## Argumentatie

### Algemeen

JMS is een technologie dat asynchrone requests behandeld. Er zijn verschillende toepassingen die gebruikt kunnen worden als JMS-broker zoals: ActiveMQ, Kafka en RabbitMQ.

Deze voorzien asynchrone communicatie en ontkoppelen processen, consumer en producer worden verwijderd van het bericht. De benamingen hiervoor zijn Message Queues, Message Brokers & Messaging tools. ActiveMQ, Kafka en rabbitMQ voorzien dan ook hetzelfde maar hebben andere voor- en nadelen. (1)

Verder hanteert JMS het message-oriented middleware principe, waarbij de data volgens een bepaald formaat opgestuurd dient te worden en via een vast formaat wordt verspreid over alle consumers. (2)

### Asynchrone communicatie

JMS maakt gebruik van asychrone communicatie doormiddel van Message Queue’s. Dit zorgt voor hoge performance en hoge doorvoer van data. Hierdoor is het mogelijk om messages of topics alvast te versturen voor de consumer en vervolgens de messages te verwerken zodra de consumer beschikbaar is. (3)

### Voorbeeld

Als werkzoekende ga je er niet vanuit dat alle bedrijven gelijk reageren, maar dat hier een paar dagen overheen kunnen gaan. Je wilt dan niet dat de werkzoekende niet in staat is om zijn applicatie af te sluiten. Doordat de communicatie asynchroon gaat, zal de werkzoekende gewoon verder kunnen gaan met zijn normale werkzaamheden, en kan hij de applicatie afsluiten zonder dat de message verloren gaat.

Indien synchrone communicatie gebruikt zou worden, dan zou de producer moeten wachten totdat alle consumers zijn message hebben ontvangen. In die tussentijd zou je dus niets kunnen doen met de applicatie, terwijl het dagen kan duren voordat een bedrijf echt reageert.

### Topics vs Queues

Een queue implementeert een load balancer principe. Een bericht wordt verstuurd naar precies 1 “consumer”. Als er geen “consumers” beschikbaar zijn op het moment van versturen dan zal het bericht behouden worden door de broker totdat “consumer” beschikbaar is. Bij een Queue is de “consumer” bekend. (4)

een topic implementeerd een“publish” en “subscribe” principe. Wanneer je een bericht “published” dan zal het naar alle “subscribers” gaan die interesse hebben (welke op het topic “gesubscribed” zijn). Nul of meer “subscribers” ontvangen een kopie van de berichten in the topic. Alleen “subscribers”, die al “gesubscribed” waren op het moment van versturen, krijgen het berichtbinnen. Voor een topic zijn de “subscribers” niet bekend. (4)

### Voorbeeld

Een werkzoekende wil zijn CV versturen naar verschillende bedrijven om zo een aan werk te komen. Hierbij wordt er gebruik gemaakt van Topics waar de werkzoekende naartoe published.

Topics zijn handig om te gebruiken doordat de message verstuurd kan worden naar meerdere subscribers (bedrijven), terwijl er bij een Queue maar een subscriber kan zijn.

Indien er dus meerdere subscribers zijn is het aan te raden om Topics te gebruiken, terwijl bij een een-op-een communicatie er beter een queue gebruikt kan worden.

### Message Filters

Een message Filter zorgt ervoor dat een consumer or subscriber alleen berichten binnen krijgen die voor hun van toepassing zijn. Een message filter kijkt of een message voldoet aan bepaalde criteria. Als deze voldoet aan de criteria dan zal deze verstuurd worden, als deze niet voldoet aan de gestelde criteria dan zal deze niet verstuurd worden naar de subscribers van het specifieke channel of topic. (5)

### Voorbeeld

Een bedrijf is alleen geïnteresseerd naar java software engineers. Deze zal zich dan ook subscriben op een topic die de criteria voor java en software engineer heeft staan. Dit bedrijf wil geen werkzoekende te zien krijgen die een baan als schoonmaker wil hebben.

Het message filter zorgt er dan ook voor dat alleen maar werkzoekende zijn die voldoen aan java software developers. Als er een werkzoekende voorbij komt die iets zoekt in de horeca, dan zal deze niet verstuurd worden in dit topic, maar in een ander topic dat wel de criteria voor “horeca” heeft staan.

