

Vergleich von Apache Kafka und RabbitMQ

Hausarbeit

vorgelegt von

Bjarne Christel

aus Mönchengladbach

geboren am: 29.03.1999

Matrikelnr.: 1344413

Tobias Piepers

aus Mönchengladbach

geboren am: 19.01.1998

Matrikelnr.: 1359763

Louisa Schmitz

aus Krefeld

geboren am: 11.07.2000

Matrikelnr.: 1319646

Hochschule Niederrhein

Fachbereich Wirtschaftswissenschaften

Studiengang M. Sc. Wirtschaftsinformatik

Wintersemester 2023/ 2024

Prüfer: Prof. Dr. Schekelmann

Inhaltsverzeichnis

Abkürzungsverzeichnis	III
Abbildungsverzeichnis.....	IV
Tabellenverzeichnis	V
1 Einleitung.....	1
2 Grundlagen.....	1
3 Systematische Literaturanalyse.....	2
3.1 Vorbereitung.....	4
3.2 Durchführung.....	6
3.3 Präsentation der Suchergebnisse.....	7
3.4 Vergleich von Apache Kafka und RabbitMQ	8
4 Projektarbeit	13
5 Fazit.....	17
Literaturverzeichnis	18
Anhang.....	21
<i>Anhang 1: Ausgewertete Literatur</i>	21

Abkürzungsverzeichnis

AMQP	Advanced Message Queuing Protocol
HTTP	Hypertext Transfer Protocol
JVM	Java Virtual Machine
Kafka	Apache Kafka
LDAP	Lightweight Directory Access Protocol
MOM	Message Oriented Middleware
MQTT	Message Queuing Telemetry Transport
RabbitMQ	Rabbit Messaging Queue
RAM	Random Access Memory
SASL	Simple Authentication and Security Layer
SSL	Secure Sockets Layer
STOMP	Simple Text Oriented Message Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security

Abbildungsverzeichnis

Abbildung 1 Durchführung.....	4
Abbildung 2 Suchbegriffe.....	5
Abbildung 3 Kafka Kommunikationsprozess.....	10
Abbildung 4 RabbitMQ Kommunikationsprozess.....	11
Abbildung 5 Programmcode Kafka: bean Konfiguration für den Empfänger.....	13
Abbildung 6 Programmcode RabbitMQ: bean Konfiguration für den Empfänger	14
Abbildung 7 Programmcode Kafka: EventListener für den Empfänger.....	14
Abbildung 8 Programmcode RabbitMQ: EventListener für den Empfänger	15
Abbildung 9 Programmcode Kafka: bean Konfiguration für den Sender	15
Abbildung 10 Programmcode RabbitMQ: bean Konfiguration für den Sender	16
Abbildung 11 Programmcode Kafka: Message Queue.....	16
Abbildung 12 Programmcode RabbitMQ: Message Queue	17

Tabellenverzeichnis

Tabelle 1 Ein- und Ausschlusskriterien.....	6
Tabelle 2 Iterationen der Suche.....	6
Tabelle 3 Resultierende Literatur.....	8
Tabelle 4 Konzeptmatrix.....	8
Tabelle 5 Systemvergleich	12

1 Einleitung

Bei den Softwareanwendungen Apache Kafka¹ und RabbitMQ² handelt es sich um Nachrichtenwarteschlangen-Systeme, die es Produzenten ermöglicht, Nachrichten an Empfänger zu versenden.³

Im Rahmen der Projektarbeit werden in dieser Ausarbeitung die beiden Anwendungen untersucht und verglichen. Ziel dieser Arbeit ist es, die Eigenschaften, Gemeinsamkeiten und Unterschiede der beiden Anwendungen herauszuarbeiten. Es wird eine systematische Literaturrecherche nach vom Brocke et al. durchgeführt, um die Aufgabenstellung zu beantworten.

Im zweiten Abschnitt der Arbeit werden die Grundlagen erläutert, die zum näheren Verständnis der Arbeit notwendig sind. Im dritten Abschnitt folgt der Hauptteil der Arbeit, die systematische Literaturrecherche sowie die Analyse der resultierenden Literatur, um die Eigenschaften, Gemeinsamkeiten und Unterschiede von Kafka und RabbitMQ herauszuarbeiten. Anschließend ist der vierte Abschnitt zur Darstellung der Implementierung. Es werden Ausschnitte des Codes dieser Projektarbeit gezeigt. Es folgt der letzte Abschnitt mit dem Fazit der Ausarbeitung.

2 Grundlagen

Hinter den zu vergleichenden Anwendungen Kafka und RabbitMQ steht das Konzept der Nachrichtenübermittlung. Es handelt sich bei den beiden Tools um Implementierungen einer sogenannten Message Oriented Middleware (MOM), eine anwendungsneutrale Kommunikationsinfrastruktur. Dabei gibt es verschiedene, asynchrone Möglichkeiten der Vermittlung.⁴ Aufgrund der Aufgabenstellung beschränkt sich die Arbeit in diesem Kontext auf die asynchrone Kommunikation.

Bei asynchroner Kommunikation handelt es sich um eine Art von Kommunikation, bei der die Gesprächspartner nicht zeitgleich verfügbar sein müssen. Nachricht und Antwort werden nicht in Echtzeit erwartet, sondern können zeitlich versetzt gesendet und empfangen werden.

¹ Nachstehend kurz: Kafka.

² MQ steht für Messaging Queue.

³ Vgl. aws (o.D.).

⁴ Dillmann (2014).

Sie bringt im menschlichen Kontext verschiedene Vorteile mit sich:

- Mehr Zeit für Antworten bedeutet mehr Zeit zum Nachdenken, Informieren und Reflektieren, um bessere Entscheidungen zu treffen
- Zusammenarbeit von Teams über verschiedene Zeitzonen hinweg
- Flexible und hybride Arbeitsmodelle
- Digitale Medien speichern die gesamte Kommunikation, Abruf zu späterem Zeitpunkt jederzeit möglich.

