Hochschule Konstanz
Fakultät Elektrotechnik
und Informationstechnik

Advanced Message Queuing Protocol (AMQP)

Jakob Hotz, Chin-I Feng

Hochschule Konstanz
Fakultät Elektrotechnik
und Informationstechnik

Agenda

- Einleitung
- Geschichte & Entwicklung
- Architektur & Konzepte
- Code-Beispiele
- Sicherheit
- Pro und Contra

Einleitung

AMQP (**A**dvanced **M**essage **Q**ueuing **P**rotocol) ist ein offenes, plattformunabhängiges und sprachneutrales Standard-Protokoll für die asynchrone Übermittlung von Nachrichten zwischen verteilten Systemen.

Es wurde entwickelt, um eine zuverlässige und sichere Kommunikation zwischen unterschiedlichen Applikationen und Plattformen zu ermöglichen.

(OASIS Standard: 2012)

Geschichte & Entwicklung

- Anfänge:
  - **2003** initiiert von John O‘Hara (JPMorgan Chase)
  - Notwendigkeit einer besseren Kommunikation zwischen Finanzanwendungen

- Weiterentwicklung:
  - **2006** Arbeitsgruppe aus vers. Unternehmen entwickelt das Protokoll weiter
  - **2008** Erste öffentliche Version von AMQP (Version 0.8)

- Standardisierung:
  - **2011** Veröffentlichung AMQP 1.0 mit wichtigen Verbesserungen
  - **2012** Anerkennung durch OASIS

Architektur & Konzepte

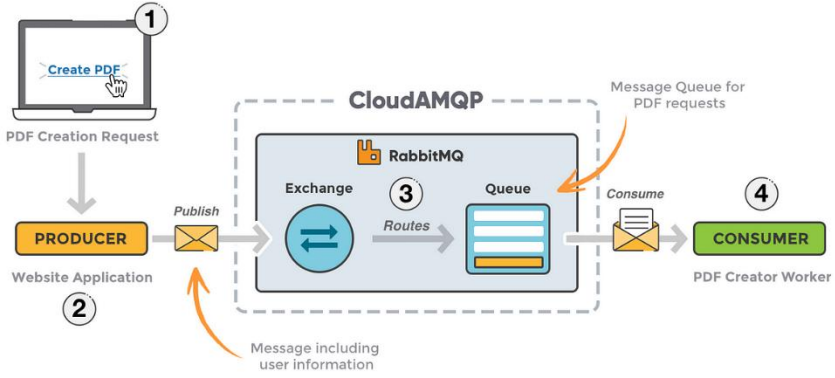
- Hauptkomponenten der AMQP-Architektur:

Komponente	Beschreibung
Client	System, das mit Broker kommuniziert
Broker	Zentrale Komponente, verwaltet Nachrichtenübertragung
Verbindung	TCP-Socket zwischen Client und Broker
Kanal	Virtuelle Verbindung, ermöglicht parallele Kommunikation
Exchange	Nachrichtenverteiler, leitet Nachrichten an Queues weiter
Queue	Nachrichtenspeicher im Broker, liefert Nachrichten an Consumer
Binding	Verknüpfung Exchange-Queue, definiert Routing-Regeln
Routing-Schlüssel	Identifikator, steuert Nachrichtenweiterleitung

