat -n instanceC join_cluster -ram instanceA@MARTIN
rabbitmqctl.bat -n instanceC start_app
```

Scalability and High Availability in RabbitMQ

- However …
- If a node that hosts a queue buffers unprocessed messages goes down – the messages are lost

Scalability and High Availability in RabbitMQ

- Default clustering mechanism provides scalability in terms of queues rather than high availability

- Mirrored queues are an extension to the default clustering mechanism that can be used to establish high availability at the broker level

Scalability and High Availability in RabbitMQ

- 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

Scalability and High Availability in RabbitMQ

- The node where the queue is created is the master node – all others are slaves
- A new master node can be promoted in case the original one goes down
- A slave node is promoted to the new master in case it has fully synchronized with the old master (by default)

Scalability and High Availability in RabbitMQ

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

```
rabbitmqadmin.py -N instanceA declare queue name=test
durable=false
rabbitmqctl -n instanceA set_policy ha-all "test" "{""ha-
mode"":""all""}"
```

Scalability and High Availability in RabbitMQ

- However …
- 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 WAN …

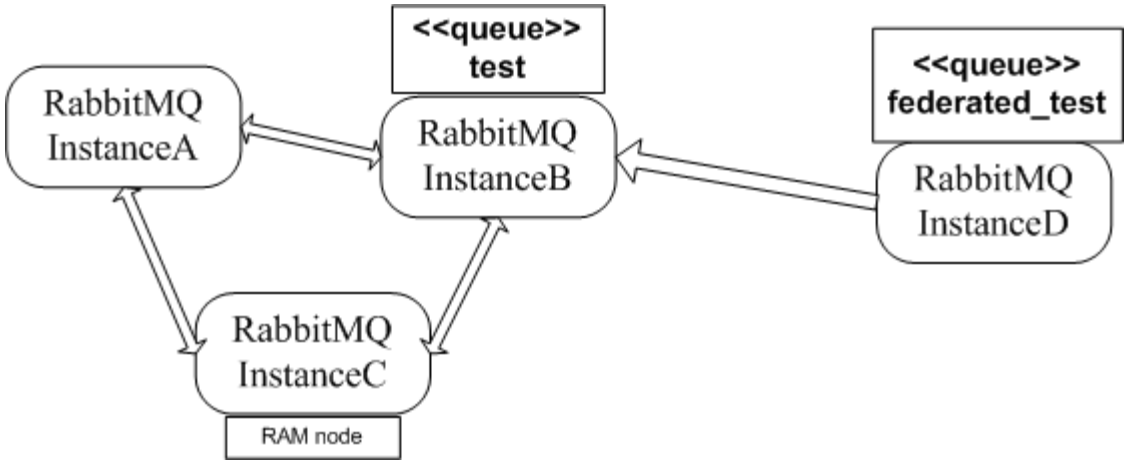
Scalability and High Availability in RabbitMQ

- In order to establish high availability among nodes in different geographic locations you can use the federation and shovel plug-ins

- The shovel plug-ins works at a lower level than the federation plug-in

Scalability and High Availability in RabbitMQ

- Assuming we have a remote node **instanceD** we can make the **test** queue federated on that node:



Scalability and High Availability in RabbitMQ

- Declare the remote **instanceD** instance and enable the federation plug-in for it:

```
set RABBITMQ_NODENAME=instanceD &
set RABBITMQ_NODE_PORT=6001 &
set RABBITMQ_SERVER_START_ARGS=
       -rabbitmq_management listener [{port,44444}] &
rabbitmq-server.bat -detached

rabbitmq-plugins -n instanceD enable rabbitmq_federation
rabbitmq-plugins -n instanceD enable
rabbitmq_federation_management
```

Scalability and High Availability in RabbitMQ

- Declare the **federated_test** queue:

```
rabbitmqadmin.py -N instanceD -P 44444 declare queue
name=federated_test durable=false
```

Scalability and High Availability in RabbitMQ

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

