
RabbitMQ

Overview

Introduce the RabbitMQ platform

RabbitMQ management portal

AMQP Protocol

Exchanges

Queues, bindings and consumers

Client support

Docker setup

Demo

Introducing RabbitMQ

Reliability

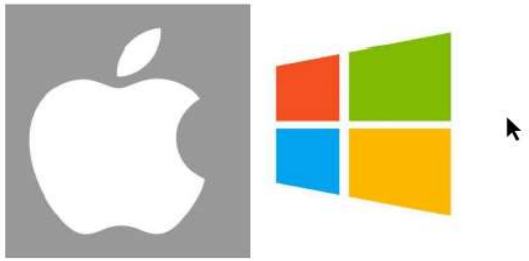
Routing

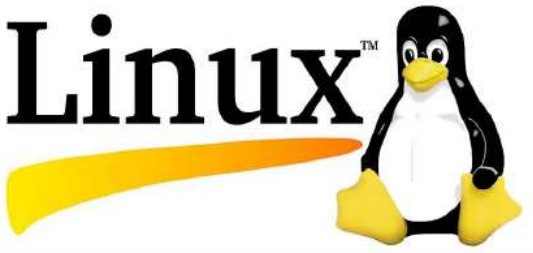
Clustering and high availability

Management web interface

Command line interface

Introducing RabbitMQ





