

RabbitMQ

Fachbereichs Wirtschaft der
Technischen Hochschule Brandenburg
Master of Science

vorgelegt von:
Franck Derrick Kateu Fonga Lekeufack
2. Semester

Betreuer: Prof. Dr. rer. nat. Windfried Pfister

Brandenburg/H., den 21. Juni 2023

1 Inhaltsverzeichnis

1	Einleitung	1
2	Core Concept und Definitionen	2
3	Architektur von RabbitMQ	5
4	Messaging-Muster	7
4.1	Point-to-Point-Modell	7
4.2	Publish-Subscribe-Modell	8
4.3	Request-Response	9
4.4	Routing-Modell	10
5	Anwendungsgebiete von Rabbit MQ	12
6	Installation, Sicherheit und Konfiguration	13
6.1	Installationen	13
6.2	Sicherheitsaspekte	20
6.3	Konfigurationen	20
7	Anwendung Szenarien von Rabbit MQ in der Praxis.....	24
8	Schlussteil.....	26
9	Literaturverzeichnis.....	27

Abbildungsverzeichnis

Abbildung 1: RabbitMQ Teil 1.....	5
Abbildung 2: RabbitMQ Teil 2.....	6
Abbildung 3: Point-to-Point Modell	8
Abbildung 4: Publiisch Subscriber Modell	9
Abbildung 5: Request/Response Modell	10
Abbildung 6: Anwendungsgebiete von RabbitMQ	12
Abbildung 7: WSL Installation.....	13
Abbildung 8: Docker Installation	14
Abbildung 9: Docker GUI	14
Abbildung 10: RabbitMQ Installation	15
Abbildung 11: RabbitMQ Server	15
Abbildung 12: Webanwendung von RabbitMQ.....	16
Abbildung 13: Zertifikat und Schlüssel	17

Abstrakt

RabbitMQ is a popular open-source message broker software that facilitates communication between distributed systems. It is widely used in microservices architecture, where different services need to communicate with each other in a reliable and scalable manner. RabbitMQ supports various messaging patterns such as point-to-point, publish-subscribe, and request-reply, and it provides a flexible and extensible architecture to handle different use cases.

This paper provides an overview of RabbitMQ and its features, including the AMQP (Advanced Message Queuing Protocol) standard, message persistence, routing, and clustering. It also discusses the benefits of using RabbitMQ in a distributed system, such as decoupling services, improving fault tolerance, and enabling horizontal scalability. Furthermore, the paper explores some of the best practices for designing and deploying RabbitMQ-based applications, such as optimizing message size, handling dead letter queues, and monitoring the system (Nguyen, 2023).

In conclusion, RabbitMQ is a powerful and versatile message broker software that can simplify the communication between distributed systems. It provides a flexible and scalable architecture that can handle a wide range of messaging patterns and use cases. By following best practices and leveraging its features, RabbitMQ can help developers build reliable and efficient distributed systems.

1 Einleitung

RabbitMQ ist eine Open-Source-Messaging-Middleware, die zur Implementierung von Nachrichtenwarteschlangen (Message Queues) verwendet wird. Es handelt sich um eine zuverlässige und flexible Plattform, die es Entwicklern ermöglicht, verschiedene Anwendungen nahtlos miteinander zu verbinden, indem sie Nachrichten untereinander austauschen (RabbitMQ, o.J.). In der heutigen, zunehmend vernetzten und verteilten Welt ist die Kommunikation zwischen Anwendungen und Systemen von entscheidender Bedeutung. RabbitMQ bietet eine Lösung für die Herausforderungen der Integration verteilter Systeme, indem es eine zuverlässige und skalierbare Möglichkeit bietet, Nachrichten zwischen verschiedenen Komponenten zu übermitteln. Das Herzstück von RabbitMQ sind Nachrichtenwarteschlangen, in denen Nachrichten vorübergehend gespeichert werden, bis sie von einem Empfänger verarbeitet werden (RabbitMQ, o.J.). Dies ermöglicht eine lose Kopplung zwischen den Komponenten, da der Absender einer Nachricht nicht wissen muss, wer sie empfangen wird oder wann dies geschieht. Stattdessen schickt der Absender die Nachricht einfach an die Warteschlange und RabbitMQ sorgt dafür, dass sie an den richtigen Empfänger weitergeleitet wird. RabbitMQ unterstützt verschiedene Nachrichtenprotokolle wie das Advanced Message Queuing Protocol (AMQP), das Message Queuing Telemetry Transport Protocol (MQTT) und das Streaming Text Oriented Messaging Protocol (STOMP). Dadurch können Entwickler die für ihre Anwendung am besten geeignete Option wählen und die Interoperabilität zwischen verschiedenen Systemen gewährleisten. Ein weiterer Vorteil von RabbitMQ ist seine Fähigkeit, die Last zu verteilen (Load Balancing). Durch die Verwendung von Nachrichtenwarteschlangen kann RabbitMQ den Nachrichtenstrom intelligent auf mehrere Empfänger verteilen und so die Auslastung des Systems optimieren. Dadurch wird die Skalierbarkeit verbessert und Engpässe werden vermieden. Zusammenfassend lässt sich sagen, dass RabbitMQ eine leistungsstarke Nachrichten-Middleware ist, die Entwicklern hilft, verteilte Systeme zu integrieren und effizient zu kommunizieren. Mit Nachrichtenwarteschlangen, Protokollunterstützung und Lastverteilungsfunktionen bietet RabbitMQ eine solide Grundlage für die Entwicklung robuster und skalierbarer Anwendungen in einer vernetzten Welt.

2 Core Concept und Definitionen

Das sind die grundlegenden Konzepte oder Prinzipien, aus denen eine bestimmte Technologie, Methode oder ein System besteht und die ihre Funktionsweise bestimmen. Im Fall von RabbitMQ sind die Grundkonzepte die grundlegenden Ideen und Komponenten, die das Design und die Funktionsweise von RabbitMQ ausmachen und es zu einem effektiven Message Broker machen. Ein gründliches Verständnis dieser Grundkonzepte ist wichtig, um die Verwendung von RabbitMQ in verschiedenen Anwendungsfällen zu verstehen und um Anwendungen zu entwickeln, die mit RabbitMQ funktionieren. Zu diesen Konzepten gehören unter anderem:

In RabbitMQ ist ein **Produzent oder Producers** eine Anwendung oder ein Prozess, der Nachrichten zur weiteren Verarbeitung an einen Nachrichtenagenten sendet. Mit anderen Worten, ein Producer ist für die Veröffentlichung von Nachrichten an einen bestimmten Exchange in RabbitMQ verantwortlich.

Wenn ein Producer eine Nachricht an einen Exchange sendet, leitet RabbitMQ die Nachricht basierend auf dem Exchange-Typ und dem Routing-Schlüssel an die entsprechende(n) Warteschlange(n) weiter. Die Warteschlangen werden dann von den Verbrauchern, die sie abonniert haben, konsumiert.

Produzenten sind eine wesentliche Komponente eines Nachrichtensystems und ermöglichen es Anwendungen, Nachrichten an einen Broker zu veröffentlichen, der sie dann zur weiteren Verarbeitung an die Verbraucher weiterleitet (AMQP, o.J.).

In RabbitMQ ist ein **Konsument oder Consumers** jede Anwendung oder jeder Prozess, der Nachrichten von einem Nachrichtenagenten zur weiteren Verarbeitung erhält. Ein Konsument abonniert sich in eine bestimmte Warteschlange (Queue), während er auf das Eintreffen von Nachrichten wartet. Wenn eine Nachricht in der Warteschlange veröffentlicht wird, liefert RabbitMQ die Nachricht an einen der verfügbaren Consumer aus. Wenn die Nachricht nicht verarbeitet werden kann, kann der Consumer die Nachricht verwerfen oder sie wieder in die Warteschlange stellen (RabbitMQ, o.J.).