Dagegen stehen allerdings auch Nachteile:

- Manchmal ist eine sofortige Antwort notwendig
- Verspätete oder ganz vergessene Antworten
- Zusätzlicher Speicheraufwand
- Reduziert zwischenmenschlichen Austausch und persönlichen Kontakt
- Nonverbale, zwischenmenschliche Kommunikation fehlt.⁵

Im Kontext dieser Arbeit geht es um asynchrone Kommunikation zwischen Anwendungen. Dies geschieht nach demselben Prinzip, eine Anwendung schickt eine Nachricht los, wartet aber nicht auf eine sofortige Antwort. Somit werden die Komponenten nicht zu eng gekoppelt und die aktuelle Verfügbarkeit der anderen Komponente ist nicht relevant.⁶

Um die Kommunikation im System dementsprechend ohne im Code verankerte Abhängigkeiten zu gestalten, kann eine ereignisgesteuerte Architektur verwendet werden. Sie gilt als Best Practice bei der Verwendung von Microservices, da die Anwendungen sich nicht kennen müssen und so die angestrebte lose Kopplung gegeben ist. Die Anwendungen sind hierbei Ereignisproduzenten und -verbraucher, wobei der Ereignisproduzent die Nachricht über einen Broker, auch MOM, an die Ereignisverbraucher übermittelt.⁷

Die ereignisgesteuerte Architektur wird hauptsächlich in zwei Muster für die Übertragung eingeteilt: Publish/ Subscribe und Event-Streaming.

3 Systematische Literaturanalyse

In diesem Kapitel wird eine systematische Literaturanalyse beschrieben. Dafür wird die verwendete Methodik dargelegt und die Durchführung dokumentiert. Aus der resultierenden

⁵ Vgl. TechSmith (o.D.).

⁶ WalkingTree Technologies (2018).

⁷ SAP (o.D.).

Literatur werden diejenigen Quellen näher betrachtet, die die definierte Ein- und Ausschlusskriterien erfüllen. Sie dienen dem Vergleich der beiden Anwendungen Kafka und RabbitMQ.

Die Literaturrecherche soll systematisch die in Bezug auf die Fragestellung relevante Literatur identifizieren und somit neue Erkenntnisse durch die Synthese vorhandener Literatur ermöglichen.⁸ Das Verfahren kann iterativ oder sequentiell durchgeführt werden. Bei einer sequentiellen Vorgehensweise werden die Phasen wiederholt nacheinander durchlaufen, während bei der iterativen Methodik mehrere Durchläufe von Suchen in der Datenbank und anschließender groben Analyse durchgeführt werden, bis eine zufriedenstellende Menge an thematisch geeigneter Literatur gefunden worden ist.⁹

Die in dieser Arbeit verwendete Vorgehensweise ist in Abbildung 1 Durchführung dargestellt. Zunächst wird die Problemstellung identifiziert und die Fragestellung formuliert. Im ersten Block der Schlagwortsuche werden die Suchbegriffe definiert und ein Suchterm aufgestellt. Mit diesem wird die Datenbank durchsucht. Im Rahmen der Relevanzbetrachtung werden die resultierenden Suchergebnisse exportiert, Dubletten entfernt und die Ein- und Ausschlusskriterien angewendet. Zu den verbleibenden Treffern werden die Volltexte beschafft. Nach dieser groben Analyse der Treffer wird hinterfragt, ob die gefundenen Treffer bereits eine zufriedenstellende Menge an Literatur darstellen oder weitergesucht werden soll. Im ersten Fall wird die Literaturrecherche selbst abgeschlossen und die Beschreibung und Analyse der Literatur begonnen. Andernfalls wird eine weitere Iteration mit einem angepassten Suchterm oder in einer weiteren Datenbank durchgeführt.

⁸ Vgl. vom Brocke, 2015, S.206.

⁹ Vgl. vom Brocke, 2015, S.209.

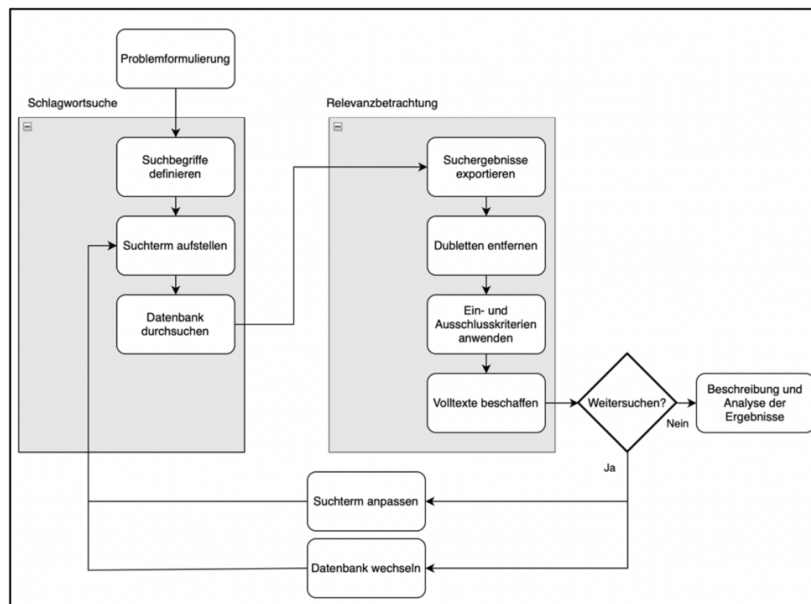


Abbildung 1 Durchführung¹⁰

3.1 Vorbereitung

Die Fragestellung der systematischen Literaturrecherche ergibt sich aus der vorliegenden Aufgabenstellung. Die beiden Tools Kafka und RabbitMQ sollen auf ihre Unterschiede und Gemeinsamkeiten hin untersucht werden. Aufgrund der Art der Software sowie der Aufgabenstellung ergeben sich die in Abbildung 2 Suchbegriffe dargestellten Begriffe als Suchbegriffe.

¹⁰ Eigene Darstellung in Anlehnung an Lempert, 2021, S.209.