Architektur & Konzepte

Client

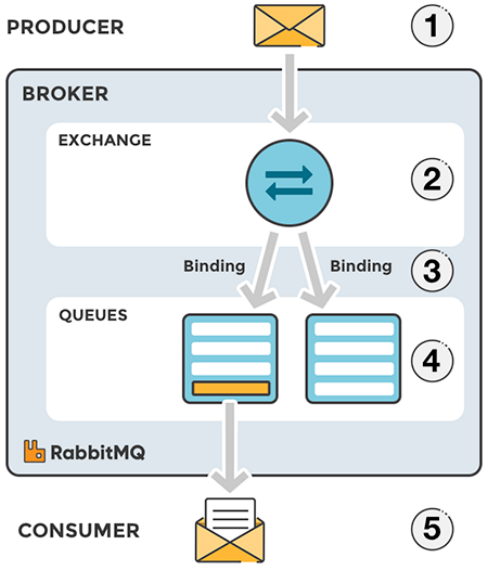
- Kommunikationsschnittstelle zwischen Anwendung und AMQP-Broker
- Sendet und empfängt Nachrichten über Broker
- Client-Aufgaben:
  - Verbindung zum Broker herstellen
  - Kanäle erstellen und verwalten
  - Nachrichten veröffentlichen (Producer)
  - Nachrichten empfangen und verarbeiten (Consumer)



Architektur & Konzepte

Broker

- Verantwortlich für Nachrichtenübertragung und –verwaltung
- Empfängt, speichert und leitet Nachrichten an Clients weiter
- Broker-Aufgaben:
  - Verbindungen von Clients akzeptieren und verwalten
  - Kanäle innerhalb von Verbindungen verwalten
  - Nachrichten empfangen und in Queues speichern
  - Nachrichten anhand von Routing-Regeln weiterleiten
  - Nachrichten an Consumer ausliefern



Architektur & Konzepte

Verbindung

- Kommunikationsweg zwischen Client und Broker
- Basierend auf TCP-Socket
- Erlaubt bidirektionale Kommunikation
- Eigenschaften:
  1. Hostname/IP-Adresse und Port des Brokers
  2. Anmeldeinformationen (Benutzername, Passwort)
  3. Verbindungsüberwachung
  4. Maximale Größe von AMQP-Rahmen

Architektur & Konzepte

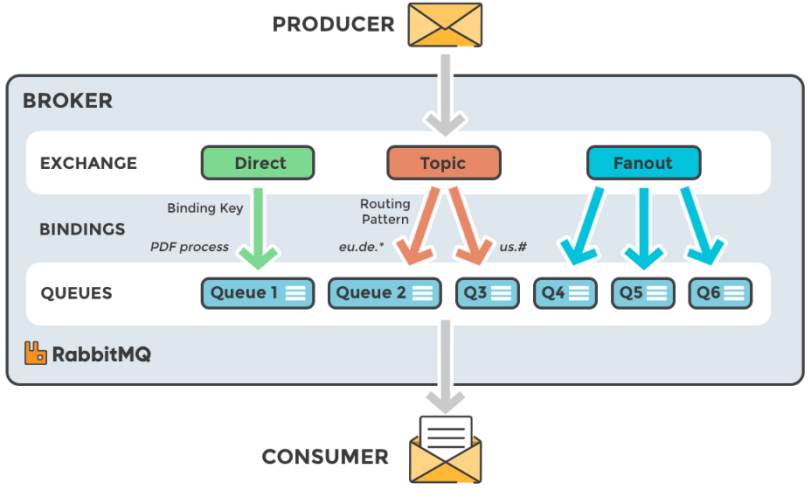
Kanal

- Virtuelle Verbindung innerhalb einer AMQP-Verbindung
- Ermöglicht parallele Kommunikation zwischen Client und Broker
- Vereinfacht Multiplexing und effiziente Ressourcennutzung
- Kanal-Erstellung:
  1. Client fordert Kanal über bestehende Verbindung an
  2. Broker erstellt Kanal und bestätigt Client-Anfrage
  3. Kommunikation über Kanal beginnt

Architektur & Konzepte

Exchange

- Nachrichtenverteiler im AMQP-Broker
- Leitet Nachrichten an passende Queues weiter
- Verwendet Routing-Regeln, definiert durch Bindings und Routing-Schlüssel
- Art des Exchanges (Direct, Topic, Fanout, Headers)