RabbitMQ management portal

Declare, list and delete RabbitMQ entities

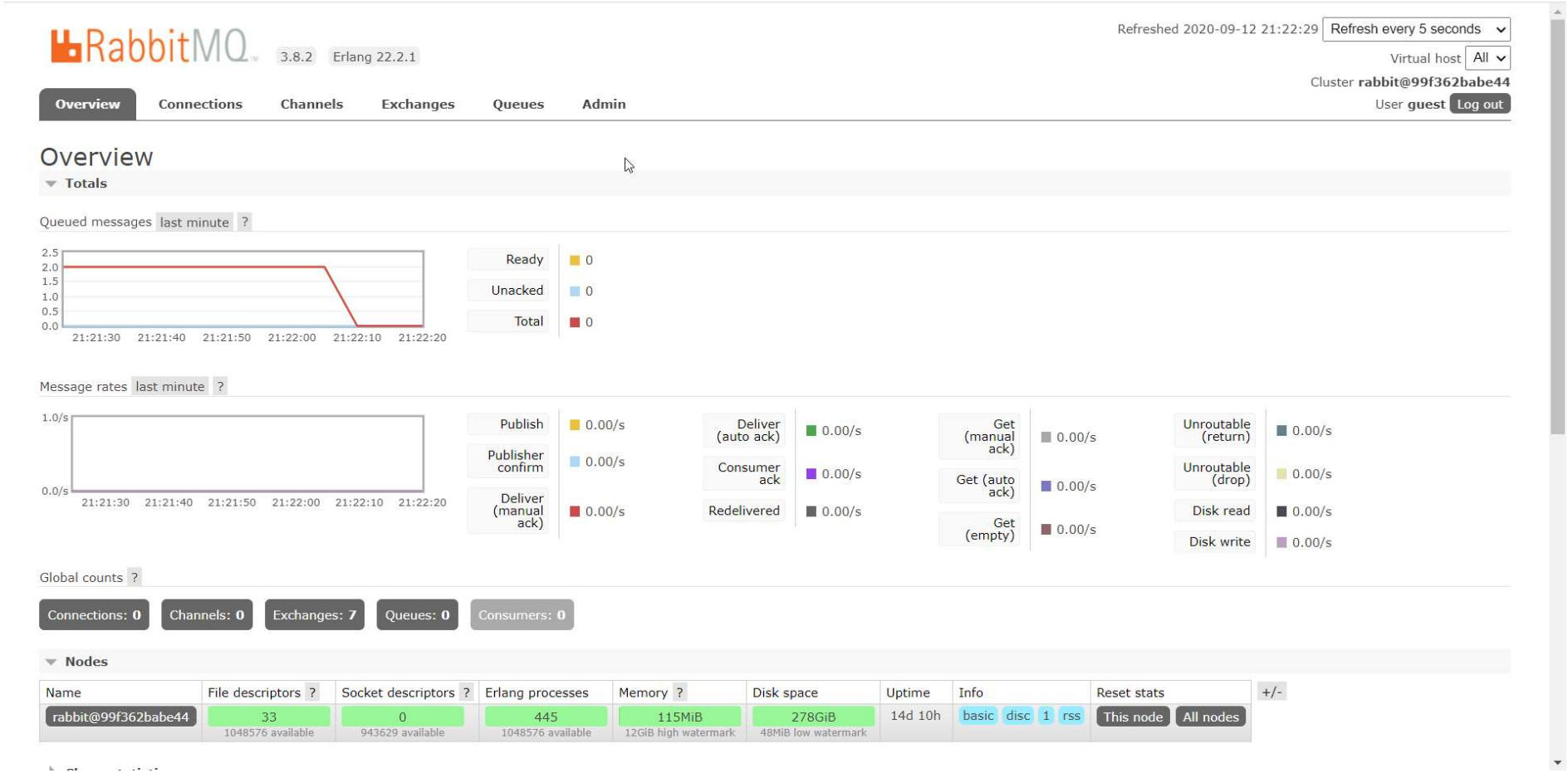
Queue and exchange monitoring

Send and receive messages

Monitor Erlang processes, file descriptors, and memory use

Force close connections, and purge queues

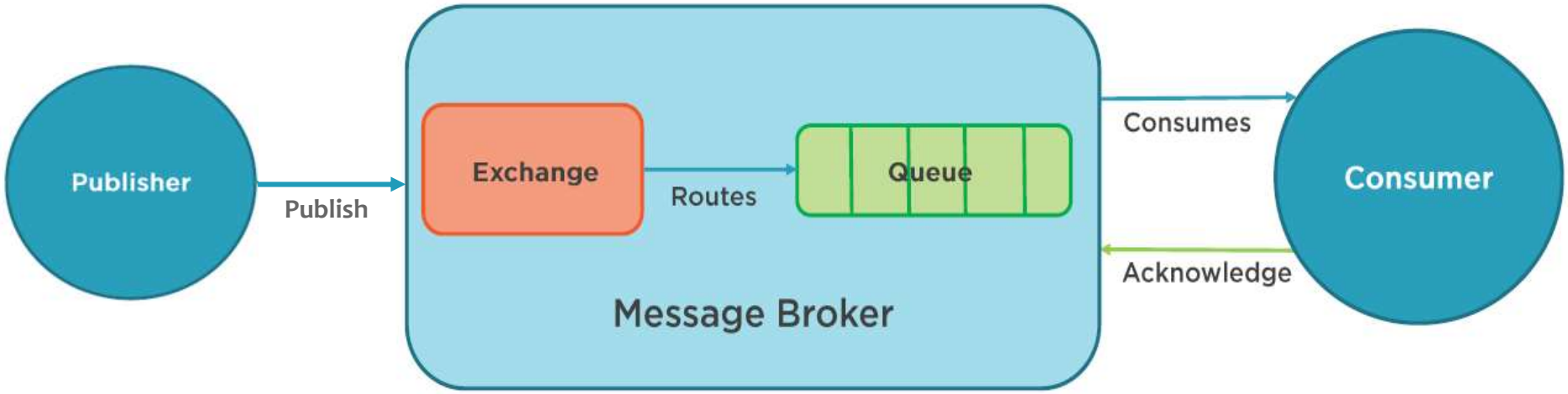
RabbitMQ management portal



AMQP Protocol

Advance Message Queueing Protocol