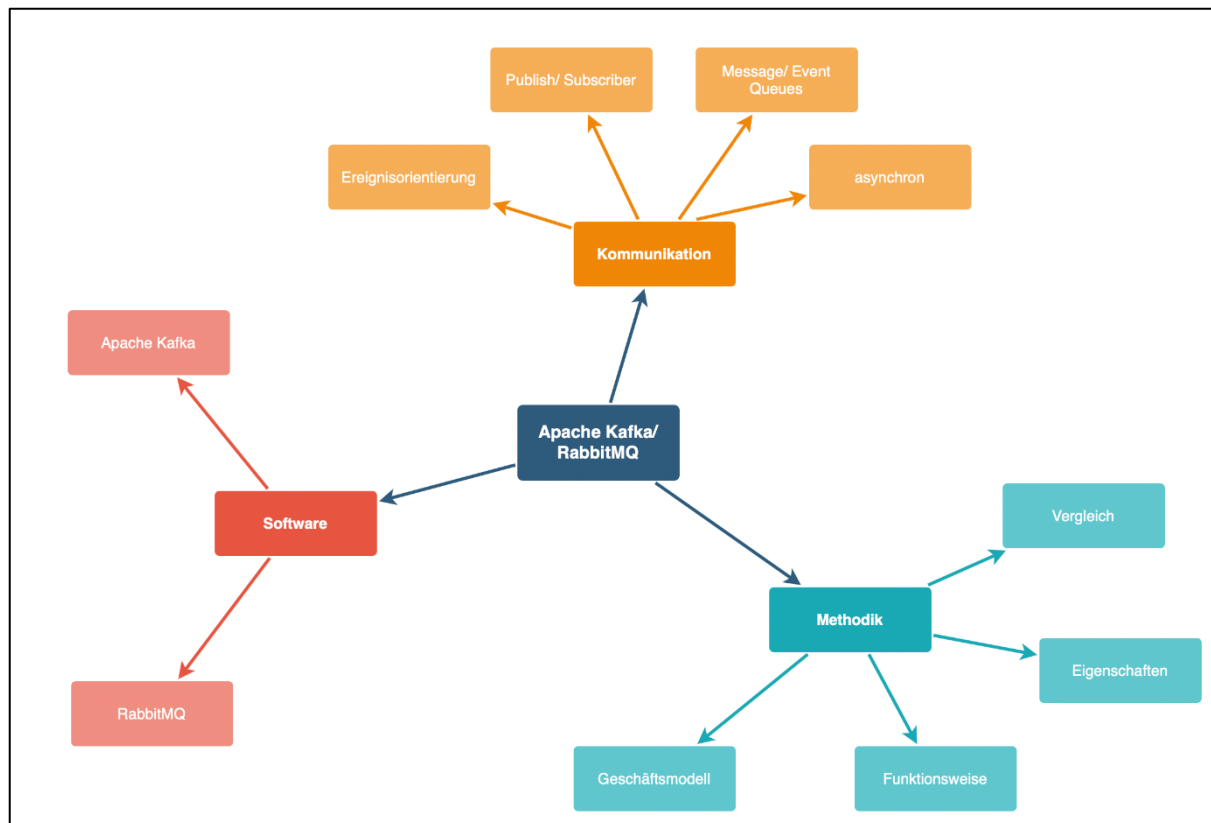


Abbildung 2 Suchbegriffe

Die Begriffe werden, soweit möglich, auch in die englische Sprache übersetzt, um auch englischsprachige Literatur zu finden. Die Begriffe können zur Erstellung der Suchterme mithilfe boolescher Operatoren AND, OR oder NOT miteinander kombiniert werden. Die Suchterme werden nach jeder Iteration neu aufgestellt und angepasst auf Basis der evaluierten Trefferliste des vorherigen Suchterms. Die Kombinationen werden daher erst in der konkreten Durchführung zusammengesetzt.

Für die Recherche werden folgende Datenbanken verwendet: SpringerLink, Hanser eLibrary und ScienceDirect. Die SpringerLink Datenbank handelt es sich um eine fachbereichsübergreifende Datenbank für naturwissenschaftliche, technische und medizinische Themen, die nach eigener Aussage weltweit führend ist¹¹. Die Hanser eLibrary widmet sich breit aufgestellt der Technik und dem Ingenieurwesen, wobei auch die IT als Fachgebiet ausgewiesen wird¹². ScienceDirect ist eine Plattform von Elsevier für peer-reviewed wissenschaftliche Literatur in den Bereichen Wissenschaft, Technik und Gesundheit¹³. Die drei Datenbanken wurden aufgrund ihrer thematischen Passung ausgewählt.

¹¹ SpringerLink (o.D.).

¹² Hanser (o.D.).

¹³ Elsevier (o.D.).

Die bei der Recherche entstehenden Trefferlisten werden unter Berücksichtigung der Ein- und Ausschlusskriterien grob untersucht und gefiltert. Die konkreten Kriterien werden in Tabelle 1 beschrieben.

Kriterium	Beschreibung	Kommentar
Verfügbarkeit	Volltext muss verfügbar sein	Öffentlich oder mit Hochschul-Zugriff verfügbar
Sprache	Deutsch, Englisch	Suchbegriffe entsprechend
Kontext	Konkret auf Kafka und/ oder RabbitMQ bezogen	Fokus auf Anwendungen in Titel und/ oder Abstract erkennbar
Zeitraum	Die letzten 5 Jahre	Aktueller Stand
Redundanz	Dubletten werden entfernt	
Dokumententyp	Nicht eingeschränkt	

Tabelle 1 Ein- und Ausschlusskriterien

3.2 Durchführung

Die Durchführung der systematischen Literaturrecherche ist in Tabelle 2 dargestellt, wobei jede Zeile eine Iteration darstellt. Die erste Iteration wird mit dem Suchterm „RabbitMQ AND Apache Kafka“ begonnen.

Datenbank	Suchterm	Einschränkung	Treffer	dv. verwendet
SpringerLink	RabbitMQ AND Apache Kafka	Ab 2018; Englisch & deutsch	264	-
SpringerLink	RabbitMQ AND Apache Kafka AND comparison	Ab 2018; Englisch & deutsch	160	5
ScienceDirect	RabbitMQ AND Apache Kafka AND comparison	Ab 2018	52	2

Tabelle 2 Iterationen der Suche

Die erste Iteration bringt insgesamt 264 Treffer, hier wird bei grober Durchsicht festgestellt, dass die Treffer thematisch weit gefächert sind. Die Ergänzung des Suchterms in der zweiten Iteration ermöglicht die Reduzierung der Treffer. Diese werden nun mithilfe der Ein- und Ausschlusskriterien eingegrenzt, es ergeben sich fünf passende Werke. Die deutsche Version des Suchterms dagegen bringt keine weitere passende Literatur. Der erfolgreiche Suchterm wird auch in der zweiten Datenbank verwendet. Es resultieren weitere zwei Werke aus insgesamt 52 Treffern.