Architektur & Konzepte

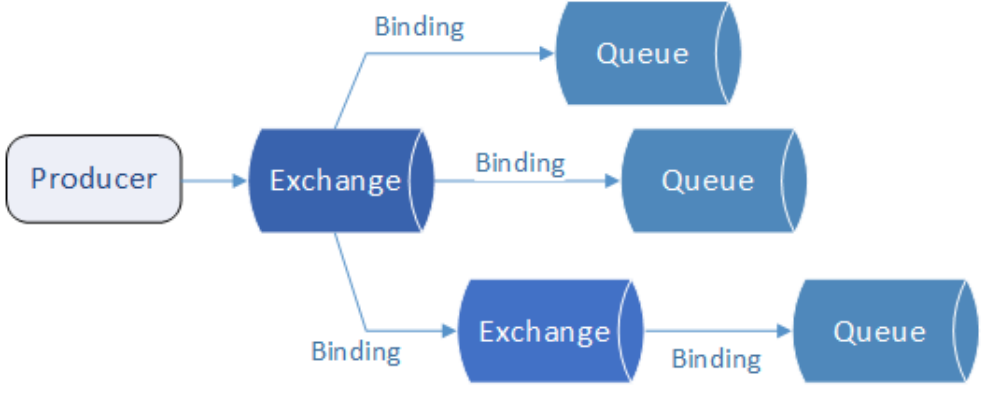
Queue

- Nachrichtenspeicher im AMQP-Broker
- Temporäre oder persistente Speicherung von Nachrichten
- Unterstützt Push- und Pull-Modus:
  1. Push-Modus: Broker liefert Nachrichten automatisch an Consumer
  2. Pull-Modus: Consumer fordert aktiv Nachrichten vom Broker an
- Queue-Eigenschaften:
  1. Haltbarkeit: Lebensdauer der Queue (transient, persistent)
  2. Exklusivität: Queue nur für bestimmten Client zugänglich
  3. Maximale Länge: Begrenzung der Anzahl von Nachrichten in der Queue

Architektur & Konzepte

Binding

- Verknüpfung zwischen Exchange und Queue
- Definiert Routing-Regeln für Nachrichten
- Binding-Prozess:
  1. Client erstellt Exchange und Queue (falls noch nicht vorhanden)
  2. Client erstellt Binding zwischen Exchange und Queue mit passendem Schlüssel
  3. Nachrichten werden zwischen Exchange und Queue weitergeleitet



Architektur & Konzepte

Routing-Schlüssel

- Schlüssel zur Steuerung der Nachrichtenverteilung
- Wird vom Producer festgelegt, wenn eine Nachricht veröffentlicht wird
- Beeinflusst, wie ein Exchange die Nachricht an Queues weiterleitet
- Abhängig vom Exchange-Typ (Direct, Fanout, Topic, Headers)
- Routing-Schlüssel-Nutzung:
  1. Steuert die Zielgruppe für Nachrichten
  2. Ermöglicht flexible und gezielte Nachrichtenverteilung