Supports version 0-9-1



Exchanges

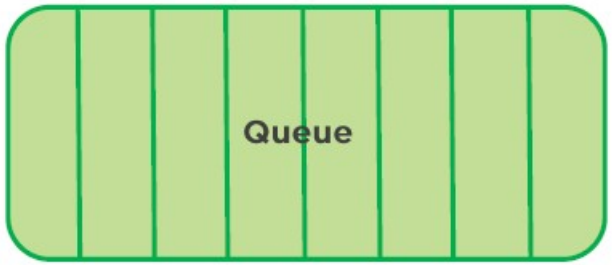
Direct Exchanges

Fanout Exchanges

Topic Exchanges

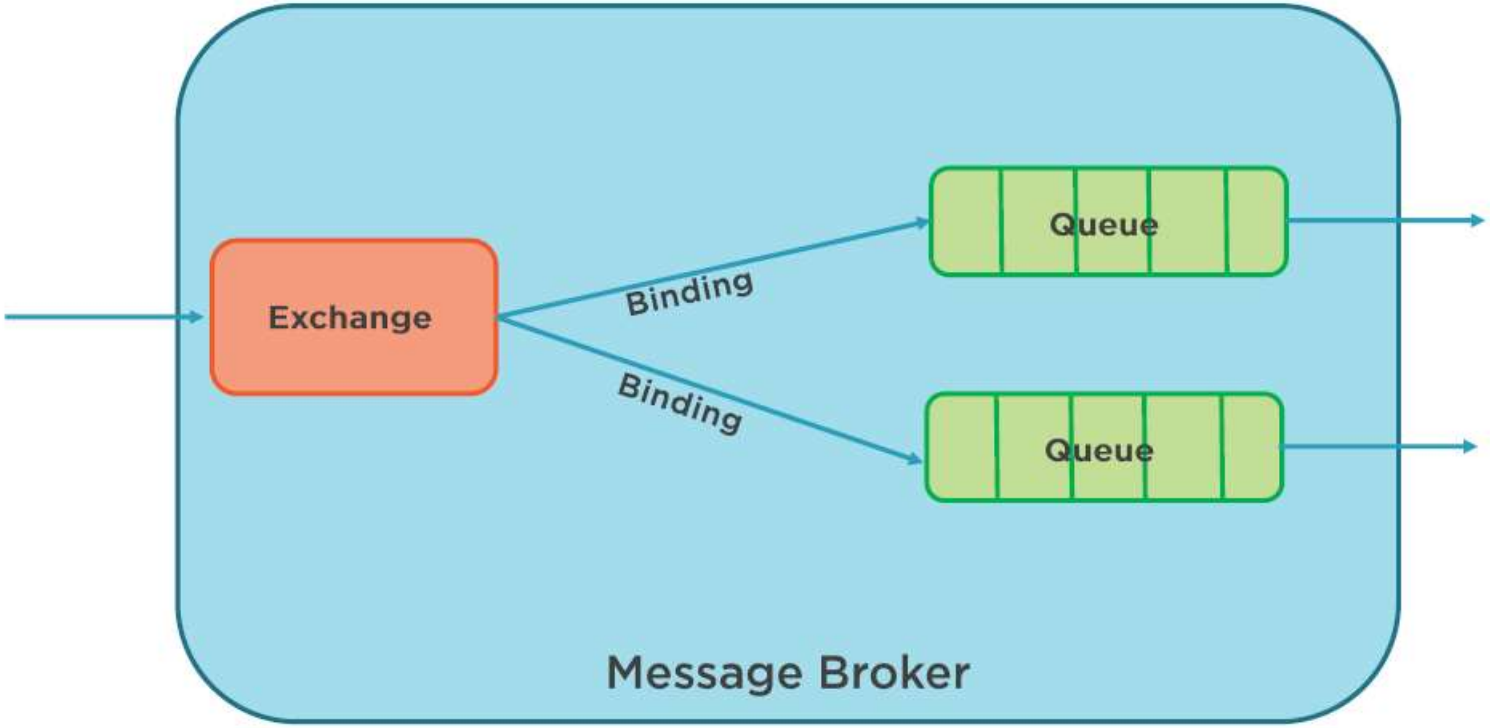
Header Exchanges

Queue

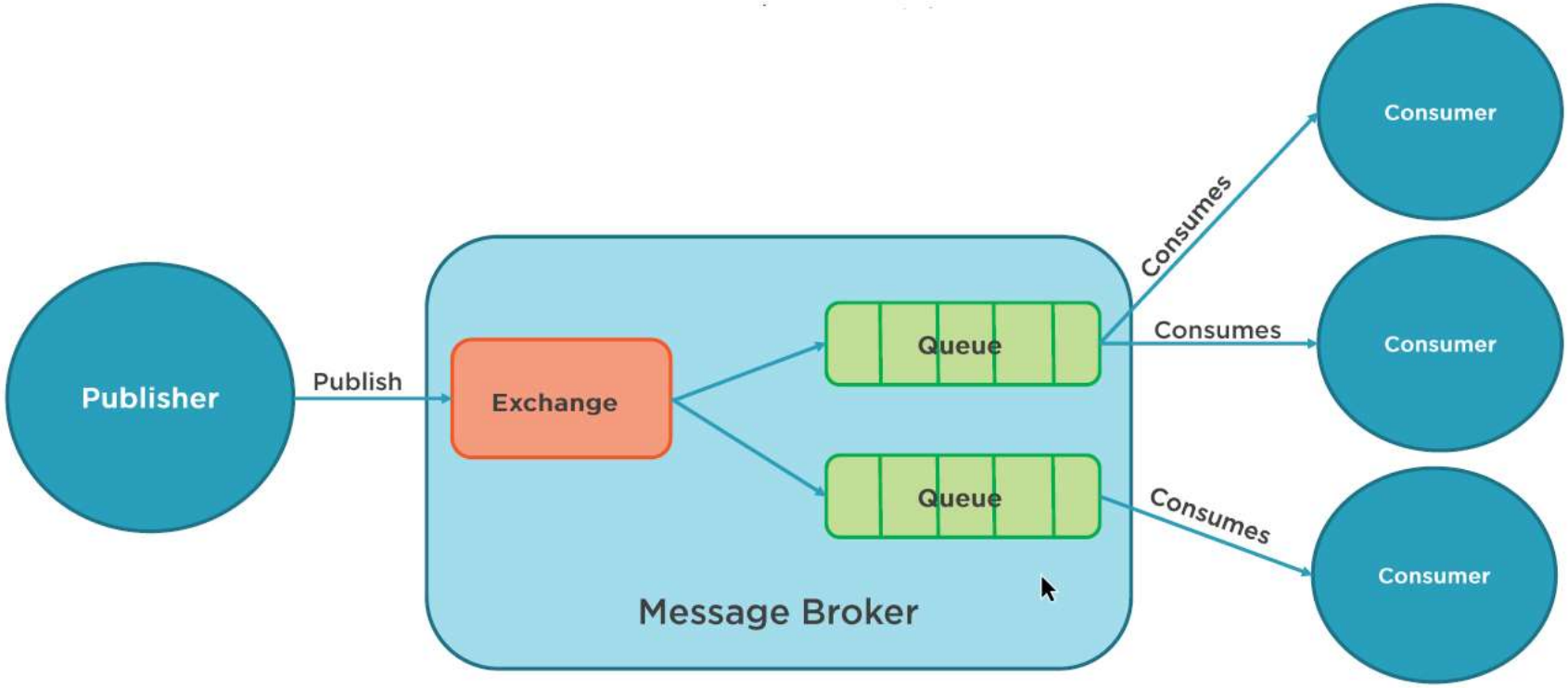


Name	*The name of the queue*
Durable	*Persisting the queue to disk*
Exclusive	*Delete queue when not needed*
Auto Delete	*Queue deleted when consumer unsubscribes*