Nach dieser Iteration wird die Literaturrecherche beendet, da weitere Durchläufe keine neuen Inhalte produzieren. Das Thema ist für die vorliegende Fragestellung hinreichend erschlossen. Die resultierenden sieben Werke werden im nächsten Schritt beschrieben und näher analysiert, um die Unterschiede und Gemeinsamkeiten der Softwareanwendungen zu definieren.

Eine vollständige Liste der resultierenden Literatur befindet sich im Anhang 1.

3.3 Präsentation der Suchergebnisse

Aus der systematischen Literaturrecherche resultieren insgesamt sieben englischsprachige Werke, fünf Bücher und zwei Artikel. Sie sind in den Jahren von 2018 bis 2023 veröffentlicht worden und befassen sich aus unterschiedlichen Perspektiven mit der angestrebten Thematik des Vergleichs von Kafka und RabbitMQ.

Nr	Name
1	Rapid Java Persistence and Microservices. Persistence Made Easy Using Java EE8, JPA and Spring ¹⁴
2	Open-Source Publish-Subscribe Systems: A Comparative Study. In: Advanced Information Networking and Applications. Proceedings of the 36th International Conference on Advanced Information Networking and Applications (AINA-2022), Volume 1 ¹⁵
3	Learn Microservices with Spring Boot 3. A Practical Approach Using Event-Driven Architecture, Cloud-Native Patterns, and Containerization ¹⁶
4	Microservices for the Enterprise. Designing, Developing, and Deploying ¹⁷

¹⁴ Malhotra, 2019.

¹⁵ Barolli et al, 2022.

¹⁶ García et al, 2023.

¹⁷ Indrasiri, 2018.

5	Practical Event-Driven Microservices Architecture. Building Sustainable and Highly Scalable Event-Driven Microservices ¹⁸
6	Publish–Subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems ¹⁹
7	Deployment and communication patterns in microservice architectures: A systematic literature review ²⁰

Tabelle 3 Resultierende Literatur

Beschrieben und teilweise auch direkt verglichen werden die Details der Systeme, die Performance und sonstige Informationen wie beispielsweise die Entstehung oder das Konzept. Die Konzeptmatrix in Tabelle 4 stellt dar, welche Quelle welche Inhalte erläutert.

	Apache Kafka			RabbitMQ			Vergleich
	System	Performance	Sonstiges	System	Performance	Sonstiges	
Rapid Java Persistence and Microservices. Persistence Made Easy Using Java EE8, JPA and Spring	✓		✓	✓		✓	
Open-Source Publish-Subscribe Systems: A Comparative Study. In: Advanced Information Networking and Applications. Proceedings of the 36 th International Conference on Advanced	✓	✓	✓	✓	✓	✓	✓
Learn Microservices with Spring Boot 3. A Practical Approach Using Event-Driven Architecture, Cloud-Native Patterns, and Containerization	✓		✓	✓		✓	
Microservices for the Enterprise. Designing, Developing, and Deploying			✓			✓	
Practical Event-Driven Microservices Architecture. Building Sustainable and Highly Scalable Event-Driven Microservices	✓	✓	✓		✓		
Publish–Subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems	✓	✓		✓	✓		✓
Deployment and communication patterns in microservice architectures: A systematic literature review			✓			✓	

Tabelle 4 Konzeptmatrix²¹

3.4 Vergleich von Apache Kafka und RabbitMQ

Aus der resultierenden Literatur können die Erkenntnisse gewonnen werden, die eine nähere Beschreibung sowie den direkten Vergleich der beiden MOM Kafka und RabbitMQ ermöglichen.

Kafka ist eine bekannte Plattform zur Verarbeitung von Datenströmen. Sie wurde ursprünglich von LinkedIn entwickelt und gehört jetzt zur Apache Software Foundation. Die Plattform

¹⁸ Rocha, 2022.

¹⁹ Lazidis et al., 2022.

²⁰ Aksakalli et al., 2021.

²¹ Eigene Darstellung in Anlehnung an Webster/ Watson, 2002, S.17.

basiert auf Scala Java Virtual Machine (JVM)²² und verwendet ein eigenes binäres Protokoll über Transmission Control Protocol (TCP).²³

Kafka ist ein verteiltes Publish/ Subscribe Nachrichtensystem. Es bietet eine persistente Ereignisprotokoll, sodass die Ereignisse immer verfügbar sind, sogar nachdem eine Anwendung es erhalten und verarbeitet hat.²⁴ Die Daten werden dauerhaft und geordnet gespeichert und sind deterministisch lesbar.²⁵ Kafka kann dabei große Mengen an Nachrichten verwalten und erreicht dabei einen höheren Durchsatz bei besserer Performance als kurzlebige Broker, bei denen die Nachrichten nach Verwendung direkt gelöscht werden.²⁶ Jedoch hat die Semantik von Kafka meist den Nachteil höherer Kosten beim Hinzufügen neuer Broker, da Daten zwischen den Komponenten repliziert werden müssen.²⁷ Die verteilten Daten verbessern zusätzlich den Prozess des Failovers und die Skalierbarkeit.²⁸

Der Prozess von Kafka, dargestellt in Abbildung 3, startet mit dem Senden einer Nachricht, die in Themen kategorisiert wird. Die Nachrichten werden an die Warteschlangen angehängt, die im Rundlaufverfahren durchgearbeitet werden. Der Publisher wird benachrichtigt, wenn die Nachricht erfolgreich veröffentlicht wurde. Die Subscriber fragen kontinuierlich nach neuen Nachrichten. Es können mehrere Subscriber eine Nachricht lesen.²⁹

²² Vgl. Malhotra, 2019, S.296.

²³ Vgl. García et al, 2023, S.258.

²⁴ Vgl. Rocha, 2022, S.94.

²⁵ Vgl. Indrasiri, 2018, S.81.