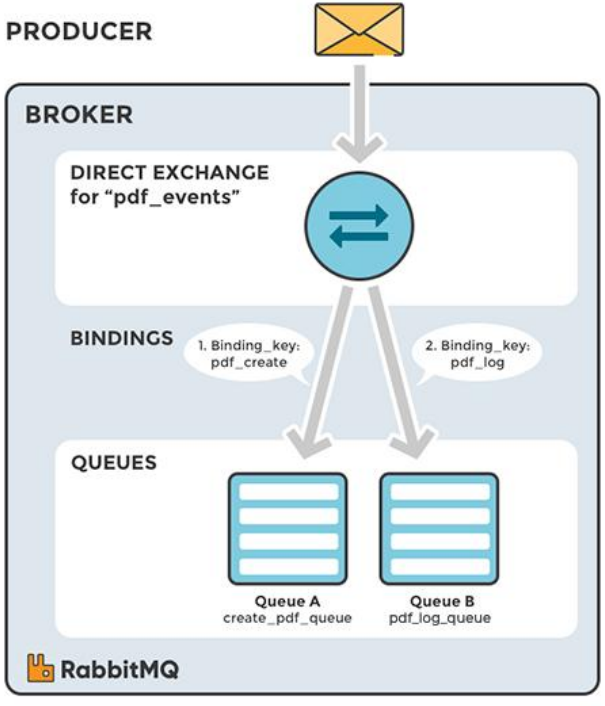
Architektur & Konzepte

Routing-Schlüssel bei „Direct" Exchange

- Exakte Übereinstimmung mit Binding-Schlüssel
- Einfache, präzise und direkte Routing-Strategie

- Anwendungsbeispiel

1. Task-Verteilung an bestimmte Arbeiter
2. Nachrichten an bestimmte Benutzer oder Gruppen



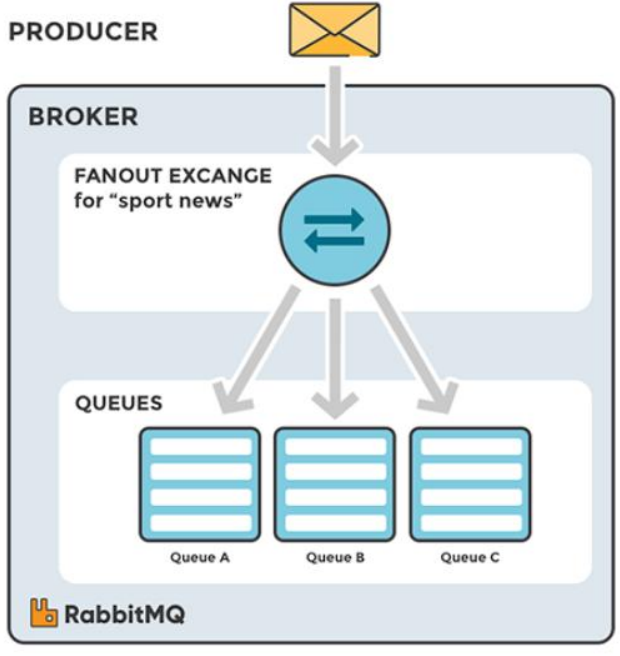
Architektur & Konzepte

Routing-Schlüssel bei „Fanout" Exchange

- Leitet Nachrichten an alle gebundenen Queues weiter
- Unabhängig vom Routing-Schlüssel

- Anwendungsbeispiel

1. Echtzeit-Updates für alle Abonnenten
2. Massenbenachrichtigungen oder Alarme



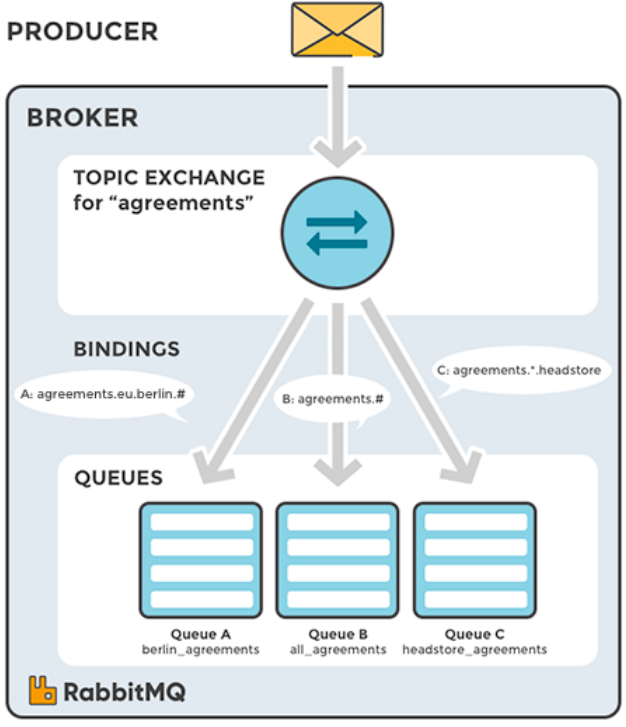
Architektur & Konzepte

Routing-Schlüssel bei „Topic“ Exchange

- Übereinstimmung mit Routing-Schlüssel-Mustern (*, #)

- Anwendungsbeispiel

1. Log-Nachrichtenverteilung nach Schweregrad und Kategorie
2. Nachrichten an verschiedene Benutzergruppen mit gemeinsamen Interessen



Architektur & Konzepte

Routing-Schlüssel bei „Headers“ Exchange

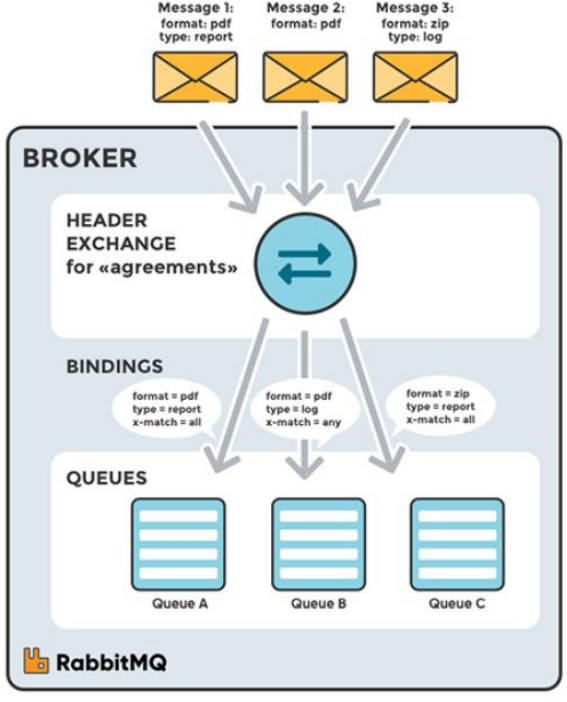
• Leitet Nachrichten basierend auf Nachrichtenattributen (Headern) weiter

• Übereinstimmungsoptionen:

1. "all": Alle Header-Argumente müssen übereinstimmen
2. "any": Mindestens ein Header-Argument muss übereinstimmen

• Anwendungsbeispiel

1. Gezielte Benachrichtigungen basierend auf Benutzereigenschaften (z.B. Sprache, Region)
2. Nachrichtenverteilung anhand von Priorität



Code-Beispiel 1

Erstellen einer Verbindung und eines Kanals

```python
# Python-Implementierung des AMQP-0-9-1-Protokolls fuer RabbitMQ
import pika

# Verbindung zum AMQP-Broker herstellen
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
```

Code-Beispiel 2

Producer (Nachrichten senden):

```
# Direct Exchange erstellen
channel.exchange_declare(exchange='direct_logs', exchange_type='direct')

# Nachricht an Exchange senden mit Routing-Schlüssel
routing_key = 'error'
message = 'Fehlermeldung von App1'
channel.basic_publish(exchange='direct_logs', routing_key=routing_key, body=message)
```

Code-Beispiel 3

Consumer (Nachrichten empfangen)

```python
# Direct Exchange und Queue erstellen
channel.exchange_declare(exchange='direct_logs', exchange_type='direct')
result = channel.queue_declare(queue='', exclusive=True)
queue_name = result.method.queue

# Binding zwischen Exchange und Queue mit exakter Übereinstimmung des Routing-Schlüssels
binding_key = 'error'  # Empfängt alle Fehlermeldungen
channel.queue_bind(exchange='direct_logs', queue=queue_name, routing_key=binding_key)

# Callback-Funktion, die bei Nachrichtenempfang aufgerufen wird
def on_message(ch, method, properties, body):
    print(f"Received: {body}")

# Nachrichten konsumieren
channel.basic_consume(queue=queue_name, on_message_callback=on_message, auto_ack=True)
channel.start_consuming()
```

