Contents

[1. Introduction 2](#_Toc143855947)

[**2.** Installation 2](#_Toc143855948)

[3. Configuration 2](#_Toc143855949)

[4. Debugging 2](#_Toc143855950)

[5. Basic Operations 2](#_Toc143855951)

[6. Advanced Features 2](#_Toc143855952)

[7. Deploying as a Docker Container 2](#_Toc143855953)

[8. Conclusion: 2](#_Toc143855954)

# Introduction

RabbitMQ, a robust and versatile message broker, plays a pivotal role in modern software architectures by enabling efficient communication between distributed application components. This document provides a concise guide to installing, configuring, and utilizing RabbitMQ in a Linux environment and deploying it as a Docker container.

Purpose of RabbitMQ

In distributed systems, where applications are composed of interconnected services, RabbitMQ is a key facilitator of seamless data exchange. It acts as an intermediary, allowing producers to send messages without requiring direct knowledge of consumers. This decoupling enhances flexibility, scalability, and fault tolerance.

Advantages

The significance of RabbitMQ lies in its ability to streamline communication by:

* Decoupling: Components can interact indirectly, reducing dependencies and promoting system flexibility.
* Scalability: RabbitMQ efficiently manages heavy workloads by distributing messages to consumers.
* Reliability: Messages are persistently stored, ensuring delivery even if a consumer or server temporarily goes offline.
* Message Routing: Its exchange and binding mechanisms enable intelligent message routing based on criteria.
* Protocol Support: RabbitMQ supports various messaging protocols, fostering compatibility.
* Extensibility: Plugins extend its features, providing functionalities like message prioritization.

# Installation

Pull the RabbitMQ Docker Image:

Open a terminal on your Ubuntu server and run the following command to pull the official RabbitMQ Docker image:

docker pull rabbitmq

Run RabbitMQ Container:

Once the image is downloaded, you can create and run a RabbitMQ container. Use the following command to run the RabbitMQ container while exposing the necessary ports:

docker run -d --name rabbitmq-container -p 5672:5672 -p 15672:15672 rabbitmq

This command does the following:

* + -d: Runs the container in detached mode.
  + --name rabbitmq-container: Assigns the name "rabbitmq-container" to the container.
  + -p 5672:5672 -p 15672:15672: Maps the container's ports 5672 (AMQP) and 15672 (RabbitMQ management) to the corresponding ports on your host machine.

1. Access RabbitMQ Management Console:

RabbitMQ provides a management console for easy monitoring and configuration. Open your web browser and navigate to http://your\_server\_ip:15672/ (replace your\_server\_ip with your Ubuntu server's IP address). You can log in using the default credentials:

* + Username: guest
  + Password: guest

Upon successful login, you can explore and manage your RabbitMQ instance through the web interface.

Or

Use docker-compose.yml file

services:

rabbitmq:

image: rabbitmq:3.12.3-management-alpine

container\_name: rabbitmq

ports:

- "5672:5672"

- "15672:15672"

env\_file:

- RABBITMQ\_DEFAULT\_USER=your\_user

- RABBITMQ\_DEFAULT\_PASS=your\_password

volumes:

- rabbitmq-data:/var/lib/rabbitmq

- /logs:/var/log/rabbitmq

volumes:

rabbitmq-data:

# Configuration

Docker Environment Variables

When running RabbitMQ as a Docker container, you can use environment variables to configure its behavior. Here are a few commonly used environment variables:

* RABBITMQ\_DEFAULT\_USER:
* RABBITMQ\_DEFAULT\_PASS:
* RABBITMQ\_DEFAULT\_VHOST:
* RABBITMQ\_ERLANG\_COOKIE: Set the Erlang cookie for clustering nodes.
* RABBITMQ\_MNESIA\_BASE: Configure the Mnesia database directory.

You can pass these variables when running the RabbitMQ container to tailor its behavior to your requirements.

By default, the RabbitMQ management interface is protected with a username and password. The default credentials are as follows:

* Username: guest
* Password: guest

1. Changing the Default Password (Recommended): To change the password for the default guest user, you can use the following command:

docker exec -it rabbitmq rabbitmqctl change\_password guest NEW\_PASSWORD

1. Adding a New User: To add a new user with specific credentials

docker exec -it rabbitmq rabbitmqctl add\_user USERNAME PASSWORD

Granting Permissions: After adding a new user, you need to grant appropriate permissions.

docker exec -it rabbitmq rabbitmqctl set\_user\_tags USERNAME administrator

Remember to update your .env file or Docker Compose configuration with the new credentials if you've changed the default username and password.

Restart container

# Debugging

<http://rabbitmq.uc.dev.tetherfi.cloud:15672> use this url and check web interface

or

* To check the status of RabbitMQ nodes:

docker exec -it rabbitmq-container rabbitmqctl status

* To list queues:

docker exec -it rabbitmq-container rabbitmqctl list\_queues

* To list connections:

docker exec -it rabbitmq-container rabbitmqctl list\_connections

# Basic Operations

Using the RabbitMQ Management Interface:

1. Open your web browser and navigate to http://rabbitmq.uc.dev.tetherfi.cloud (replace with your actual URL).
2. Log in with your RabbitMQ administrative credentials.
3. Navigate to the "Queues" tab.
4. Click the "Add a new queue" button.
5. Provide a name for the queue and any desired settings.
6. Click "Add queue" to create the new queue.