²⁶ Vgl. Rocha, 2022, S.95.

²⁷ Vgl. Rocha, 2022, S.95.

²⁸ Vgl. Indrasiri, 2018, S.81.

²⁹ Vgl. Lazidis et al., 2022, S.7f.

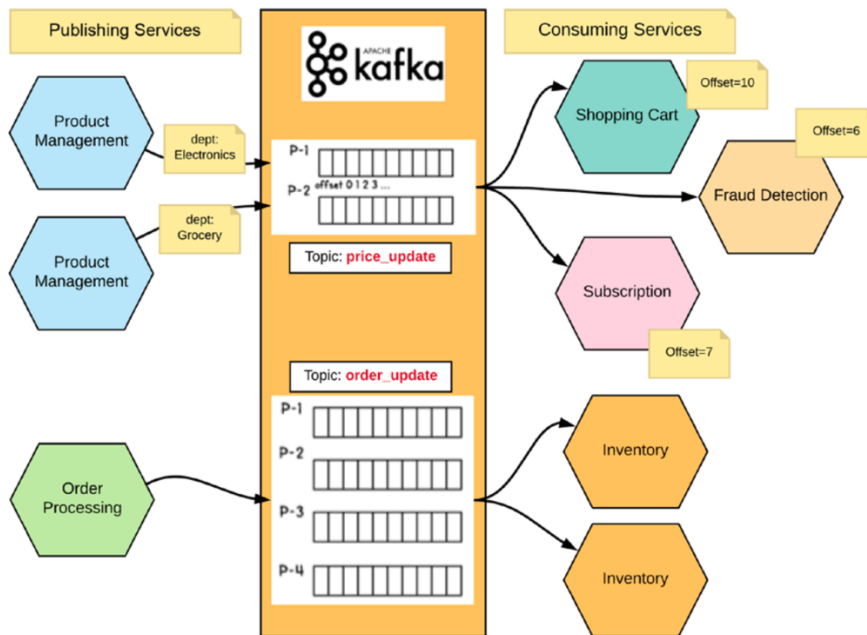


Abbildung 3 Kafka Kommunikationsprozess³⁰

RabbitMQ ist ein bekannter, open-source Nachrichtenbroker, welcher die Protokolle Advanced Message Queuing Protocol (AMQP), Message Queuing Telemetry Transport (MQTT) sowie Simple Text Oriented Message Protocol (STOMP)³¹ unterstützt. Es ist in der Programmiersprache Erlang geschrieben und dort führend. Der Prozess wird in Abbildung 4 schematisch dargestellt und beginnt mit dem Senden einer Nachricht durch den Publisher an den RabbitMQ exchange. Auf dem Routingschlüssel wird die Nachricht an die Queues verteilt, je nach Typ an eine oder mehrere. Die mit den entsprechenden Queues verbundenen Consumer bekommen so die für sie interessanten Nachrichten.³²

³⁰ Indrasiri, 2018, S.82.

³¹ Vgl. García et al, 2023, S.258.

³² Vgl. Malhotra, 2019, S.286.

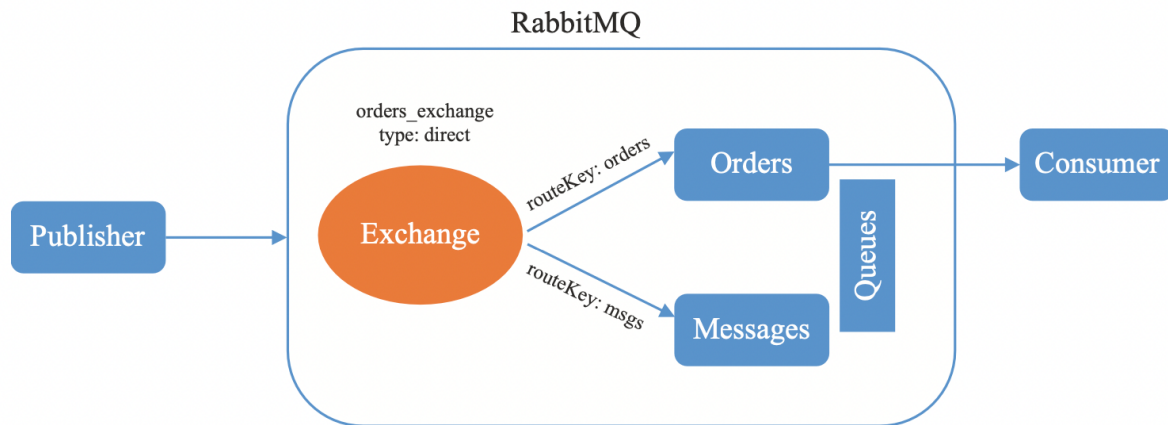


Abbildung 4 RabbitMQ Kommunikationsprozess³³

RabbitMQ hat eine destruktiv konsumierende Semantik, das heißt, die Nachrichten werden gelöscht, sobald sie einmal verwendet worden sind. Im Gegensatz zu anderen, traditionellen Nachrichtenbrokern mit kurzlebigen Nachrichten verfügt RabbitMQ darüber hinaus über die Möglichkeit, nicht-destruktive Nachrichten zu vermitteln.³⁴ RabbitMQ hat einen starken Fokus auf Punkt-zu-Punkt-Kommunikation und adressiert bestenfalls solche Consumer, die einen schnellen Durchlauf haben und die Nachrichten nicht lange speichern.³⁵

RabbitMQ ist vor allem für die Kommunikation zwischen Microservices geeignet, während Kafka für die backendseitige Nachrichtenverarbeitung bevorzugt werden sollte.³⁶

Beide MOM sind bekannte Nachrichtensysteme, die in der Industrie zur asynchronen Kommunikation weit verbreitet sind.³⁷ Sie sind open-source und gelten als state-of-the-art.³⁸ Sie ermöglichen eine verteilte und entkoppelte Verarbeitung innerhalb der Architektur.³⁹ Die lose Kopplung wird durch die asynchrone Kommunikation ermöglicht, die besonders für Status verändernde Operationen gut geeignet sind.⁴⁰ Eigenschaften beider MOM sind die hohe

³³ Malhotra, 2019, S.286.

³⁴ Vgl. Rocha, 2022, S.94.