```
rabbitmqctl -n instanceD set_parameter federation-upstream
upstream
"{""uri"":""amqp://localhost:5770"",""expires"":3600000,
""queue"":""test""}"

rabbitmqctl -n instanceD set_policy federate-queue
--apply-to queues "federated_test"
"{""federation-upstream"":""upstream""}"
```

Scalability and High Availability in RabbitMQ

- 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 RabbitMQ parameters

Scalability and High Availability in RabbitMQ

- 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 RabbitMQ parameters

Scalability and High Availability in RabbitMQ

destination / source	exchange	queue
exchange	federation dynamic shovel	dynamic shovel
queue	static shovel dynamic shovel	federation dynamic shovel

Scalability and High Availability in RabbitMQ

(demo)

Integrations

Integrations

- RabbitMQ provides integrations with other protocols such as STOMP, MQTT and LDAP by means of RabbitMQ plug-ins
- The Spring Framework provides integration with AMQP protocol and RabbitMQ in particular

Integrations

- 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

Integrations

- Utilities of the Spring AMQP framework can used either directly in Java or preconfigured in the Spring configuration

- The Spring Integration framework provides adapters for the AMQP protocol

Integrations

- The following Maven dependency can be used to provide the Spring AMQP framework to the application:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.amqp</groupId>
        <artifactId>spring-rabbit</artifactId>
        <version>1.4.5.RELEASE</version>
    </dependency>
</dependencies>
```

Integrations

- **RabbitAdmin** (plain Java):

```
CachingConnectionFactory factory = new
      CachingConnectionFactory("localhost");
RabbitAdmin admin = new RabbitAdmin(factory);
Queue queue = new Queue("sample-queue");
admin.declareQueue(queue);
TopicExchange exchange = new TopicExchange("sample-topic-
      exchange");
admin.declareExchange(exchange);
admin.declareBinding(BindingBuilder.bind(queue).to(exchange)
      .with("sample-key"));
factory.destroy();
```

Integrations

- Container listener (plain Java):

```
CachingConnectionFactory factory =
       new CachingConnectionFactory(
"localhost");
SimpleMessageListenerContainer container = new
SimpleMessageListenerContainer(
       factory);
Object listener = new Object() {
    public void handleMessage(String message) {
    System.out.println("Message received: " +
    message);
}};
MessageListenerAdapter adapter = new
       MessageListenerAdapter(listener);
container.setMessageListener(adapter);
container.setQueueNames("sample-queue");
container.start();
```

Integrations

- **RabbitTemplate** (plain Java):

```
CachingConnectionFactory factory =
       new CachingConnectionFactory("localhost");
RabbitTemplate template = new
RabbitTemplate(factory);
template.convertAndSend("", "sample-queue",
        "sample-queue test message!");
```

Integrations

- All of the above examples can be configured using the Spring configuration
- Must cleaner and decouples RabbitMQ configuration for the business logic

Integrations

(demo)

Security

Security

- RabbitMQ uses SASL for authentication (SASL PLAIN used by default)
- RabbitMQ uses access control lists (permissions) for authorization

Security

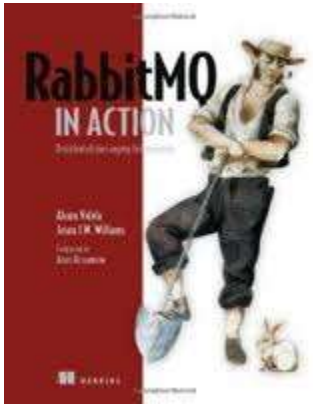
- SSL/TLS support can be enabled for the AMQP communication channels
- SSL/TLS support can be enabled for node communication between nodes in a cluster
- SSL/TLS support can be enabled for the federation and shovel plug-ins

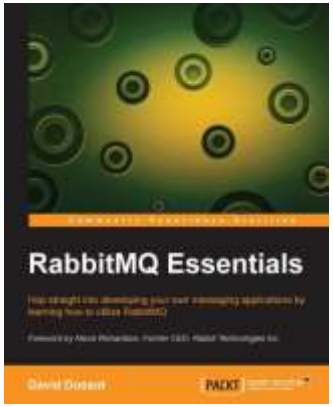
Summary

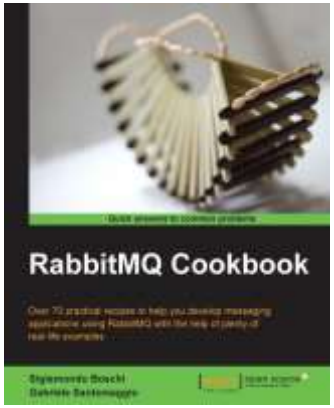
- RabbitMQ is a robust messaging solution that can be used in a variety of scenarios based on you application needs

- RabbitMQ may not be the best possible solution compared to other messaging brokers – always consider benchmarks based on size and number of messages

Readings


RabbitMQ
IN ACTION


RabbitMQ Essentials
David Dossot
PACKT


RabbitMQ Cookbook


BG JUG

Thanks

?