Sicherheitsmechanismen

- **Authentifizierung**: Identifizierung von Clients (Benutzern)
  - Verwendung von Benutzernamen und Passwörtern oder externen Authentifizierungssystemen

- **Autorisierung**: Festlegen von Zugriffsrechten für Clients
  - Erlaubt oder verbietet Aktionen wie das Erstellen von Queues, Exchanges oder das Senden von Nachrichten

- **Verschlüsselung**: Schutz der übertragenen Daten
  - Einsatz von Transport Layer Security (TLS) zur Verschlüsselung der Verbindung und zum Schutz der Kommunikation

Best Practices

- Verwendung starker Benutzernamen und Passwörter
- Minimierung der Anzahl von Administratoren und Zugriffsrechten
- Regelmäßige Aktualisierung von Software und Sicherheitsrichtlinien
- Netzwerksegmentierung und Firewall-Konfiguration zum Schutz des AMQP-Brokers

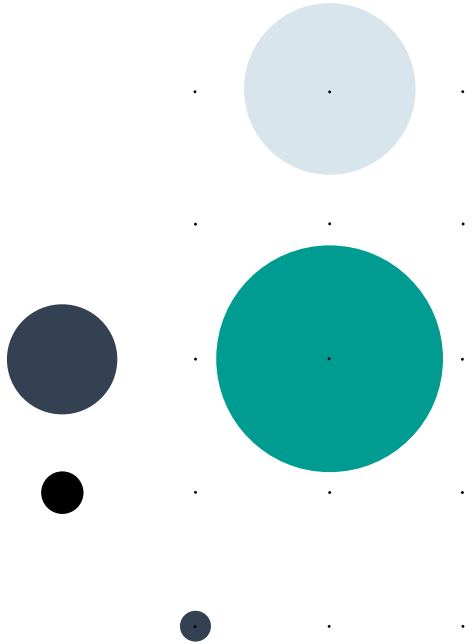
Pro und Contra

- **Pro**
  - Skalierbarkeit: Unterstützt verteilte, wachsende Anwendungen
  - Flexibles Routing: Verschiedene Exchange-Typen für vielfältige Anforderungen
  - Zuverlässigkeit: Bestätigte Nachrichtenübermittlung
  - Sicherheit: Authentifizierung, Autorisierung, Verschlüsselung
  - Interoperabilität: Offenes Protokoll für vielfältige Plattformen

- **Contra**
  - Komplexität: Umfangreicher Funktionsumfang erschwert Implementierung
  - Overhead: Höherer Kommunikationsaufwand als leichtgewichtigere Protokolle (MQTT)

Vielen Dank

für die Aufmerksamkeit



Quellen

- OASIS Standard, "Advanced Message Queuing Protocol (AMQP) Version 1.0", 29. Oktober 2012. Verfügbar unter: https://www.oasis-open.org/committees/download.php/49239/amqp-core-overview-v1.0-wd01.pdf
- https://suthesana.medium.com/what-is-rabbitmq-5194af585d36
- https://www.cloudamqp.com/docs/amqp.html
- https://www.cloudamqp.com/blog/the-relationship-between-connections-and-channels-in-rabbitmq.html
- https://www.tutlane.com/tutorial/rabbitmq/rabbitmq-exchanges
- https://www.cogin.com/articles/rabbitmq/rabbitmq-exchanges-guide.php
- https://www.educba.com/rabbitmq-routing-key/
- https://www.cloudamqp.com/blog/part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html
- https://www.wallarm.com/what/what-is-amqp#:~:text=Its%204%20categories%20are%3A%20Fanout,indispensable%20component%20of%20the%20broker.&text=Channel%20refers%20to%20a%20multiplexed,built%20inside%20an%20existing%20connection.