³⁵ Vgl. Rocha, 2022, S.94.

³⁶ Vgl. Malhotra, 2019, S.296.

³⁷ Vgl. Aksakalli et al., 2021, S.11.

³⁸ Vgl. Barolli et al., 2022, S.105; vgl. Lazidis, 2022, S.2.

³⁹ Vgl. García et al, 2023, S.8.

⁴⁰ Vgl. Aksakalli et al., 2021, S.15.

Garantie, keine doppelten Nachrichten zu überbringen⁴¹, die hohe Fehlertoleranz und die vollständige Dokumentation⁴².

Der direkte Vergleich zwischen den beiden MOMs ist auch von Lazidis et al. durchgeführt worden. Der Vergleich ist dabei unterteilt worden in die Kategorien System und Performance, wobei die Performance anhand der Metriken Durchlauf und Latenz gemessen wird⁴³.

Der Vergleich der beiden Systeme ist in Tabelle 5 kompakt zusammengefasst.

Eigenschaft	Kafka	RabbitMQ
Open-Source	✓	✓
Sprache	Scala, Java	Erlang
Lernschwierigkeit	5/5	4/5
Nachrichtenformat	Byte array	Byte array JSON
Stream Processing	- Stream API	Stream API
Broker Clustering	✓	✓
Broker Federation	✓	✓
Fehlertoleranz	✓	✓
Nachrichtenabfragen	✓	-
Nachrichtenupdates	✓	-
Nachrichtenaufbewahrung	✓	✓
Nachrichtenduplikation	✓	✓
Empfangsbestätigung	✓	✓
Nachrichtenübermittlung	Pull	Push, Pull
Nachrichtenspeicherung	Disk partitions	Queue (disk, Random Access Memory (RAM))
Verbindungsverschlüsselung	Transport Layer Security / Secure Sockets Layer (TLS/SSL)	TLS/SSL
Authentication, Authorization	OAuth2.0, Kerberos, Authorization lists	Lightweight Directory Access Protocol (LDAP), Simple Authentication and Security Layer (SASL), OAuth2.0
Protokoll	Custom TCP	AMQP, STOMP, MQTT, Hypertext Transfer Protocol (HTTP)

Tabelle 5 Systemvergleich⁴⁴

⁴¹ Vgl. García et al, 2023, S.253.

⁴² Vgl. Barolli et al., 2022, S.109.

⁴³ Vgl. Lazidis, 2022, S.4.

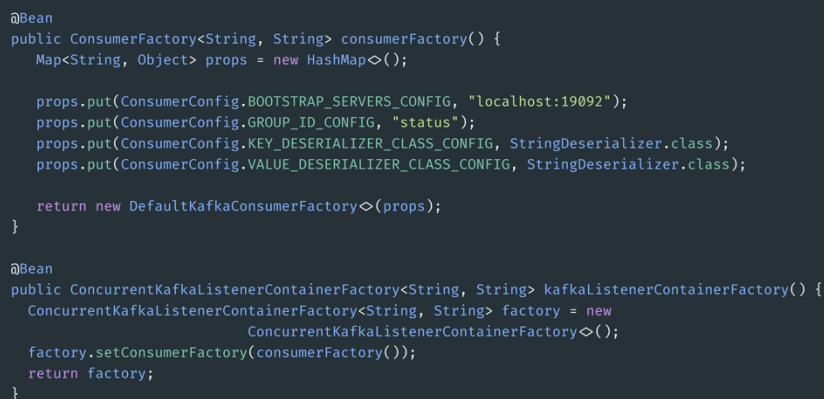
⁴⁴ Vgl. Barolli et al., 2022, S.110; vgl. Lazidis, 2022, S.13.

Verschiedene Quellen kommen bis auf zwei Punkte auf gleiche Ergebnisse, nur in den Punkten Format und Stream Processing fallen Differenzen auf, die hier entsprechend ausgewiesen werden.

Kafka funktioniert besonders schnell und hat auch bei großen Datenmengen eine sehr kurze Latenzzeit. Der hohe Grad an Parametrisierung ist hierbei ein wichtiger Faktor für die gute Performance. RabbitMQ kann zwar eine noch kürzere Latenzzeit erreichen, allerdings nur für geringere Durchlaufmengen.⁴⁵ Aus diesem Grund ist in Anwendungsfällen mit kleinen Durchlaufmengen RabbitMQ vorzuziehen. Muss eine Anwendung mit anderen Komponenten interagieren oder handelt es sich um große Datenmengen, so ist Kafka die performantere Wahl. Beide sind jedoch in Geschwindigkeit und Fehlertoleranz Spitzenreiter in ihrem Bereich.⁴⁶ Weitere Performance-Tests in der vorliegenden Literatur werden aus dieser Arbeit explizit ausgeschlossen, da die Anwendungen in den Tests synchron durchgeführt worden sind und somit nicht in den Kontext dieser Projektarbeit mit asynchroner Kommunikation passen.

4 Projektarbeit

In diesem Abschnitt werden Ausschnitte aus dem Code gezeigt. Es wird jeweils die Umsetzung von Kafka und RabbitMQ gegenübergestellt.

The image shows a screenshot of a code editor with a dark background and light-colored text. It displays two Java methods for configuring a Kafka consumer. The first method, @Bean, is consumerFactory() and returns a DefaultKafkaConsumerFactory<> with a props map containing bootstrap.servers, group.id, key.deserializer.class, and value.deserializer.class. The second method, @Bean, is kafkaListenerContainerFactory() and returns a ConcurrentKafkaListenerContainerFactory<> with the consumerFactory() as its consumerFactory().

```
@Bean
public ConsumerFactory<String, String> consumerFactory() {
    Map<String, Object> props = new HashMap<>();

    props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:19092");
    props.put(ConsumerConfig.GROUP_ID_CONFIG, "status");
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);

    return new DefaultKafkaConsumerFactory<>(props);
}

@Bean
public ConcurrentKafkaListenerContainerFactory<String, String> kafkaListenerContainerFactory() {
    ConcurrentKafkaListenerContainerFactory<String, String> factory = new
        ConcurrentKafkaListenerContainerFactory<>();
    factory.setConsumerFactory(consumerFactory());
    return factory;
}
```

Abbildung 5 Programmcode Kafka: BeanKonfiguration für den Empfänger

⁴⁵ Vgl. Barolli et al., 2022, S.111; vgl. Lazidis et al., 2022, S.14f.