Verbraucher sind ein wesentlicher Bestandteil eines Nachrichtensystems und ermöglichen es Anwendungen, Nachrichten von einem Broker zu empfangen, sie zu verarbeiten und bei dem Empfang von Nachrichten von Consumer, verarbeitet

er die Nachricht und sendet eine Bestätigung an RabbitMQ zurück, um zu bestätigen, dass die Nachricht erfolgreich verarbeitet wurde (CloudAMQP, O.J)

Ein Broker ist eine Serveranwendung, die Nachrichten von Produzenten (sendende Anwendungen) empfängt und an Konsumenten (empfangende Anwendungen) weiterleitet. Der Broker fungiert als Vermittler zwischen Produzenten und Konsumenten und ermöglicht es Anwendungen, miteinander zu kommunizieren, ohne direkt miteinander verbunden zu sein.

Bei Zusendung eine Nachricht an RabbitMQ, übernimmt der Broker die Aufgabe, die Nachricht zu verarbeiten und an den richtigen Konsumenten weiterzuleiten. Der Broker verwaltet die Warteschlangen (Queues) und die Routingregeln, um sicherzustellen, dass die Nachrichten an die richtigen Verbraucher gesendet werden. Der Broker spielt eine wichtige Rolle in einem Nachrichtensystem und ist ein unverzichtbares Element, das eine zuverlässige und effiziente Kommunikation zwischen verschiedenen Anwendungen und Systemen ermöglicht (RabbitMQ, o.J).

Eine Warteschlange ist eine Standleitung oder ein Puffer, in dem Nachrichten gespeichert werden, die von Produzenten (sendenden Anwendungen) gesendet wurden und auf die Verarbeitung durch einen oder mehrere Konsumenten (empfangende Anwendungen) warten. Wenn ein Produzent eine Nachricht an RabbitMQ sendet, wird die Nachricht an einen Exchange weitergeleitet. Der Exchange führt dann ein Routing auf der Grundlage des Exchange-Typs und des Routing-Schlüssels der Nachricht durch und leitet die Nachricht an eine oder mehrere Warteschlangen weiter.

Eine Warteschlange ist eine Linie, die Nachrichten in der Reihenfolge speichert, in der sie angekommen sind. Die Verbraucher können sich dann mit der Warteschlange verbinden und die Nachrichten aus der Warteschlange abrufen, um sie zu verarbeiten. Wenn die Nachricht bei einem Verbraucher eingegangen ist, wird sie aus der Warteschlange entfernt, um sicherzustellen, dass sie nicht mehrfach verarbeitet wird.

Ein Exchange ist ein Vermittler zwischen den Produzenten (sendenden Anwendungen) und den Warteschlangen. Er ist dafür verantwortlich, dass die Nachrichten an die richtige Warteschlange verteilt werden, basierend auf dem Routingschlüssel der Nachricht und der vom Exchange implementierten Routinglogik. Wenn ein Produzent eine Nachricht an RabbitMQ sendet, sendet er sie an einen Exchange. Der Exchange führt eine Routinglogik aus, um anhand des Routingschlüssels der Nachricht und der Art des Exchanges zu bestimmen, an welche Warteschlange die Nachricht weitergeleitet werden soll.

Das Konzept von RabbitMQ bringt viele Vorteile mit sich (Dipl.-Ing. Luber, 2021):

- Software frei verfügbar unter Open-Source-Lizenz
- verfügbar für alle gängigen Betriebssysteme
- funktioniert mit allen gängigen Programmiersprachen
- geeignet für Vor-Ort-Installationen, verteilte Umgebungen und Cloud-Computing
- anpassbar über Plug-Ins
- unterstützt verschiedene Messaging-Protokolle wie AMQP, STOMP oder MQTT
- schlanke, stabile und einfach zu installierende Anwendung
- grafische Benutzeroberfläche über Plug-in verfügbar
- beliebte und große Community im Internet
- professioneller Support verfügbar
- einfache Skalierbarkeit
- hochverfügbare Installation möglich
- Entkopplung von Producer- und Consumer-Prozessen durch asynchrones Messaging
- auch für die Zustellung an mehrere Empfänger und für Broadcasts geeignet
- unterstützt Verschlüsselung und Authentifizierung

3 Architektur von RabbitMQ

RabbitMQ ist eine Open-Source-Nachrichtenvermittlungssoftware, die eine flexible und robuste Architektur aufweist, um die zuverlässige Übertragung von Nachrichten zwischen verschiedenen Systemen und Anwendungen zu ermöglichen. Die Architektur von RabbitMQ besteht aus mehreren Elementen, die zusammenarbeiten, um die Übertragung von Nachrichten zu ermöglichen. Neben den Hauptkomponenten Producer, Consumers, Exchanges und Queues gibt es noch folgende Komponenten

Virtueller Host: Eine logische Gruppierung von Exchanges, Warteschlangen und Berechtigungen. Jeder virtuelle Host kann seine eigenen Benutzer und Berechtigungen haben, um die Sicherheit und Isolierung von Nachrichten zu gewährleisten. **Channel (Kanal):** Ein Kanal, der die Kommunikation zwischen Anwendungen und dem RabbitMQ-Server ermöglicht. Jeder Kanal ist eine separate Verbindung zur RabbitMQ-Instanz, sodass mehrere Nachrichtenströme über eine einzige Verbindung verwaltet werden können. Durch die Kombination dieser Komponenten ermöglicht RabbitMQ eine zuverlässige und flexible Übertragung von Nachrichten zwischen Anwendungen und Systemen.

Die folgenden Darstellungen macht deutlich, wie Nachrichten zwischen den architektonischen Komponenten von RabbitMQ weitergeleitet werden. Es ist deutlich zu sehen, wie die Nachrichten durch die Kategorien am Autobahnkreuz weitergeleitet werden (CloudAMQP, O.J).

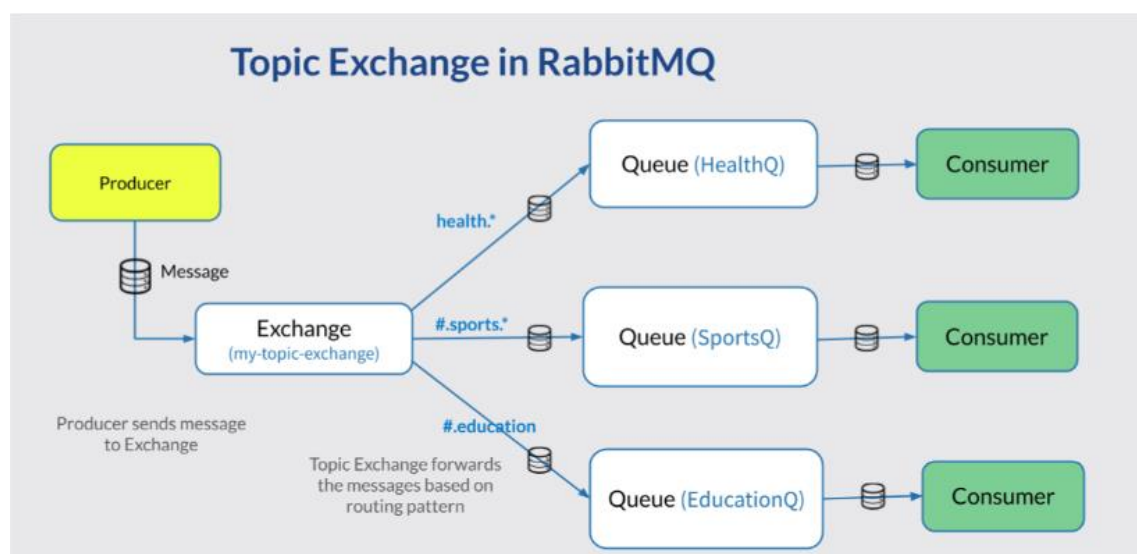


Abbildung 1: RabbitMQ Teil 1

Quelle: (Bikraume, 2020)

Eine andere Forme der Architektur von RabbitMQ ist in folgende Darstellung deutlich zu sehen.