### Load balancing

Load balancing is een techniek om werk te verdelen tussen verschillende computers. Op deze manier kunnen processen, harde schijven en andere hulpbronnen optimaal gebruikt worden en zal dit de benodigde verwerkingstijd verminderen. Als er een bericht binnenkomt dan zal de load balancer beslissen welke server in de pool beschikbaar is. Het zal het bericht vervolgens door de gekozen server laten verwerken. Doordat de load balancer de berichten verdeeld over de servers die beschikbaar zijn is het gebruik van een loadbalancer vooral goed wanneer er zwaar verkeer is. (6)

## Beveiliging

ActiveMQ heeft ondersteuning voor JAAS (Java Authentication and Authorization Service). Via configuratie bestanden kan er een gebruikersnaam en wachtwoord opgezet worden zodat niet iedereen toegang heeft tot de broker. Met behulp van een Access Control List kunnen permisies uitgedeeld worden binnen activeMQ voor geautoriseerde gebruikers. (7)

Kijkend naar de vraag-aanbod applicatie zal dit niet van toepassing zijn. Het gaat namelijk over publieke communicatie tussen werkzoekende en bedrijven. Beide zullen gemakkelijk het systeem moeten gebruiken, en niet in afwachting moeten zijn van een administrator die ze toegang tot het systeem geeft.

## Interoperabiliteit

Als 2 applicaties verschillende messaging providers gebruiken, dan kunnen ze alsnog gemakkelijk met elkaar communiceren met elkaar via JMS vanwege de hoge interoperabiliteit. (3)

### JMS vs Alternatieven

Alternatieven voor JMS zijn AMQP, MQTT en NMS. Allen zijn vergelijkbaar maar hebben bepaalde voordelen en nadelen in vergelijking met JMS. (8)

#### NMS

NMS is de vergelijkbare variant van JMS maar dan voor C# .net. Doordat dit voor C# is en niet voor java is dit verder niet van toepassing voor de vraag-aanbod applicatie.

#### Advanced Message Queueing Protocol (AMQP)

AMQP is ook vergelijkbaar met JMS en NMS maar probeert het compatibiliteit probleem tussen verschillende talen op te lossen. Gebruik makend van AMQP zal het mogelijk moeten zijn dat een java applicatie kan communiceren met een .net applicatie via messaging communicatie.

#### Message Queueing Telemerty Transport (MQTT)

Niet alle apparaten kunnen AMQP handelen. Hiervoor is MQTT een lichter alternatief dat gebruikt kan worden door hardware met mindere specs. Hierdoor wordt MQTT vooral gebruikt door IoT-apparaten.

## ActiveMQ vs Alternatieven

ActiveMQ

* Guaranteed delivery (9)
* Jaas ondersteuning (10)
* Gemakkelijk te gebruiken door EIP.

RabbitMQ

* reliable delivery (11)
* Problemen met clustering waaronder message loss en handmatige afstemming na partitionering. (12)
* SASL ondersteuning voor java (13)
* JMS plugin vereist. (14)

Kafka

* Distributed
* Clustering hoge prioriteit
* Zookeeper vereist
* Complexe architectuur
* Reliable delivery

WSO2 Message Broker

* Guaranteed delivery with failover message store and scheduled failover message forwarding processor. (15)
* Vereist WSO2 platform
* Vrij groot en complex systeem
* Ondersteuning met Java, C#, C++, PHP, Ruby, Erlang en meer (16)
* Snelle message delivery (16)

# Advies JMS

JMS kent veel voordelen bij het opzetten van de communicatie in gedistribueerde systemen, zoals:

* Asynchrome communicatie
* Consumer hoeft niet online te zijn op het tijdstip van verzenden
* Berichten kunnen naar verschillende systemen verstuurd worden die los van elkaar staan.
* De broker kan beveiligd worden, en daardoor afgesloten worden van ongeautoriseerde systemen.
* Indien er veel verkeer is wordt er gebruik gemaakt van load balancing, zodat berichten efficiënter verstuurd worden.
* Gebruik van Topics. Hierdoor hoeven consumers niet te weten wie het ontvangt, en krijgen subscribers alleen de berichten uit dit specifieke topic.
* Er kan gebruik gemaakt worden van message filters, zodat subscribers alleen de berichten krijgen waarop ze “gesubscribed” zijn.

# Advies JMS-Brokers

Lettend op JMS-brokers zoals ActiveMq, RabbitMq, Kafka en Wso2 en de opgestelde non-functional voor guaranteed delivery is de volgende conclusie te trekken:

Alle brokers hebben ondersteuning voor guaranteed delivery op de een of ander manier.