⁴⁶ Vgl. Lazidis et al., 2022, S.16.

```

@Bean
public TopicExchange topic() {
    return new TopicExchange("change.status");
}

@Bean
public Queue statusQueue() {
    return new AnonymousQueue();
}

@Bean
public Binding bindingStatus(TopicExchange topic, Queue statusQueue) {
    return BindingBuilder.bind(statusQueue)
        .to(topic)
        .with("status.change.ready");
}

```

Abbildung 6 Programmcode RabbitMQ: BeanKonfiguration für den Empfänger

```

public class EventListener {

    private ITestCaseService testCaseService;

    public EventListener (ITestCaseService testCaseService) {
        this.testCaseService = testCaseService;
    }

    @KafkaListener(topics = "statusChanged", groupId = "status")
    public void listen(String message) {

        ObjectMapper mapper = new ObjectMapper();
        TestCaseT0[] testCaseListeT0Array = null;

        try {
            testCaseListeT0Array = mapper.readValue(message, TestCaseT0[].class);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }

        Collection<TestCaseT0> testCaseListeT0 = Arrays.asList(testCaseListeT0Array);

        if (!testCaseService.processTestCases(testCaseListeT0))
            System.out.println("Fehler!");
    }
}

```

Abbildung 7 Programmcode Kafka: EventListener für den Empfänger

```

public class EventListener {

    private ITestCaseService testCaseService;

    public EventListener (ITestCaseService testCaseService) {
        this.testCaseService = testCaseService;
    }

    @RabbitListener(queues = "#{statusQueue.name}")
    public void listen(String message) {

        if (event.equals("statusChanged")) {

            ObjectMapper mapper = new ObjectMapper();
            TestCaseT0[] testCaseListeT0Array = null;

            try {
                testCaseListeT0Array = mapper.readValue(message, TestCaseT0[].class);
            } catch (JsonProcessingException e) {
                e.printStackTrace();
            }

            Collection<TestCaseT0> testCaseListeT0 = Arrays.asList(testCaseListeT0Array);

            if (!testCaseService.processTestCases(testCaseListeT0))
                System.out.println("Fehler!");
        }
    }
}

```

Abbildung 8 Programmcode RabbitMQ: EventListener für den Empfänger

```

@Bean
IMessageQueue messageQueue(KafkaTemplate<String, String> kafkaTemplate) {
    return new QueueAdapter(kafkaTemplate);
}

@Bean
public KafkaAdmin kafkaAdmin() {
    Map<String, Object> configs = new HashMap<>();
    configs.put(AdminClientConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:19092");
    return new KafkaAdmin(configs);
}

@Bean
public NewTopic topic() {
    return TopicBuilder.name("statusChanged")
        .partitions(1)
        .replicas(3)
        .compact()
        .build();
}

@Bean
public ProducerFactory<String, String> producerFactory() {
    Map<String, Object> configProps = new HashMap<>();

    configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:19092");
    configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
    configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);

    return new DefaultKafkaProducerFactory<>(configProps);
}

@Bean
public KafkaTemplate<String, String> kafkaTemplate() {
    return new KafkaTemplate<>(producerFactory());
}

```

Abbildung 9 Programmcode Kafka: BeanKonfiguration für den Sender

```

@Bean
IMessageQueue messageQueue(AmqpTemplate amqpTemplate) {
    return new QueueAdapter(amqpTemplate);
}

@Bean
public TopicExchange topic() {
    return new TopicExchange("change.status");
}

```

Abbildung 10 Programmcode RabbitMQ: BeanKonfiguration für den Sender

```

public class QueueAdapter implements IMessageQueue{

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public QueueAdapter (KafkaTemplate<String, String> kafkaTemplate) {
        this.kafkaTemplate = kafkaTemplate;
    }

    public boolean send(DomainEvent domainEvent) {

        String payload = "";

        if (domainEvent.getEvent().equalsIgnoreCase("statusChanged")) {

            UserStory userStory = (UserStory)domainEvent.getObject();
            Collection<TestCaseT0> testCaseList = new ArrayList<TestCaseT0>();

            for (TestCase testCase : userStory.getAllTestCases())
                testCaseList.add(new TestCaseT0(testCase));

            ObjectMapper objectMapper = new ObjectMapper();

            try {
                payload = objectMapper.writeValueAsString(testCaseList);
            } catch (JsonProcessingException e) {
                e.printStackTrace();
                return false;
            }

            System.out.println("Payload: " + payload);

            kafkaTemplate.send(domainEvent.getEvent(), payload);
            return true;
        }
    }
}

```

Abbildung 11 Programmcode Kafka: Message Queue

```

public class QueueAdapter implements IMessageQueue{

    public final String key = "status.change.ready";

    @Autowired
    private final AmqpTemplate amqpTemplate;

    @Autowired
    private TopicExchange topic;

    public QueueAdapter (AmqpTemplate amqpTemplate) {
        this.amqpTemplate = amqpTemplate;
    }

    public boolean send(DomainEvent domainEvent) {

        String payload = "";

        if (domainEvent.getEvent().equalsIgnoreCase("statusChanged")) {

            UserStory userStory = (UserStory)domainEvent.getObject();
            Collection<TestCaseT0> testCaseList = new ArrayList<TestCaseT0>();

            for (TestCase testCase : userStory.getAllTestCases())
                testCaseList.add(new TestCaseT0(testCase));

            ObjectMapper objectMapper = new ObjectMapper();

            try {
                payload = objectMapper.writeValueAsString(testCaseList);
            } catch (JsonProcessingException e) {
                e.printStackTrace();
                return false;
            }

            System.out.println("Payload: " + payload + "\n Topic-Name: " + topic.getName() + "\nKey: " + key);

        }

        amqpTemplate.convertAndSend(topic.getName(), key, domainEvent.getEvent() + "/" + payload);
        return true;
    }
}

```

Abbildung 12 Programmcode RabbitMQ: Message Queue

5 Fazit

Kafka und RabbitMQ sind bekannte und in der Industrie vertretene MOM, die sich etabliert haben. In vielen Qualitätsmerkmalen ähnlich unterscheiden sie sich jedoch in ihrem Konzept, ihrer Umsetzbarkeit und Umsetzung sowie ihrer Performance. Daraus ist weniger eine Rangordnung zu schließen als die verschiedenen, optimalen Anwendungsfälle, in denen sie mit ihren besonderen Eigenschaften optimal zu nutzen sind.

Literaturverzeichnis

- Aksakalli, Işıl Karabey/ Çelik, Turgay/ Can, Ahmet Burak/ Tekinerdoğan, Bedir (2021): Deployment and communication patterns in microservice architectures: A systematic literature review, in: The Journal of Systems & Software 180 (2021).
- AWS (o.D.): Was ist der Unterschied zwischen Kafka und RabbitMQ?, <https://aws.amazon.com/de/compare/the-difference-between-rabbitmq-and-kafka/#:~:text=RabbitMQ%20has%20low%20latency.,millions%20of%20messages%20per%20second.&text=RabbitMQ%20supports%20a%20broad%20range%20of%20languages%20and%20legacy%20protocols>, Zugriff am 14.01.2024.
- Dillmann, Nicolas (2014): MoR: Messaging – RabbitMQ & Apache Kafka, <https://www.zweitag.de/blog/mor-messaging-rabbitmq-apache-kafka>, Zugriff am 14.01.2024.
- Elsevier (o.D.): ScienceDirect: Elseviers Plattform für peer-reviewed wissenschaftliche Literatur, <https://www.elsevier.com/de-de/products/sciencedirect>, Zugriff am 14.01.2024.
- García, Moisés Macero/ Telang, Tarun (2023): Learn Microservices with Spring Boot 3. A Practical Approach Using Event-Driven Architecture, Cloud-Native Patterns, and Containerization, 3. Aufl., apress.
- Hanser (o.D.): Willkommen in der HANSER eLibrary, <https://www.hanser-elibrary.com>, Zugriff am 14.01.2024.
- Indrasiri, Kasun/ Siriwardena, Prabath (2018): Microservices for the Enterprise. Designing, Developing, and Deploying, apress.
- Lazidis, Apostolos/ Petrakis, Euripides G. M./ Chouliaras, Spyridon/ Sotiriadis, Stelios (2022): Open-Source Publish-Subscribe Systems: A Comparative Study, in: Advanced Information Networking and Applications. Proceedings of the 36th International

Conference on Advanced Information Networking and Applications (AINA-2022), Hrsg.: Barolli, Leonard/ Hussain, Farookh/ Enokido, Tomoya, Volume 1, Springer, S.105-115.^[a]

Lazidis, Apostolos/ Tsakos, Konstantinos/ Petrakis, Euripides G.M. (2022): Publish–Subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems, in: Internet of Things 19 (2022).^[b]

Lempert, Sebastian (2021): IoT-Software-Plattformen. Methoden zur Bewertung und Auswahl der am besten geeigneten Plattform, Wiesbaden, Springer Gabler.

Malhotra, Raj (2019): Rapid Java Persistence and Microservices. Persistence Made Easy Using Java EE8, JPA and Spring, apress.

Rocha, Hugo Filipe Oliveira (2022): Practical Event-Driven Microservices Architecture. Building Sustainable and Highly Scalable Event-Driven Microservices, apress.

SAP (o.D.): Was ist eine ereignisgesteuerte Architektur?, <https://www.sap.com/swiss/products/technology-platform/what-is-event-driven-architecture.html>, Zugriff am 14.01.2024.

SpringerLink (o.D.): Willkommen bei SpringerLink, <https://www.springer.com/de/hilfe/about-springerlink/18548>, Zugriff am 14.01.2024.

TechSmith (o.D.): Synchrone und asynchrone Kommunikation: Vorteile, Nachteile und Beispiele für die Anwendung, <https://www.techsmith.de/blog/asynchrone-kommunikation/#:~:text=allerorts%20zu%20finden.-,Was%20ist%20asynchrone%20Kommunikation%3F,sind%20und%20verfuegbar%20sein%20muessen>, Zugriff am 14.01.2024.

Vom Brocke, Jan/ Simons, Alexander/ Riemer, Kai/ Niehaves, Bjoern/ Plattfaut, Ralf/ Cleven, Anne (2015): Standing on the Shoulders of Giants: Challenges and Recommendations of Literature Search in Information Systems Research, in: Information Systems Research. Communications of the Association for Information Systems, Jg. 37, Nr.9, S. 205-224.

WalkingTree Technologies (2018): Inter-service communication in Microservices,
<https://walkingtreetech.medium.com/inter-service-communication-in-microservices-c54f41678998>, Zugriff am 14.01.2024.

Anhang

Anhang 1: Ausgewertete Literatur

Nr.	Titel	Autor(en)	Jahr
1	Rapid Java Persistence and Microservices. Persistence Made Easy Using Java EE8, JPA and Spring	Raj Malhotra	2019
2	Open-Source Publish-Subscribe Systems: A Comparative Study. In: Advanced Information Networking and Applications. Proceedings of the 36 th International Conference on Advanced Information Networking and Applications (AINA-2022), Volume 1	Apostolos Lazidis, Euripides G. M. Petrakis, Spyridon Chouliaras, Stelios Sotiriadis; Hrsg.: Leonard Barolli, Farookh Hussain, Tomoya Enokido	2022
3	Learn Microservices with Spring Boot 3. A Practical Approach Using Event-Driven Architecture, Cloud-Native Patterns, and Containerization	Moisés Macero García, Tarun Telang	2023
4	Microservices for the Enterprise. Designing, Developing, and Deploying	Kasun Indrasiri, Prabath Siriwardena	2018
5	Practical Event-Driven Microservices Architecture. Building Sustainable and Highly Scalable Event-Driven Microservices	Hugo Filipe Oliveira Rocha	2022
6	Publish–Subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems	Apostolos Lazidis, Konstantinos Tsakos, Euripides G.M. Petrakis	2022
7	Deployment and communication patterns in microservice architectures: A systematic literature review	Işıl Karabey Aksakalli, Turgay Çelik, Ahmet Burak Can, Bedir Tekinerdoğan	2021