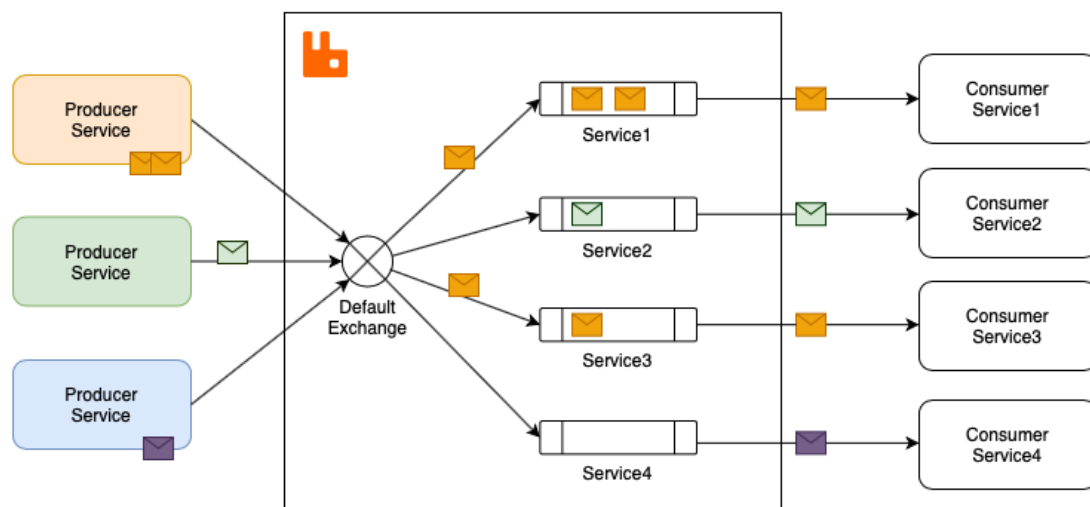


Abbildung 2: RabbitMQ Teil 2

Quelle: (Olah, 2020)

4 Messaging-Muster

Es gibt in RabbitMQ vier grundlegende Nachricht-Muster, die verwendet werden können. Die Art und Weise wie die Nachrichten vom Sender zu Empfänger weitergeleitet sind, spielt eine große Rolle.

4.1 Point-to-Point-Modell

Das Point-to-Point-Messaging (P2P) ist ein Modell zur Nachrichtenübertragung, das eine direkte Kommunikation zwischen einem Sender und einem Empfänger ermöglicht. In diesem Fall sendet der Sender eine Nachricht an eine bestimmte Warteschlange (Queue) und der Empfänger liest die Nachricht in der Warteschlange (RabbitMQ, o.J.). Beim P2P-Modell können mehrere Empfänger auf eine Warteschlange zugreifen, aber nur einer von ihnen erhält die Nachricht. Jede Nachricht wird nur von einem Empfänger verarbeitet. Wenn es mehrere Empfänger gibt, wird die Nachricht an den Empfänger mit der höchsten Priorität gesendet, der die Nachricht als Erster aus der Warteschlange abrufen kann (Gregor & Bobby, o.J.).

Außerdem eignet sich das P2P-Modell gut für die Verarbeitung von Aufgaben, bei denen jeder Empfänger eine bestimmte Aufgabe ausführt, z. B. die Bearbeitung von Bestellungen oder Aufträgen. Es bietet eine hohe Zuverlässigkeit, da jede Nachricht nur einmal an einen Empfänger gesendet wird und nicht verloren geht. Die Implementierung der P2P in RabbitMQ erfordert die Einrichtung einer Warteschlange, an die der Absender die Nachrichten sendet, und die Einrichtung von Empfängern, die die Nachrichten aus der Warteschlange abholen. RabbitMQ stellt eine API zur Verfügung, die es Entwicklern ermöglicht, Warteschlangen zu erstellen und Nachrichten zu senden und zu empfangen.

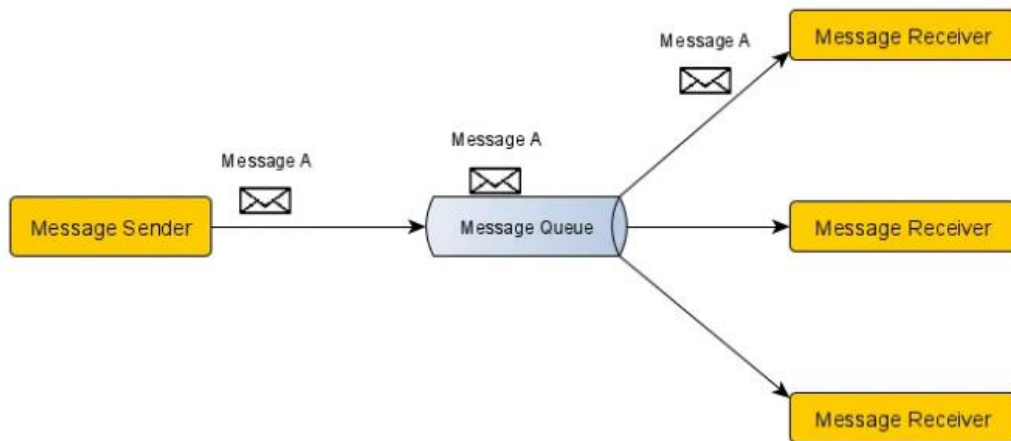


Abbildung 3: Point-to-Point Modell

Quelle: (Tran, 2020)

4.2 Publish-Subscribe-Modell

Publish/Subscribe Messaging, auch Pub/Sub Messaging genannt, ist eine Form des asynchronen Messagings. In diesem Bereich werden die Produzenten (Producers) von Nachrichten als Publishers) und die Konsumenten von Nachrichten als Abonnenten bezeichnet. Der Publisher produziert Nachrichten für ein Thema, und alle Abonnenten, die dieses Thema abonniert haben, empfangen die gesendeten Nachrichten und konsumieren sie. Es besteht einen Unterschied zwischen dem Point-to-Point-Messaging-Modell und dem Publish/Subscribe-Messaging-Modell, dieser liegt bei der Anzahl der Empfänger einer Nachricht. Ein weiterer Unterschied besteht darin, dass beim Point-to-Point-Modell der Absender der Nachricht den Empfänger kennen muss, während beim Publish/Subscribe-Modell der Herausgeber der Nachricht nicht wissen muss, wo die Nachricht konsumiert werden wird. Dieses Merkmal bietet eine erhebliche Entkopplung für die Anwendung bei der Anwendung des Publish/Subscribe-Nachrichtenmodells (Tran, 2020).

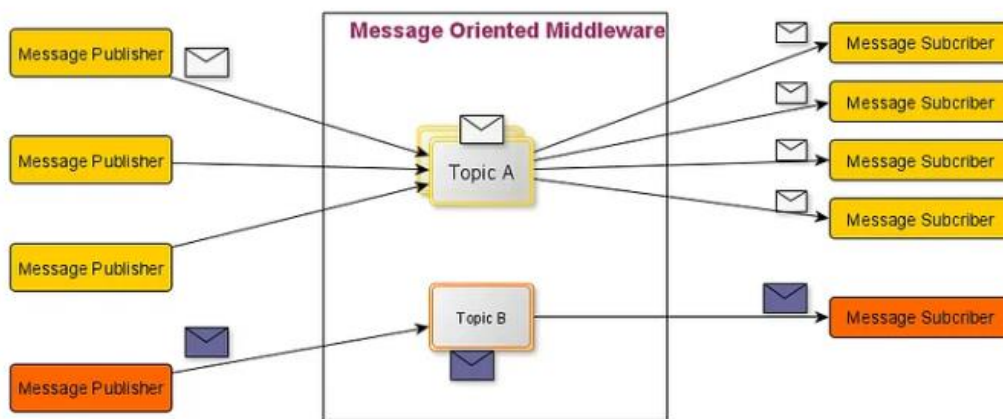


Abbildung 4: PubliSch Subscriber Modell

Quelle: (Tran, 2020)

4.3 Request-Response

Das Request/Resp-Messaging-Modell (Req/Resp) ist ein Modell zur Nachrichtenübertragung, das eine synchronisierte Kommunikation zwischen einem Sender und einem Empfänger ermöglicht. Beim Req/Resp-Modell sendet der Requestor eine Request-Message an eine Warteschlange (Queue) und der Responder (Replier) antwortet mit einer Response-Message.