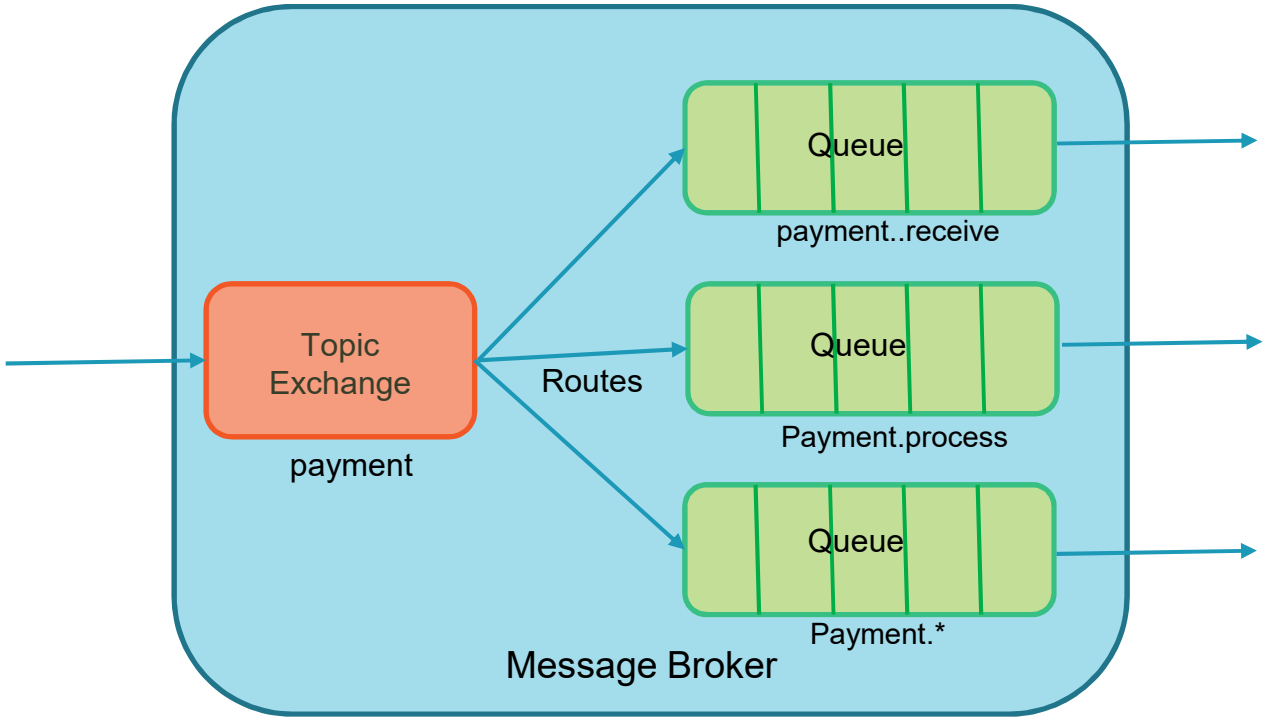
Binding



Consumer


Publisher
Publish
Exchange
Queue
Queue
Message Broker
Consumes
Consumes
Consumes
Consumer
Consumer
Consumer

Topic Exchanges



Client support

RabbitMQ

Features | Get Started | Support | Community | Docs | Blog

Introduction

RabbitMQ is a message broker: it accepts and forwards messages. You can think about it as a post office: when you put the mail that you want posting in a post box, you can be sure that Mr. or Ms. Mailperson will eventually deliver the mail to your recipient. In this analogy, RabbitMQ is a post box, a post office and a postman.

The major difference between RabbitMQ and the post office is that it doesn't deal with paper, instead it accepts, stores and forwards binary blobs of data – *messages*.

RabbitMQ, and messaging in general, uses some jargon.

- *Producing* means nothing more than sending. A program that sends messages is a *producer*:

- *A queue* is the name for a post box which lives inside RabbitMQ. Although messages flow through RabbitMQ and your applications, they can only be stored inside a *queue*. A *queue* is only bound by the host's memory & disk limits, it's essentially a large message buffer. Many *producers* can send messages that go to one queue, and many *consumers* can try to receive data from one *queue*. This is how we represent a queue:

Prerequisites

This tutorial assumes RabbitMQ is installed and running on `localhost` on standard port (`5672`). In case you use a different host, port or credentials, connections settings would require adjusting.

Where to get help

If you're having trouble going through this tutorial you can contact us through the mailing list.

1 "Hello World!"

The simplest thing that does *something*

- Python
- Java
- Ruby
- PHP
- C#
- JavaScript
- Go
- Elixir
- Objective-C
- Swift
- Spring AMQP

2 Work queues

Distributing tasks among workers (the competing consumers

https://www.rabbitmq.com/tutorials/tutorial-one-java.html

RabbitMQ Docker

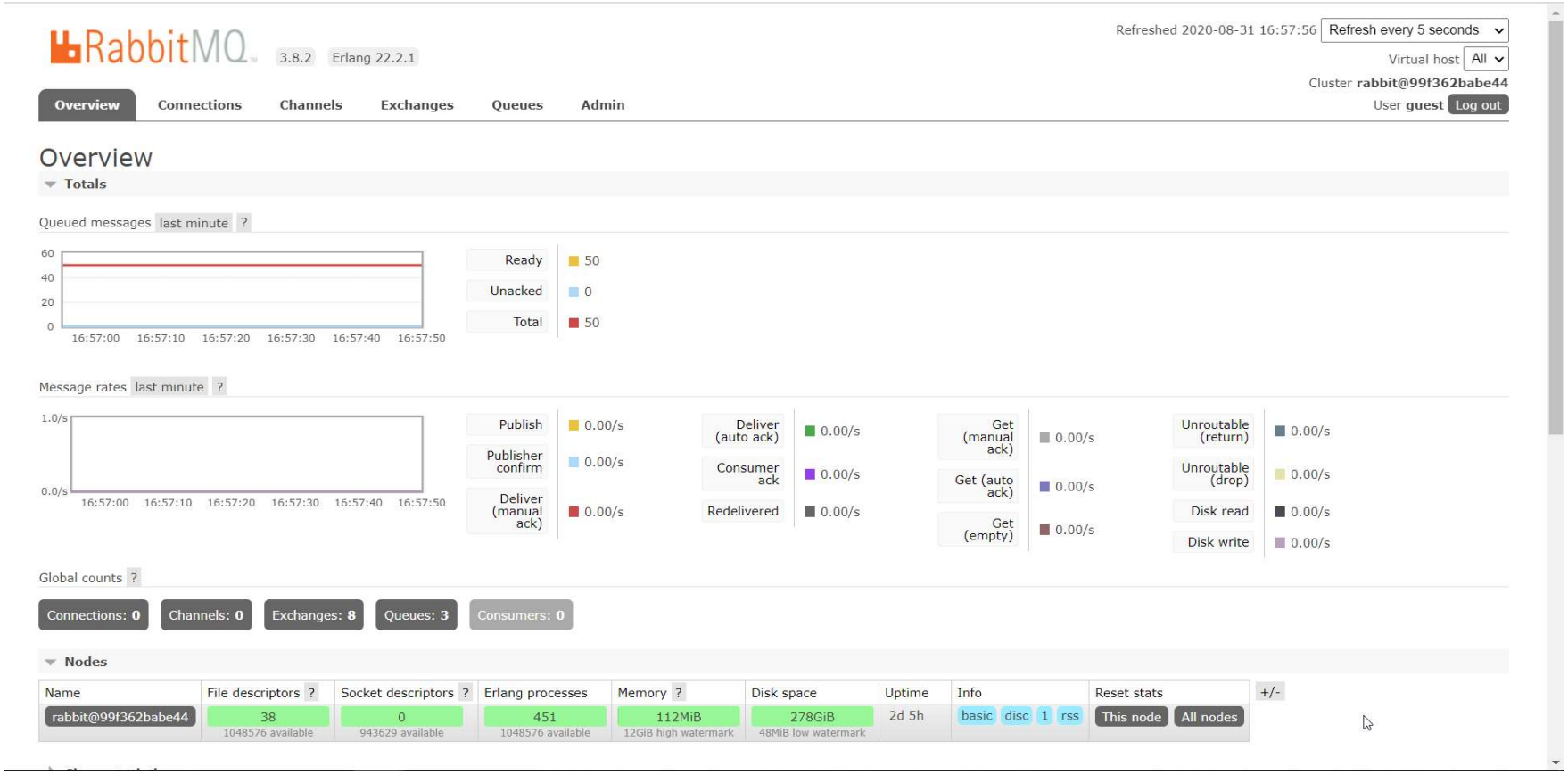
```
$ cat docker-compose.yml
version: "3.3"
services:
  rabbitmq:
    image: rabbitmq:3-management
    container_name: rabbitmq
    networks:
      - cluster
    volumes:
      - /[redacted]/rabbitmq/data/:/var/lib/rabbitmq/
      - /[redacted]/rabbitmq/logs/:/var/log/rabbitmq/
    environment:
      RABBITMQ_DEFAULT_USER: ${RABBITMQ_DEFAULT_USER}
      RABBITMQ_DEFAULT_PASS: ${RABBITMQ_DEFAULT_PASS}
    ports:
      - 5672:5672
      - 15672:15672
networks:
  cluster:
  $
$
$
```