Hierbij heeft ActiveMq volledige ondersteuning voor guaranteed delivery in de basis.

Kafka gebruikt Zookeeper voor de ondersteuning van reliable delivery, wat er voor zorgt dat het op de een of andere manier (uiteindelijk) bij minimaal een server uit komt, en deze eventueel op een later moment verstuurd naar overige servers.

RabbitMq heeft ook reliable delivery ondersteuning, maar heeft jammer genoeg alleen JMS ondersteuning via een plugin.

De wso2 message broker heeft een zeer geavanceerde ondersteuning voor gauranteed delivery, met een failover een een scheduler implementatie. Wso2 lijkt alleen een stuk complexer te zijn dan de overige brokers, en vereist zover bekend de wso2 message broker het wso2 platform.

Kijkend op de guaranteed delivery dan is de WSO2 message broker de beste keuze. Dit omdat het zowel guaranteed delivery heeft als een snelle message delivery. Jammer genoeg is het gebruik van WSO2 binnen dit project niet haalbaar door de complexiteit van WSO2. Daardoor moet er gekeken worden naar een haalbaar alternatief, wat activeMQ is. activeMQ is makkelijker in gebruik dan WSO2 door de EIP ondersteuning, en de grote van activeMQ.

# Verwijzingen

1. **Stackshare.** RabbitMQ vs Kafka vs ActiveMQ. *Stackshare.* [Online] https://stackshare.io/stackups/activemq-vs-kafka-vs-rabbitmq.

2. **Erdogan, Levent.** Concepts and Fundamentals of JMS Programming. *InformIT.* [Online] December 2002. http://www.informit.com/articles/article.aspx?p=30308.

3. **j2eereference.** Advantages of Java Message Service (JMS). *j2eereference.* [Online] april 2017. https://j2eereference.com/advantages-java-message-service-jms/.

4. **Stackoverflow.** JMS Topic vs Queues. *Stackoverflow.* [Online] April 2011. https://stackoverflow.com/questions/5576415/jms-topic-vs-queues.

5. **enterpriseintegrationpatterns.com.** Message Filter. *enterpriseintegrationpatterns.* [Online] https://www.enterpriseintegrationpatterns.com/patterns/messaging/Filter.html.

6. **Rouse, Margaret.** load balancing. *TechTarget.* [Online] Oktober 2018. https://searchnetworking.techtarget.com/definition/load-balancing.

7. **Apache.** Security. *Apache ActiveMQ.* [Online] http://activemq.apache.org/security.html.

8. **Fernando, Chanaka.** Comparison of Asynchronous Messaging Technologies: JMS, AMQP, and MQTT. *Dzone.* [Online] Maart 2016. https://dzone.com/articles/comparison-of-asynchronous-messaging-technologies.

9. **Apache.** Guarantees of sends and commits. *ActiveMQ Docs.* [Online] https://activemq.apache.org/artemis/docs/1.0.0/send-guarantees.html.

10. —. Security. *ActiveMQ.* [Online] http://activemq.apache.org/security.html.

11. **RabbitMQ.** Reliability Guide. *RabbitMQ.* [Online] https://www.rabbitmq.com/reliability.html.

12. **Duprat, Antoine.** How to choose a Message Queue. *Medium.* [Online] February 2019. https://medium.com/linagora-engineering/how-to-choose-a-message-queue-247dde46e66c.

13. **RabbitMq.** Authentication, Authorisation, Access Control. *RabbitMq.* [Online] https://www.rabbitmq.com/access-control.html.

14. —. RabbitMQ JMS Client. *RabbitMq.* [Online] https://www.rabbitmq.com/jms-client.html.

15. **wso2.** Guaranteed Delivery with Failover Message Store and Scheduled Failover Message Forwarding Processor. *WSO2 docs.* [Online] https://docs.wso2.com/display/ESB490/Guaranteed+Delivery+with+Failover+Message+Store+and+Scheduled+Failover+Message+Forwarding+Processor.

16. **WSO2.** Features. *wso2.* [Online] https://wso2.com/products/message-broker/#Features.

17. **Mahmoud, Qusay H.** Getting Started with Java Message Service (JMS). *Oracle.* [Online] November 2004. https://www.oracle.com/technetwork/articles/java/introjms-1577110.html.