Im Gegensatz zu den Modellen Publish/Subscriber (Pub/Sub) und Point-to-Point (P2P), bei denen Nachrichten asynchron gesendet und empfangen werden können, muss der Empfänger beim Modell Req/Resp die Anforderungsnachricht empfangen, sie verarbeiten und eine Antwortnachricht an den Sender zurückschicken (Gregor & Bobby, o.J.). Der Sender muss die Antwort abwarten, bevor er weitere Aufgaben ausführt. Das Req/Resp-Modell eignet sich gut für Anwendungsfälle, in denen eine synchronisierte Kommunikation erforderlich ist, z. B. bei der Abfrage von Informationen oder der Ausführung von Methoden aus der Ferne. Es bietet hohe Sicherheit und Zuverlässigkeit, da die Kommunikation zwischen Sender und Empfänger eng synchronisiert ist und die Verarbeitung der Nachrichten gesteuert werden kann (David, 2014).

Die Implementierung von Req/Resp in RabbitMQ erfordert die Einrichtung einer Warteschlange, in die der Sender die Abfragenachrichten sendet, und die Einrichtung von Empfängern, die die Nachrichten aus der Warteschlange

abrufen und darauf antworten können. RabbitMQ bietet eine Vielzahl von Routing-Optionen und Nachrichtenvorlagen, um Req/Resp zu implementieren.

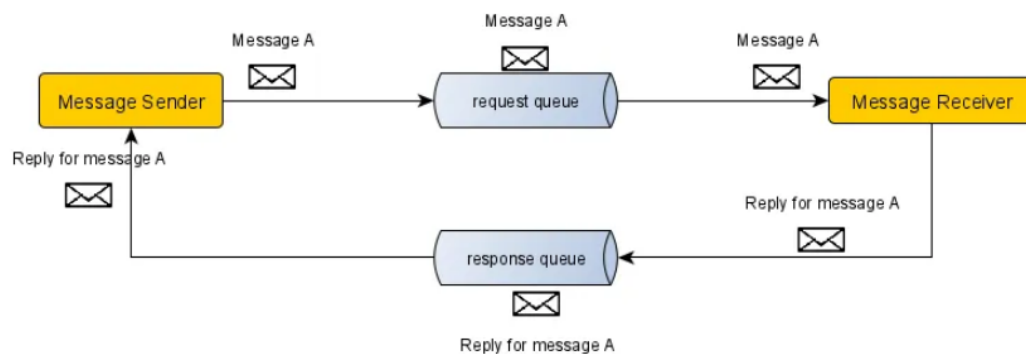


Abbildung 5: Request/Response Modell

Quelle: (Tran, 2020)

4.4 Routing-Modell

Das Routing-Modell ist ein erweitertes Nachrichtenmodell in RabbitMQ, das eine selektive Weiterleitung von Nachrichten an bestimmte Warteschlangen auf der Grundlage von Routing-Schlüsseln ermöglicht. In diesem Modell gibt es einen Exchange, der die Nachrichten von den Absendern empfängt und sie auf der Grundlage der in den Nachrichten enthaltenen Routingschlüssel an bestimmte Warteschlangen weiterleitet.

Das Routingmodell wird häufig verwendet, um Nachrichten anhand bestimmter Kriterien wie Typ, Region oder Sprache an bestimmte Empfänger oder Empfängergruppen weiterzuleiten.

Das Routing-Modell von RabbitMQ wird durch die Verwendung von Exchanges umgesetzt, die Nachrichten auf der Grundlage von Routing-Schlüsseln an bestimmte Warteschlangen weiterleiten.

Dieses Modell kann 3 Arten von Exchanges benutzen. Unter anderen zählen:

- Direct Exchange
- Fanout Exchange
- Topic Exchange
- Headers Exchange

Dies werden im nächsten Kapitel näher erläutert.

Das Routing bietet Flexibilität und Skalierbarkeit, da es die Möglichkeit bietet, bestimmte Nachrichten an bestimmte Empfänger zu senden und die Verarbeitung der Nachrichten zu optimieren.

Bei der Implementierung des Routingmodells in RabbitMQ müssen die Absender Nachrichten an den Exchange senden, der mit dem richtigen Routingschlüssel verknüpft ist. Empfänger müssen bestimmte Warteschlangen erstellen und diese mit dem Exchange verknüpfen, um Nachrichten basierend auf den Routingschlüsseln zu empfangen.

5 Anwendungsgebiete von Rabbit MQ

Wie die Abbildung unten zeigt, sind die Anwendungsbereiche von RabbitMQ und seinen Kommunikationsprotokollen vielfältig und unterschiedlich. Aufgrund seiner Flexibilität bietet es viele Vorteile und wird von Unternehmen aller Art immer mehr geschätzt.

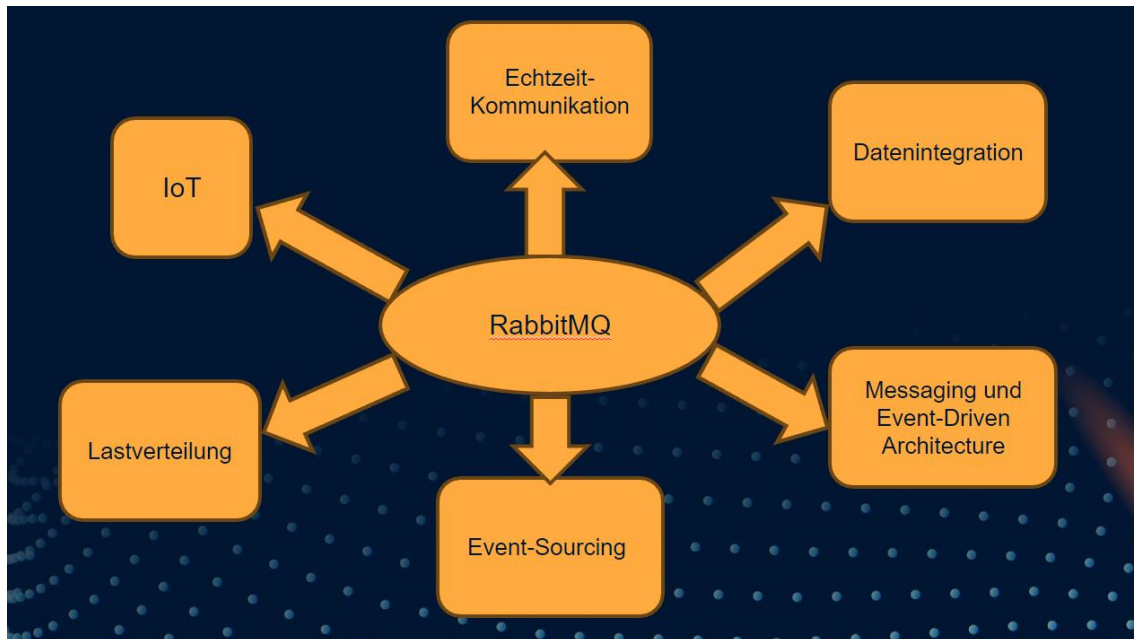


Abbildung 6: Anwendungsgebiete von RabbitMQ

6 Installation, Sicherheit und Konfiguration

6.1 Installationen

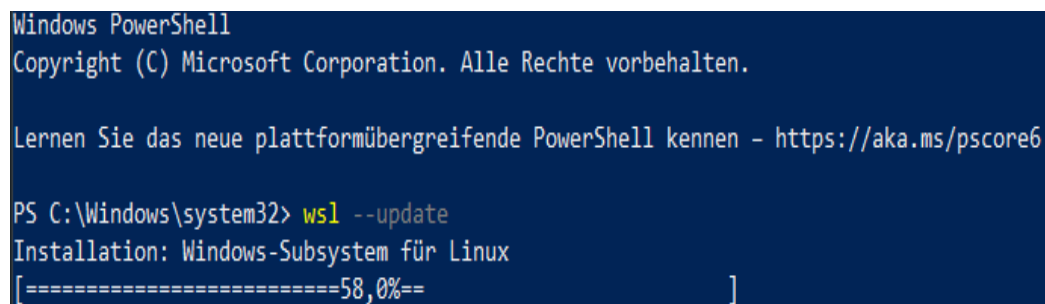
Docker Desktop

Die Verwendung von RabbitMQ als Docker-Container erfordert einige Installationen, um die Umgebung vorzubereiten, in der der Docker und der Container installiert werden.

Zunächst die WSL ist dafür erforderlich. WSL ist ein Windows-Subsystem für Linux. Es ermöglicht Entwicklern, eine Linux-Umgebung (einschließlich der meisten Kommandozeilen-Tools, Dienstprogramme und Anwendungen) direkt unter Windows auszuführen, ohne die zusätzlichen Kosten für einen herkömmlichen virtuellen Computer tragen zu müssen.

Dazu wird die neueste Version von WSL installiert und führt gleichzeitig ein Update mit diesem Update durch.

WSL (Windows Subsystem für Linux)



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

Lernen Sie das neue plattformübergreifende PowerShell kennen - https://aka.ms/pscore6

PS C:\Windows\system32> wsl --update
Installation: Windows-Subsystem für Linux
[=====58,0%==]
```

Abbildung 7: WSL-Installation

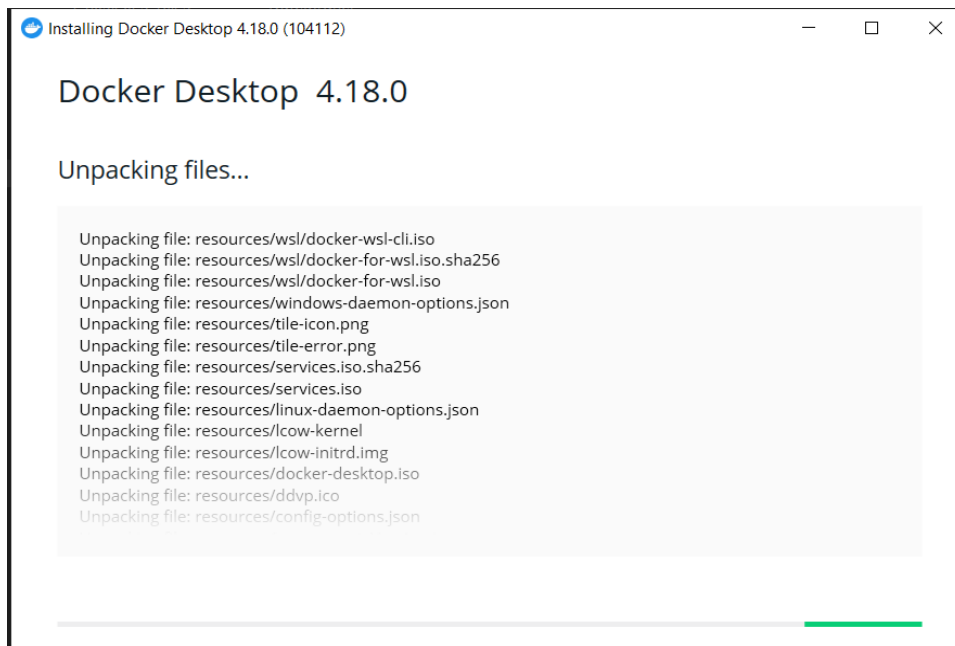


Abbildung 8: Docker Installation

Anschließend wurde auf <https://hub.docker.com/> der Docker Desktop heruntergeladen und installiert und so sieht die bereits installierte Docker aus.

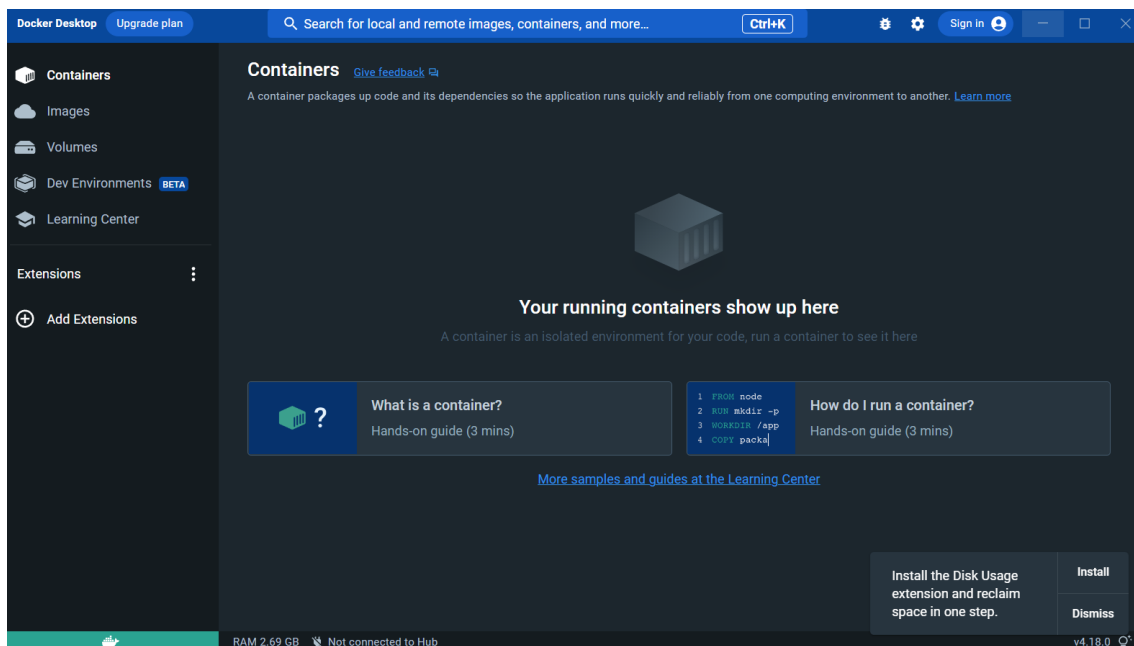


Abbildung 9: Docker GUI

Docker ist eine Implementierung der Containertechnologie, die sich durch besonders benutzerfreundliche Eigenschaften auszeichnet und den Begriff Container als Alternative zu virtuellen Maschinen populär gemacht hat. Ein Container fasst eine einzelne Anwendung mit all ihren Abhängigkeiten wie Bibliotheken, Hilfsprogrammen und statischen Daten in einer Image-Datei

zusammen, ohne jedoch ein vollständiges Betriebssystem zu enthalten. Container können daher mit einer leichten Virtualisierung verglichen werden (Linfeed, 2017).

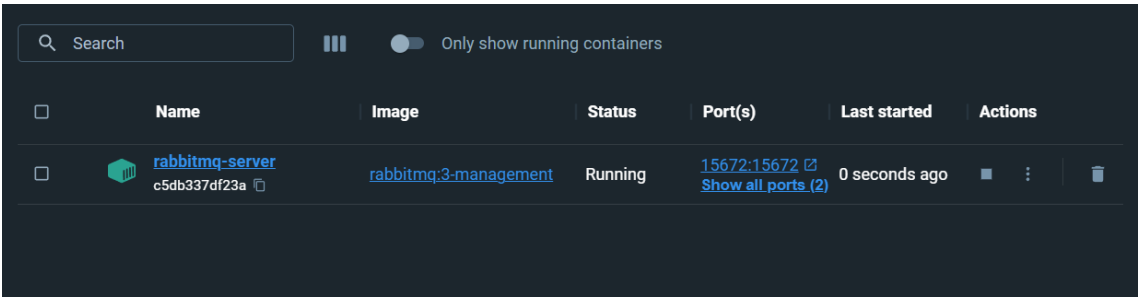
RabbitMQ

Die Installation des Containers erfolgt durch Ausführen des folgenden Befehls in der Befehlszeile.

`docker run -d --hostname rabbitmq --name rabbitmq-server -p 15672:15672 -p 5672:5672 rabbitmq:3-management`

```
PS C:\Users\User> docker run -d --hostname rabbitmq --name rabbitmq-server -p 15672:15672 -p 5672:5672 rabbitmq:3-management
Unable to find image 'rabbitmq:3-management' locally
3-management: Pulling from library/rabbitmq
99803d4b97f3: Pull complete
6ae56123b5f3: Pull complete
1e76166effaa: Pull complete
0f3c583fd97a: Pull complete
84395b4e521d: Pull complete
20155a77fadf: Pull complete
9632567e53a9: Pull complete
1d5f47dc6047: Pull complete
0f30db4e6876: Pull complete
c2e6a9789401: Pull complete
2c59fd913d75: Pull complete
Digest: sha256:93ab23e1ef34c14e4c0a5128fa6525aaaf1b7b078e6607b0926bf6c57e5510dee
Status: Downloaded newer image for rabbitmq:3-management
67a9d1e3447273ca08cd772c82d3d4c3213fa3a3e1f10b860bf9616a4fd1bc7b
PS C:\Users\User>
```

Abbildung 10: RabbitMQ Installation



	Name	Image	Status	Port(s)	Last started	Actions
<input type="checkbox"/>	rabbitmq-server c5db337df23a	rabbitmq:3-management	Running	15672:15672 Show all ports (2)	0 seconds ago	

Abbildung 11: RabbitMQ Server

Wenn der Container installiert ist, kann man unter folgender Adresse auf das Interface von rabbitmq zugreifen: localhost:port (localhost:15672).

Laut Dokumentation sollte das Standardpasswort und der Benutzername "guest" lauten.

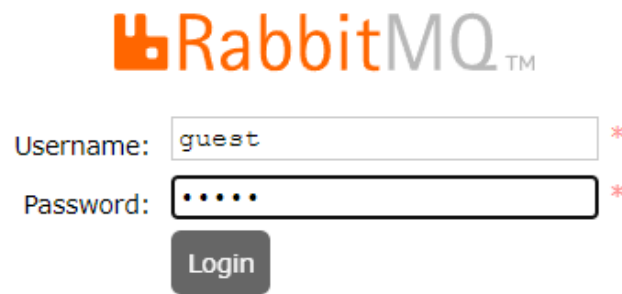


Abbildung 12: Webanwendung von RabbitMQ

OpenSSL & Zertifikat Generieren

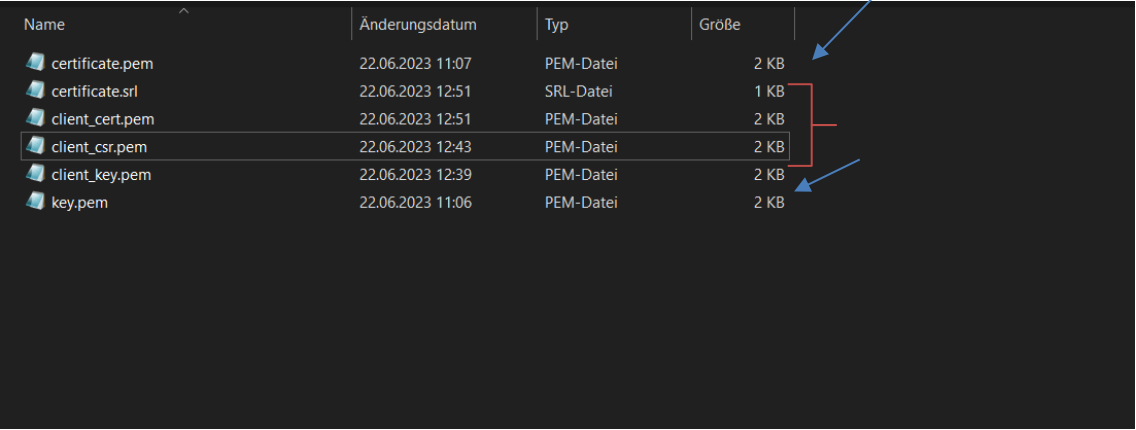
OpenSSL (Open Secure Socket Layer) ist eine Open-Source-Implementierung des Verschlüsselungsprotokolls SSL und seines Nachfolgers TLS (Transport Layer Security). Die Software enthält Werkzeuge zur Erstellung von privaten RSA-Schlüsseln, Prüfsummen und CSR (Certificate Signing Requests). OpenSSL wird auf etwa zwei Dritteln aller Webserver eingesetzt. Das Kommandozeilenprogramm openssl dient dazu, Zertifikate zu beantragen, zu erstellen und zu verwalten. Die Werkzeuge zum Ver- und Entschlüsseln kryptografischer Funktionen werden von der Basisbibliothek bereitgestellt (IT-Wissen, 0.J).

Folgende Befehle wurden für die Installation von OpenSSL benötigt

- `openssl req -newkey rsa:2048 -nodes keyout key.pem -x509 -days 365 -out certificate.pem`

```
-----
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Brandenburg
Locality Name (eg, city) []:Brandenburg an der Havel
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Technische Hochschule Brandenburg
Organizational Unit Name (eg, section) []:Wirtschaft
Common Name (e.g. server FQDN or YOUR name) []:Franck Kateu
Email Address []:lekeufac@th-brandenburg.de
```

[illegible]



Name	Änderungsdatum	Typ	Größe
certificate.pem	22.06.2023 11:07	PEM-Datei	2 KB
certificate.srl	22.06.2023 12:51	SRL-Datei	1 KB
client_cert.pem	22.06.2023 12:51	PEM-Datei	2 KB
client_csr.pem	22.06.2023 12:43	PEM-Datei	2 KB
client_key.pem	22.06.2023 12:39	PEM-Datei	2 KB
key.pem	22.06.2023 11:06	PEM-Datei	2 KB

Abbildung 17: Erzeuge Dokument (Certificates and Keys)

6.2 Sicherheitsaspekte

Software-Sicherheit ist das Konzept der Implementierung von Mechanismen in das Sicherheitskonstrukt, die dazu beitragen, dass dieses Konstrukt gegenüber Angriffen funktionsfähig (oder widerstandsfähig) bleibt. Das bedeutet, dass Software, bevor sie auf den Markt kommt, einem Software-Sicherheitstest unterzogen wird, um festzustellen, ob sie böartigen Angriffen standhalten kann. Die Sicherheit von Software spielt eine große Rolle, weil ein Malware-Angriff jede Software schädigen und ihre Integrität, Authentifizierung und Verfügbarkeit gefährden werden kann. Wenn die Programmierer dies bereits bei der Programmierung und nicht erst im Nachhinein berücksichtigen, können Schäden vermieden werden (THALES, O.J). In RabbitMQ gibt es verschiedene Aspekte der Sicherheit, die berücksichtigt werden müssen.

Die auf AMQP basierenden Lösungen unterstützen Sicherheit auf der Sicherheitsebene mit TLS. RabbitMQ bietet Authentifizierungs- und Zugriffskontrollmechanismen für den Zugriff auf Warteschlangen.

6.3 Konfigurationen

Die SSL-Konfiguration

TLS dient in erster Linie zwei Zwecken: der Verschlüsselung des Verbindungsverkehrs und der Authentifizierung (Verifizierung) der Gegenstelle, um Man-in-the-Middle-Angriffe abzuschwächen. Beide Aufgaben werden durch eine Reihe von Rollen, Richtlinien und Verfahren erfüllt, die als Public Key Infrastructure (PKI) bekannt sind.

Eine PKI basiert auf dem Konzept digitaler Identitäten, die kryptographisch (mathematisch) verifiziert werden können (RabbitmqSSL, O.J).

Jeder TLS-fähige Server verfügt in der Regel über ein eigenes **Zertifikat-Schlüssel-Paar**, das er zur Berechnung eines verbindungspezifischen Schlüssels verwendet, der zur Verschlüsselung des über die Verbindung gesendeten Datenverkehrs eingesetzt wird (RabbitmqSSL, O.J).

Im Zusammenhang mit Messaging und Tools wie RabbitMQ ist es durchaus üblich, dass auch Clients Zertifikat/Schlüssel-Paare verwenden, damit Server ihre Identität überprüfen können.

Ein TLS-fähiger RabbitMQ-Knoten muss eine Reihe von Zertifikaten einer Zertifizierungsstelle, die er für vertrauenswürdig hält, in einer Datei (einem CA-Bündel), einer Zertifikatsdatei (öffentlicher Schlüssel) und einer privaten Schlüsseldatei haben. Die Dateien werden aus dem lokalen Dateisystem gelesen.

Um die Sicherheit der Kontrollzugriff an Queues zu erhöhen, ist es notwendig, das Standardpasswort und den Standardbenutzernamen der UI von RabbitMQ durch entsprechende Zugangsdaten zu ersetzen und die entsprechenden Berechtigungen zu geben. Folgende Befehlen ermögliche das zu tun. Die Befehle müssen in dem Terminal in RabbitMQ-Server ausgeführt werden.

1- Default User löschen

```
# rabbitmqctl delete_user guest
```

2- Neuer User anlegen

```
# rabbitmqctl add_user compute01 RABBIT_PASS
```

3- Berechtigungen zuweisen

```
# rabbitmqctl set_permissions compute01 ".*" ".*" ".*"
```

um einen unberechtigten Zugriff durch andere Prozesse und Benutzer auf dem Mailserver zu verhindern, wird ebenfalls die Konfigurationsdatei angepasst.

Damit der Client und Server einen sicheren Weg miteinander bauen können, werden die Pfade zu dem Zertifikat und den Key in die Konfigurationsdatei hingewiesen.

```
/etc/rabbitmq/conf.d/10-defaults.conf •  
1  loopback_users.guest = false  
2  
3  log.console = true  
4  
5  listeners.ssl.default = 5672  
6  
7  ssl_options.certfile   = /etc/rabbitmq/certificat.pem  
8  ssl_options.keyfile    = /etc/rabbitmq/key.pem  
9  ssl_option.verify      = verify_peer  
10 ssl_options.fail_if_no_peer_cert = true
```

Abbildung 18: Konfigurationsdatei

TLS-Zertifikate (Transport Layer Security) dienen dazu, den Server gegenüber dem Client zu authentifizieren. Das Zertifikat enthält Informationen über den Server, einschließlich seines öffentlichen Schlüssels. Der Client kann das Zertifikat überprüfen, um sicherzustellen, dass es von einer vertrauenswürdigen Zertifizierungsstelle (CA) ausgestellt wurde. So kann der Client sicherstellen, dass er mit dem erwarteten Server kommuniziert und nicht Opfer eines Phishing- oder Man-in-the-Middle-Angriffs wird.

Diese Authentifizierung erfolgt automatisch, wenn und nur wenn das Client-Zertifikat vom Server signiert wurde oder durch ein Certificate-Authority, das vom Server erkannt werden kann. Mit dem folgenden Befehl wird das Clientzertifikat von der Serverautorität signiert:

```
openssl x509 -req -in client_csr.pem -CA server_cert.pem -CAkey  
server_key.pem -CAcreateserial -out client_cert.pem
```

Die Abbildung 20 verdeutlicht diese Authentifizierung auf 3 Ebene (Client, Certificate Authority and Server).

```

Administrator: Eingabeaufforderung
A challenge password []:Ogy@0U40kwH1
An optional company name []:.

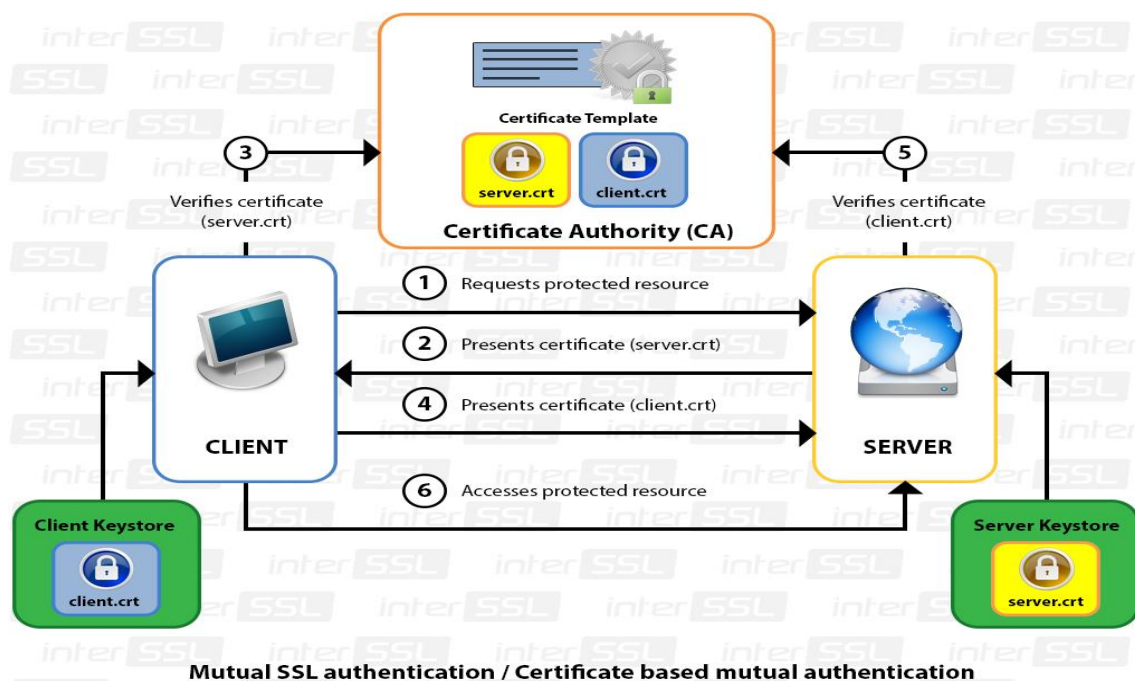
C:\Program Files\OpenSSL-Win64\bin\client certificat>openssl x509 -req -in client_csr.pem -CA server_cert.pem -CAkey server_key.pem -CAcreateserial -out client_cert.pem
Certificate request self-signature ok
subject=C = DE, ST = Brandenburg, L = "Brandenburg an der Havel ", O = Technische Hochschule Brandenburg, OU = Wirtschaft, CN = Franck Derrick, emailAddress = lekeufac@th-brandenburg.de
Could not open file or uri for loading CA certificate from server_cert.pem
584B0000:error:16000069:STORE routines:ossl_store_get0_loader_int:unregistered scheme:crypto\store\store_register.c:237:scheme=file
584B0000:error:80000002:system library:file_open:No such file or directory:providers\implementations\storemgmt\file_store.c:267:calling stat(server_cert.pem)
unable to load CA certificate

C:\Program Files\OpenSSL-Win64\bin\client certificat>openssl x509 -req -in client_csr.pem -CA certificate.pem -CAkey key.pem -CAcreateserial -out client_cert.pem
Certificate request self-signature ok
subject=C = DE, ST = Brandenburg, L = "Brandenburg an der Havel ", O = Technische Hochschule Brandenburg, OU = Wirtschaft, CN = Franck Derrick, emailAddress = lekeufac@th-brandenburg.de

C:\Program Files\OpenSSL-Win64\bin\client certificat>

```

Abbildung 19: Signieren von Client-Certificat durch Server Certificate-Authority



Quelle: (InterSSL, o.J)

Abbildung 20: Client-Server-Authetifizierung

7 Anwendung Szenarien von Rabbit MQ in der Praxis

Heutzutage wird der Fortschritt der Technologie in vielen Bereichen angewendet. Einer der wichtigsten Aspekte bei der Einführung neuer Technologien in Unternehmen ist neben den Kosten, dem qualifizierten Personal, die Flexibilität der Software. RabbitMQ ist ein Werkzeug zur Kommunikation zwischen mehreren Anwendern, dass eine hohe Flexibilität bietet und daher in vielen Bereichen eingesetzt wird:

- Cloud-native Applikation
- IoT
- Datenverarbeitung
- E-Commerce

In dieser Aufgabe liegt der Fokus auf dem IoT-Bereich, in dem mehrere Szenarien implementiert werden, um die Flexibilität und Skalierbarkeit zu zeigen. Es wird in diesem Bereich, um die Luftqualitätskontrolle mit Warnungsinformationen.

Anwendung Szenario 1

Das erste Programm wird eine einzelne Nachricht an die Warteschlange senden. Der Nachrichtensender wird so programmiert, dass er alle 10 Sekunden automatisch eine Nachricht an die Warteschlange sendet. Die Empfänger erhalten also die Nachrichten. Jedes Mal, wenn die Nachricht von den Empfängern verbraucht wurde, wird sie automatisch aus der Warteschlange entfernt. Dieses Beispiel entspricht in dem Anhang die Anwendung Szenario Nummer 1.

Anwendung Szenario 2

In diesem Tutorial werden wir eine Arbeitswarteschlange erstellen, die dazu dient, zeitraubende Aufgaben auf mehrere Arbeiter zu verteilen. Die Grundidee von Arbeitswarteschlangen (auch Aufgabenwarteschlangen genannt) besteht darin, dass wir eine ressourcenintensive Aufgabe nicht sofort erledigen und warten müssen, bis sie erledigt ist. Stattdessen planen wir die Aufgabe so, dass sie zu einem späteren Zeitpunkt erledigt wird. Wir verkapseln eine Aufgabe in einer Nachricht und senden sie an die Warteschlange. Dieses Konzept ist besonders nützlich für Webanwendungen, bei denen es unmöglich ist, eine

komplexe Aufgabe in einem kurzen HTTP-Anfragefenster zu erledigen. Wir werden nun Zeichenketten senden, die komplexe Aufgaben darstellen. Da wir keine echte Aufgabe haben, wie z. B. Bilder, die in der Größe verändert werden müssen, oder PDF-Dateien, die zurückgegeben werden müssen, tun wir einfach so, als wären wir beschäftigt - mithilfe der Funktion `time.sleep()`. Wir nehmen die Anzahl der Punkte in der Zeichenkette als ihre Komplexität an; jeder Punkt entspricht einer Sekunde "Arbeit". Eine Aufgabe, die von Hello... simuliert wird, dauert beispielsweise drei Sekunden (Tutorials, kein Datum).

Diese Aufgabe entspricht die zweite Anwendungsszenario in dem Anhang. Erklärungen zu dem Code und Methoden werden näher in die Präsentation durchgeführt.

8 Schlussteil

Zusammenfassend lässt sich sagen, dass RabbitMQ eine leistungsstarke Nachrichten-Middleware ist, die eine zentrale Rolle bei der Entwicklung von verteilten und skalierbaren Anwendungen spielt. Mit dem Schwerpunkt auf Zuverlässigkeit, Flexibilität und Skalierbarkeit bietet RabbitMQ eine solide Grundlage für den Aufbau von Systemen, die den Austausch und die Verteilung von Nachrichten erfordern. Ein wichtiger Vorteil von RabbitMQ besteht darin, dass es verschiedene Nachrichtenmodelle unterstützt, darunter Publish/Subscribe, Point-to-Point und Request/Response. Diese Flexibilität ermöglicht es Entwicklern, die Kommunikationsmodelle zu wählen, die für ihre spezifischen Anwendungsfälle am besten geeignet sind. Darüber hinaus bietet diese Software eine Vielzahl von Funktionen wie persistente Speicherung von Nachrichten, Nachrichtenrouting, Nachrichtenbestätigung und Lastenausgleich. Diese Funktionen sorgen dafür, dass Nachrichten sicher und effizient zwischen den Komponenten eines verteilten Systems übertragen werden können. Ein weiterer großer Vorteil davon ist seine Unterstützung für eine Vielzahl von Programmiersprachen und Plattformen. Es bietet offizielle Client-Bibliotheken für verschiedene Sprachen wie Java, Python, Ruby und .NET, was die Integration in bestehende Systeme vereinfacht.

Darüber hinaus bietet RabbitMQ eine hohe Skalierbarkeit und Ausfallsicherheit. Es ermöglicht das Hinzufügen zusätzlicher Message Broker, um die Arbeitslast zu verteilen und die Verfügbarkeit des Systems zu verbessern. Durch den Einsatz von Clustering-Techniken kann RabbitMQ hochverfügbare und robuste Messaging-Infrastrukturen bereitstellen. Insgesamt bietet RabbitMQ eine zuverlässige und flexible Messaging-Lösung für die Entwicklung von verteilten Anwendungen. Es ermöglicht Entwicklern, komplexe Kommunikationsmodelle zu implementieren und die Skalierbarkeit, Zuverlässigkeit und Effizienz ihrer Systeme zu verbessern. Mit seiner aktiven Entwicklergemeinschaft und der breiten Unterstützung ist RabbitMQ eine gute Wahl für den Einsatz in verschiedenen Anwendungsbereichen.

9 Literaturverzeichnis

- AMQP, C. (o.J). *AMQP*. Von <https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html>) abgerufen
- Bikraume, K. (23. 03 2020). *Topic Exchange*. Von <https://jstobigdata.com/rabbitmq/topic-exchange-in-amqp-rabbitmq/> abgerufen
- CloudAMQP. (O.J). *CloudAMQP*. Von <https://www.cloudamqp.com/docs/stomp.html> abgerufen
- David, D. (2014). *Rabbit MQ Essential*. Von <https://www.packtpub.com/product/rabbitmq-essentials/9781783983209> abgerufen
- Dipl- Ing Luber, D.-I. (. (15. 12 2021). *Big Data Insider*. Von <https://www.bigdata-insider.de/was-ist-rabbitmq-a-1082460/> abgerufen
- Educba. (o.J). *Educba*. Abgerufen am 01. 05 2023 von <https://www.educba.com/rabbitmq-routing-key/>
- Gregor, H., & Bobby, W. (o.J). *Enterprise Integration Patterns*. Abgerufen am 24. 04 2023 von <https://www.enterpriseintegrationpatterns.com/patterns/messaging/PointToPointChannel.html>
- IoT, B. (02. 04 2018). *IoT Boys*. Von <https://iotboys.com/what-is-amqp-how-amqp-works-for-internet-of-things/> abgerufen
- IT-Wissen. (0.J). *IT-Security Wissen* . Von <https://it-security-wissen.de/openssl.html> abgerufen
- Jabali, M. (15. 01 2013). *An Introduction to STOMP*. Von <https://dzone.com/articles/introduction-stomp> abgerufen
- Linfeed. (25. 04 2017). *Docker*. Von <https://www.dev-insider.de/was-sind-docker-container-a-3be08dcd766e143c5f850bd6433ac09e/> abgerufen
- Microsoft, B. (29. 11 2022). *Microsoft Build*. Abgerufen am 19. 04 2023 von <https://learn.microsoft.com/de-de/azure/service-bus-messaging/service-bus-amqp-overview>
- MQTT. (o.J). *MQTT*. Von <https://mqtt.org/> abgerufen
- Nguyen, T. (2. 4 2023). *Frontendmag*. Von <https://www.frontendmag.com/insights/signalr-vs-rabbitmq/> abgerufen
- Olah, G. (03. 04 2020). *Erlang Solution* . Von <https://www.erlang-solutions.com/blog/an-introduction-to-rabbitmq-what-is-rabbitmq/> abgerufen

- OpenStack, O. (O.J). *Messaging-Sicherheit*. Von
<https://docs.openstack.org/de/security-guide/messaging/security.html>
abgerufen
- PAESSLER. (03. 04 2019). *PAESSLER The Monitoring Experts*. Von
<https://www.paessler.com/de/it-explained/mqtt> abgerufen
- RabbitMQ, D. (o.J). *RabbitMQ*. Abgerufen am 25. 04 2023 von
<https://www.rabbitmq.com/documentation.html>
- RabbitmqSSL. (O.J). *RabbitmqSSL*. Abgerufen am 21. 05 2023 von
<https://www.rabbitmq.com/ssl.html>
- STOMP. (O. J). *Fun Tech Projects*. Von
<https://funprojects.blog/2020/05/14/stomp-protocol-with-rabbitmq-node-red-and-python/> abgerufen
- THALES. (O.J). *Building a future we can all trust*. Von
<https://cpl.thalesgroup.com/de/software-monetization/what-is-software-security> abgerufen
- Tran, T. (19. 12 2020). *Programming Sharing*. Von
<https://programmingsharing.com/point-to-point-and-publish-subscribe-messaging-model-2efc4d2b6726> abgerufen
- Tutorials, R. (kein Datum). *RabbitMQ Tutorials*. Von
<https://www.rabbitmq.com/tutorials/tutorial-two-python.html> abgerufen

Anhang

Python-Code für die 2 Anwendungsbeispiele