RabbitMQ Docker

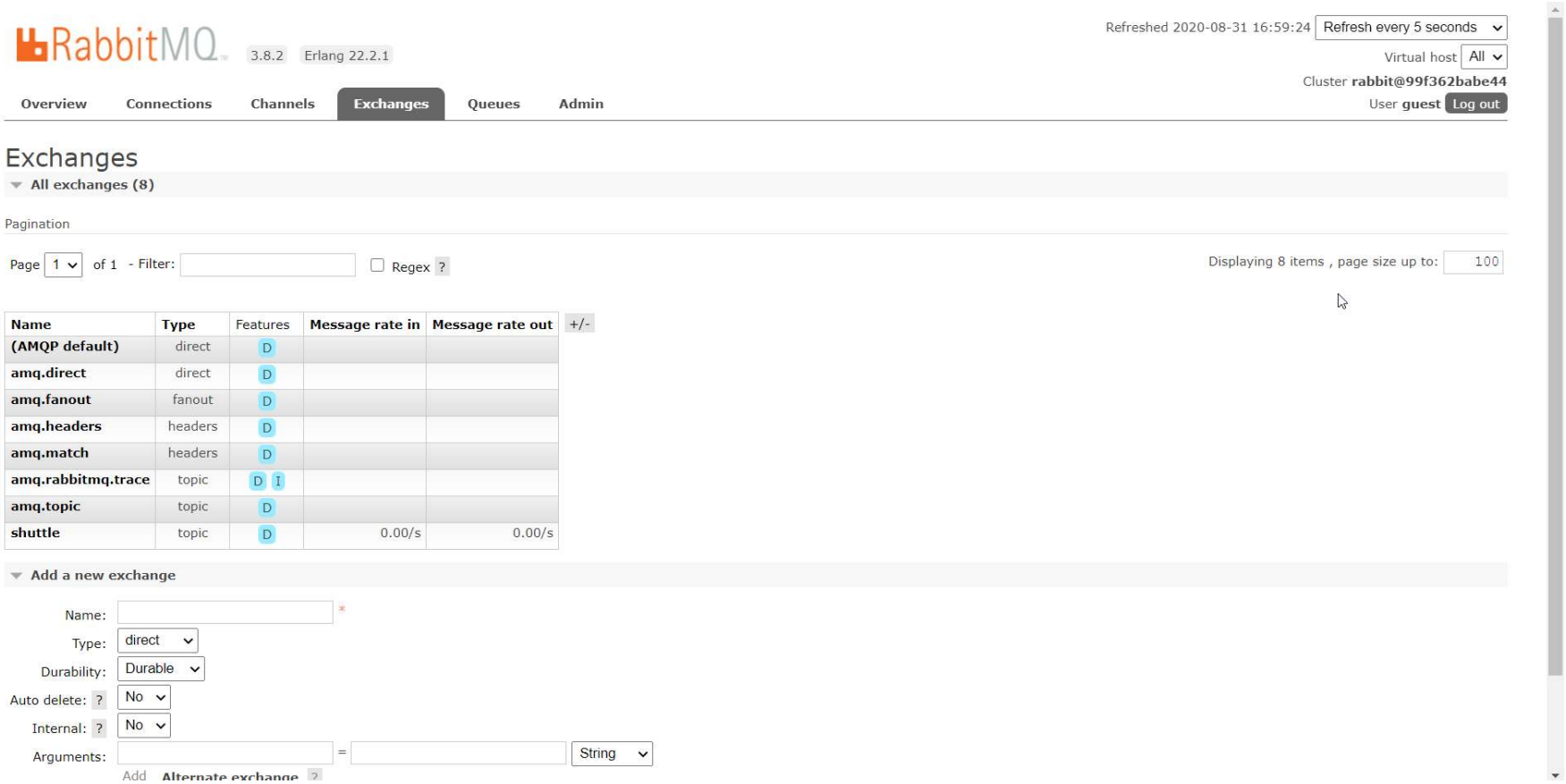
- 4369: epmd, a peer discovery service used by RabbitMQ nodes and CLI tools
- 5672, 5671: used by AMQP 0-9-1 and 1.0 clients without and with TLS
- 25672: used for inter-node and CLI tools communication (Erlang distribution server port) and is allocated from a dynamic range (limited to a single port by default, computed as AMQP port + 20000). Unless external connections on these ports are really necessary (e.g. the cluster uses federation or CLI tools are used on machines outside the subnet), these ports should not be publicly exposed. See networking guide for details.
- 35672-35682: used by CLI tools (Erlang distribution client ports) for communication with nodes and is allocated from a dynamic range (computed as server distribution port + 10000 through server distribution port + 10010). See networking guide for details.
- 15672: HTTP API clients, management UI and rabbitmqadmin (only if the management plugin is enabled)
- 61613, 61614: STOMP clients without and with TLS (only if the STOMP plugin is enabled)
- 1883, 8883: (MQTT clients without and with TLS, if the MQTT plugin is enabled
- 15674: STOMP-over-WebSockets clients (only if the Web STOMP plugin is enabled)
- 15675: MQTT-over-WebSockets clients (only if the Web MQTT plugin is enabled)
- 15692: Prometheus metrics (only if the Prometheus plugin is enabled)

https://www.rabbitmq.com/networking.html#ports

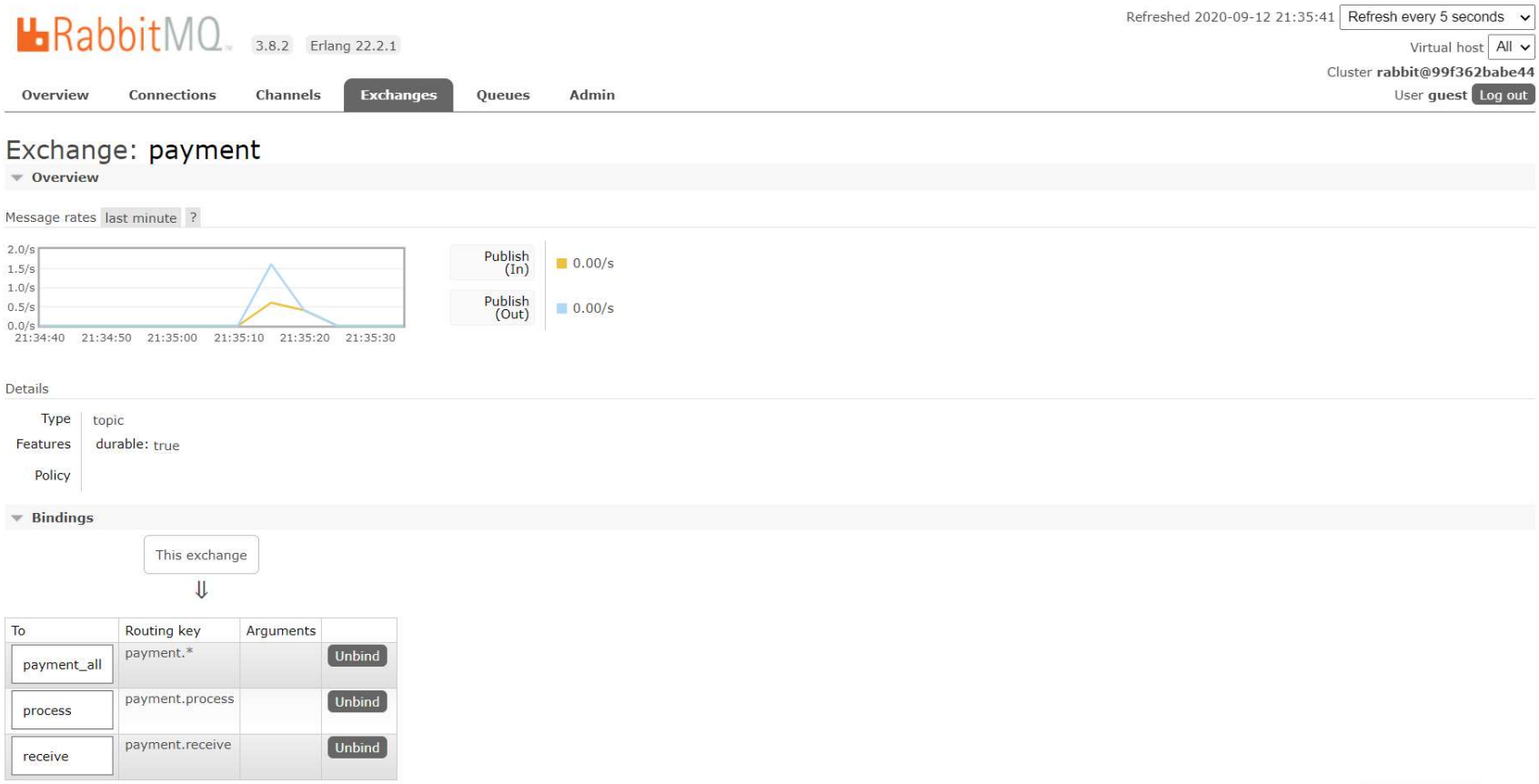
RabbitMQ Management Portal



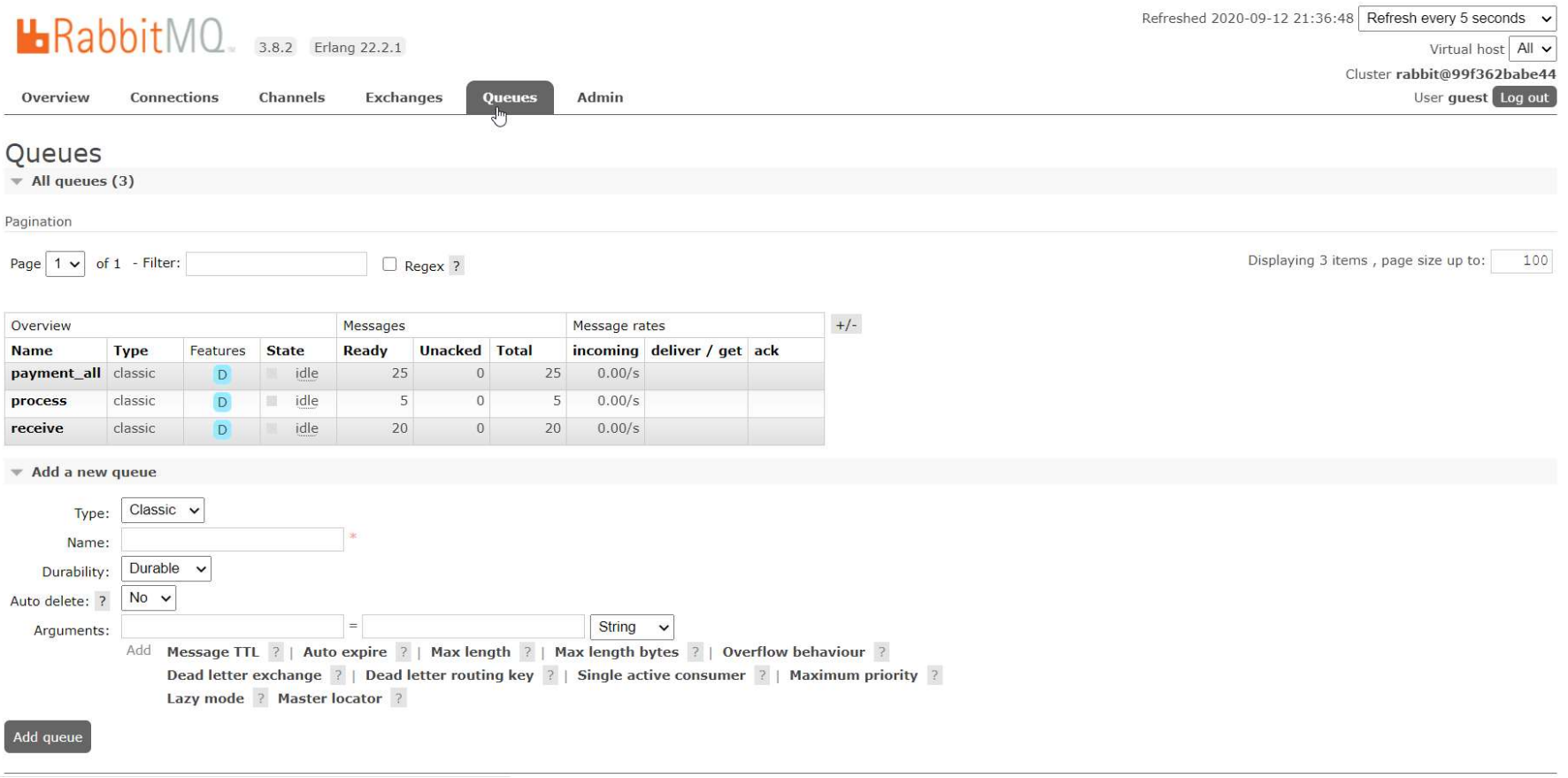
RabbitMQ Management Portal



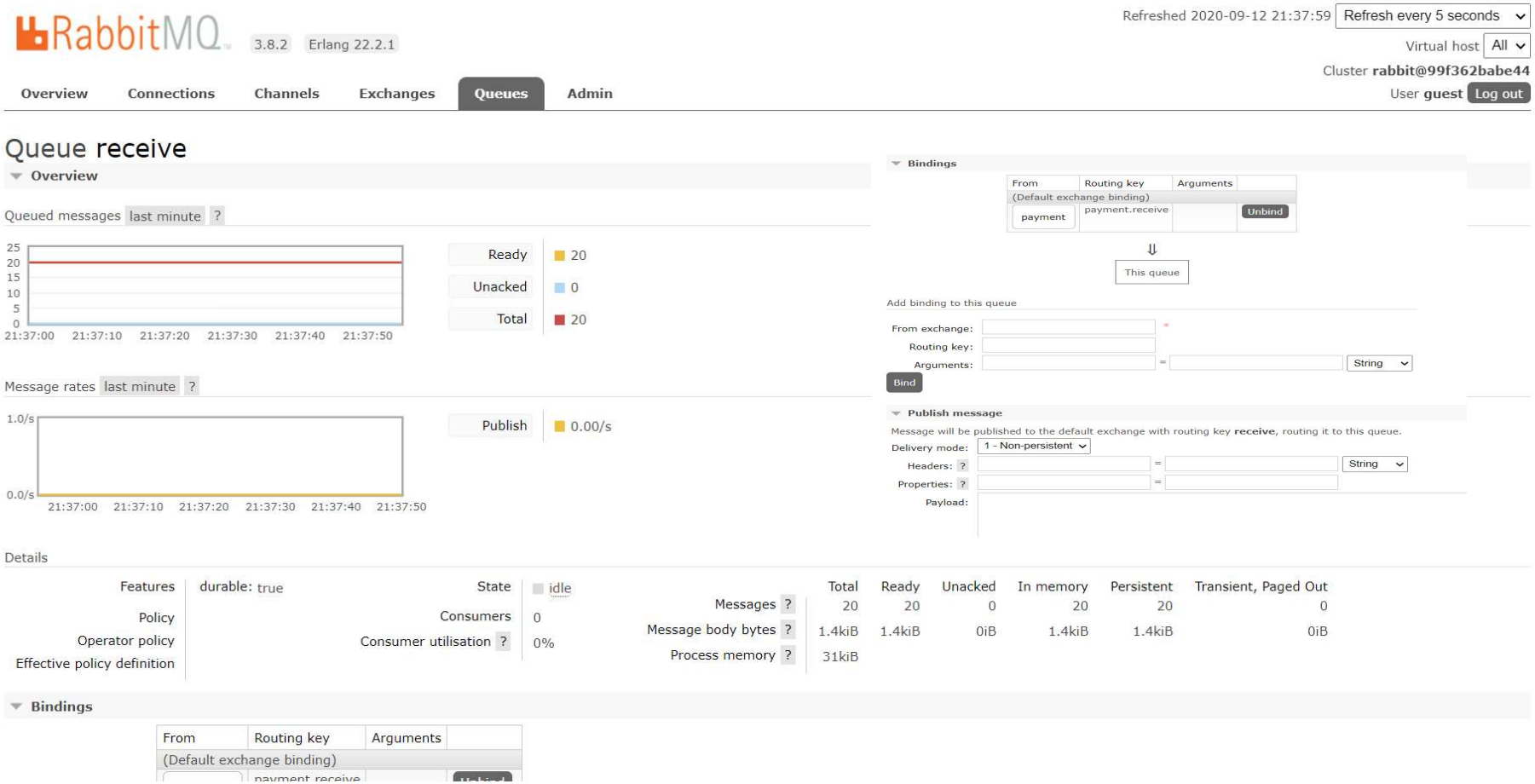
RabbitMQ Management Portal



RabbitMQ Management Portal



RabbitMQ Management Portal



Demo

RabbitMQ management portal

Exchanges

Queues

Publish and subscribe

Administration

Docker container

Summary

- Introduce the RabbitMQ platform
- RabbitMQ management portal
- AMQP Protocol
- Exchanges
- Queues, bindings and consumers
- Client support
- Docker setup
- Demo

Thanks!