Using Command Line:

To add a new queue named "my\_queue" using the command line:

docker exec -it rabbitmq-container rabbitmqadmin declare queue name=my\_queue durable=true

Publishing Messages

Using the RabbitMQ Management Interface:

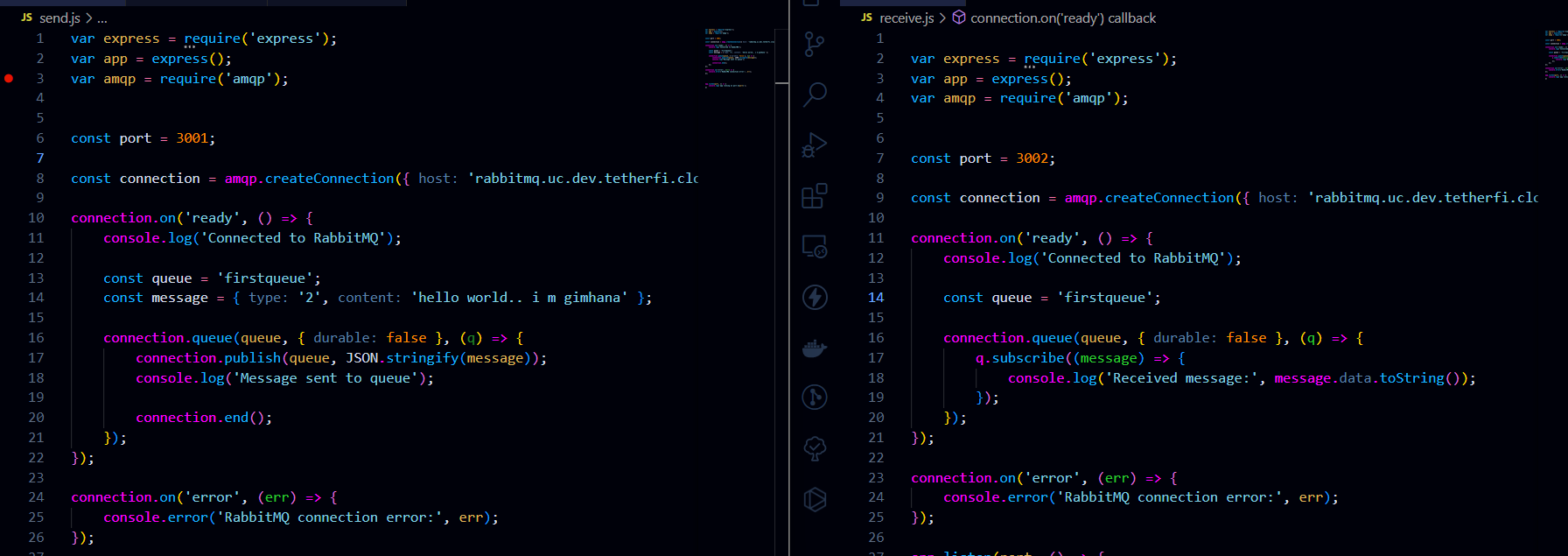
1. Open the RabbitMQ Management Interface.
2. Navigate to the "Publish message" section within your queue.
3. Enter the message content and any desired message properties.
4. Click "Publish message" to send the message to the queue.

Using Command Line:

To publish a message to the "my\_queue" using the command line:

**docker exec -it** rabbitmq-container rabbitmqadmin publish exchange= routing\_key=my\_queue payload="Hello, RabbitMQ!"

Subscribing to a Queue, then user allows to send and receive messages



# Advanced Features

**Exchanges and Bindings:** Exchanges are routing mechanisms that determine how messages should be distributed to queues. Bindings connect exchanges to queues. Different types of exchanges include:

* **Direct Exchange:** Routes messages to queues based on a routing key.
* **Fanout Exchange:** Broadcasts messages to all queues bound to it.
* **Topic Exchange:** Routes messages to queues based on patterns in the routing key.
* **Headers Exchange:** Routes messages based on message headers.

**Message Acknowledgment:** Message acknowledgment is crucial to ensure reliable message processing. A message is only removed from the queue when the consumer acknowledges its receipt. If a consumer fails to process a message, it can either reject it (which sends it back to the queue) or requeue it for later processing.

**Dead Letter Exchanges:** Dead letter exchanges provide a way to handle messages that couldn't be processed. If a message repeatedly fails to be processed, it can be sent to a dead letter queue, allowing you to inspect and handle problematic messages separately.

# Deploying as a Docker Container

Use docker-compose.yml file

services:

rabbitmq:

image: rabbitmq:3.12.3-management-alpine

container\_name: rabbitmq

ports:

- "5672:5672"

- "15672:15672"

env\_file:

- RABBITMQ\_DEFAULT\_USER=your\_user

- RABBITMQ\_DEFAULT\_PASS=your\_password

volumes:

- rabbitmq-data:/var/lib/rabbitmq

- /logs:/var/log/rabbitmq

volumes:

rabbitmq-data:

or RabbitMQ management Studio

# Conclusion: