Message Broker

Bustopologie am Beispiel von RabbitMQ

Informationsmanagement u. Big Data SS20
Jan Löwenstrom (34937)

09.07.2020

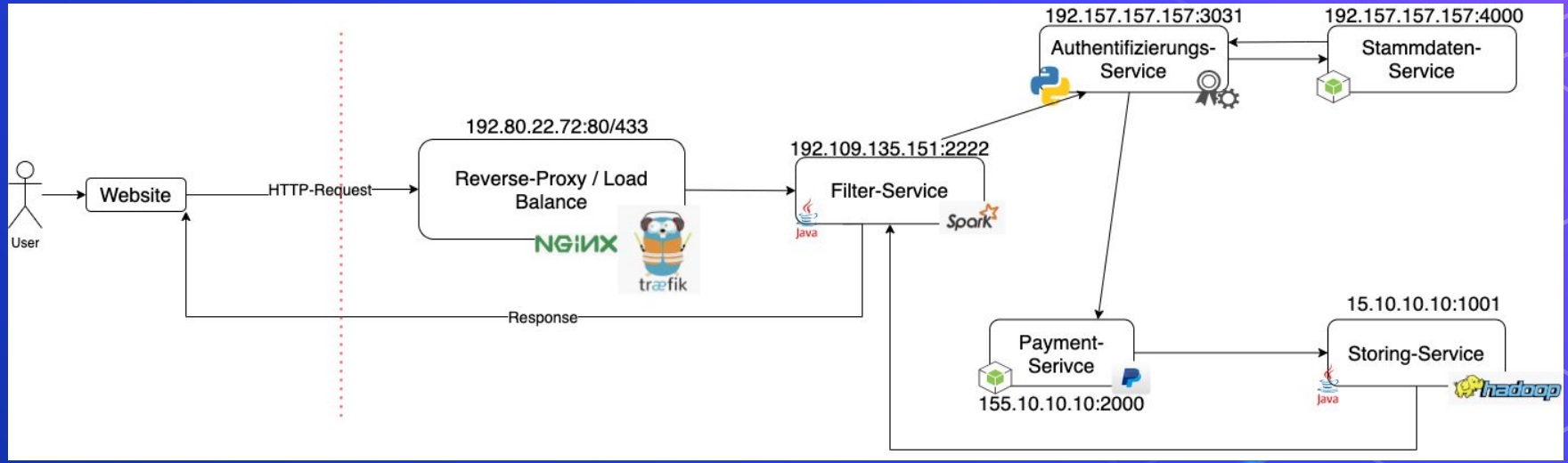
Gliederung

1. Szenarien
2. Topologien
3. RabbitMQ
4. Bottleneckanalyse
5. Fazit

1. Einsatzszenario

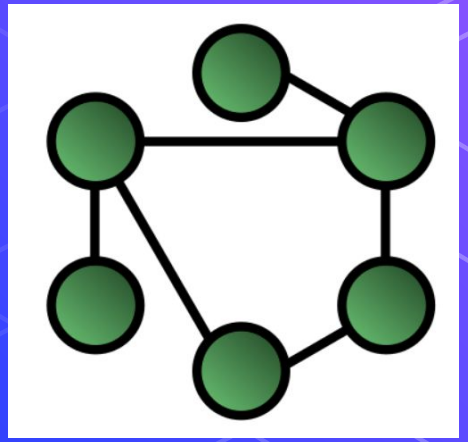
Kommunikation in einer Microservice-Architektur

1. Szenario - Buchungswebsite



2. Punkt zu Punkt mittels HTTP

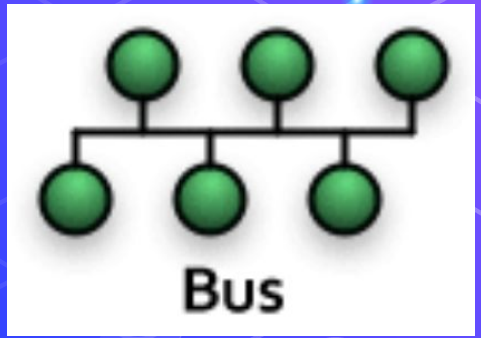
- RESTful HTTP API
- Blocking
- Failover und Retry-Logik für jeden Req.
- Einschätzung über Response-Time
- Leichte Fehlersuche
- Kenntnisse über Infrastruktur nötig
- Service Discovery Server (Netflix Eureka)



Topologie: Vermaschtes Netz *(mesh)*

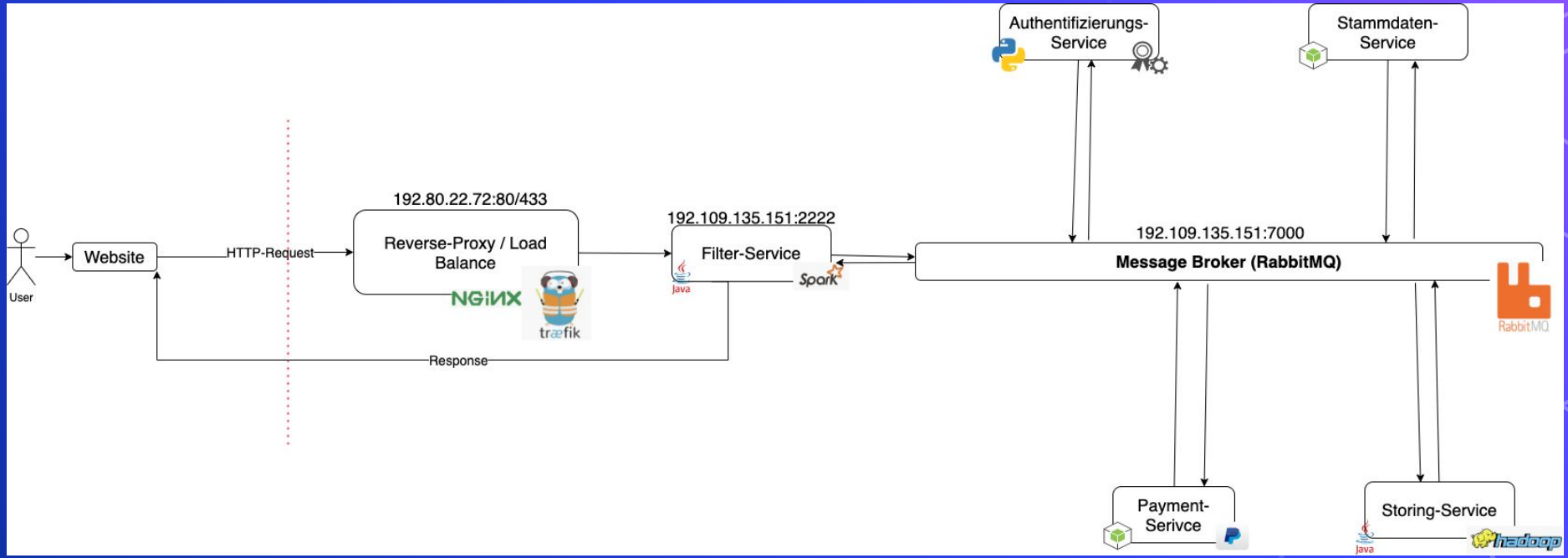
2. Bustopologie mittels Message Broker

- Zentrales Übertragungsmedium
  Message Broker als Data-Bus
- Asynchron (Entkopplung Request und Response)
- Eventbasiert
- Broker übernimmt Load Blancing, Retry-Logik, Failover, etc.



Topologie: Bus

2. Buchungswebsite mit RabbitMQ



3. Rabbit MQ

“Messaging that just works”

3. RabbitMQ

- Open-Source
- Implementiert AMQP (Advanced Message Queuing Protocol)
- Einfache Installation
- Relativ schlechte Überwachung
- Cluster Support
- Dead-Letter-Queue, Auto-Requeue, Dockerized, Management UI

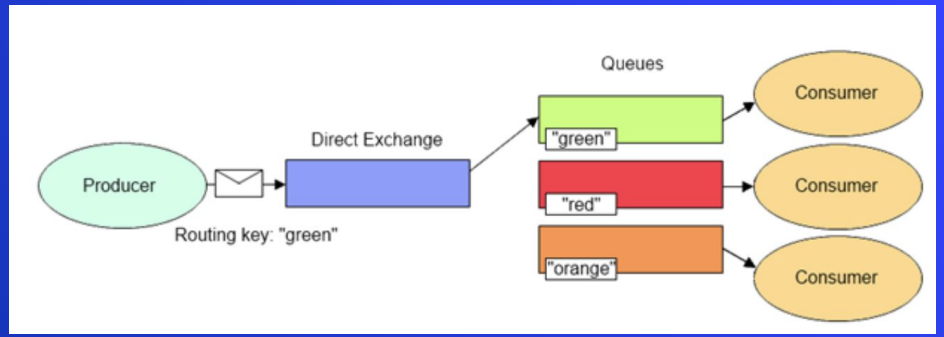
3. RabbitMQ - Komponenten

- Message: Paket an Informationen mit Header (Key-value pairs), Body im Binärformat
- Producer/Publisher: Service, der Messages sendet
- Consumer/Subscriber: Service, der Nachrichten erhält

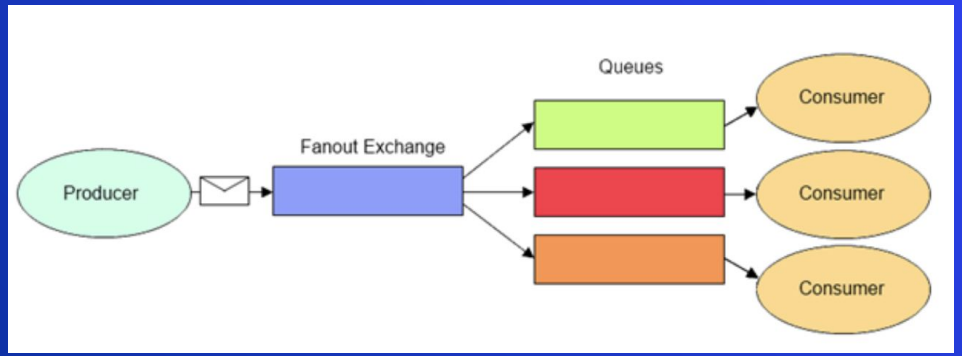
3. RabbitMQ - Komponenten

- Queue: Buffer, der Nachrichten hält (wahlweise RAM, Storage oder beides)
- Exchange: Abstraktion von Queues. Verteilt Nachrichten an bestimmte Queues;
  Types = Direct, Fanout, Topic, Headers

3. RabbitMQ - Exchange Typen

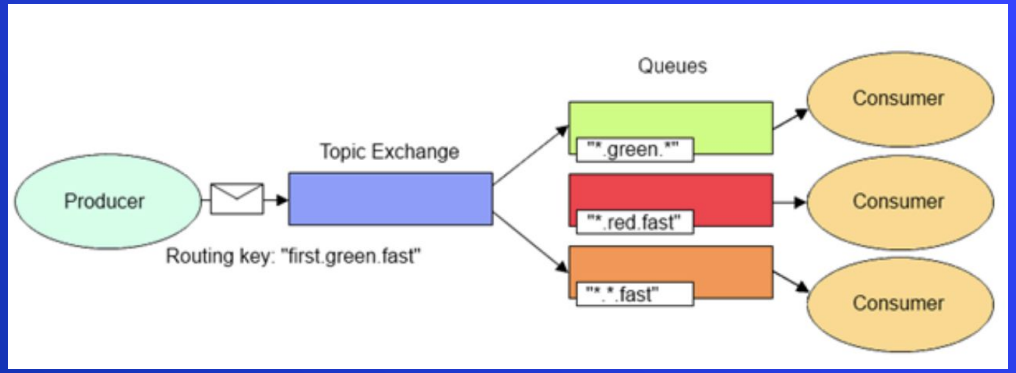


Direct



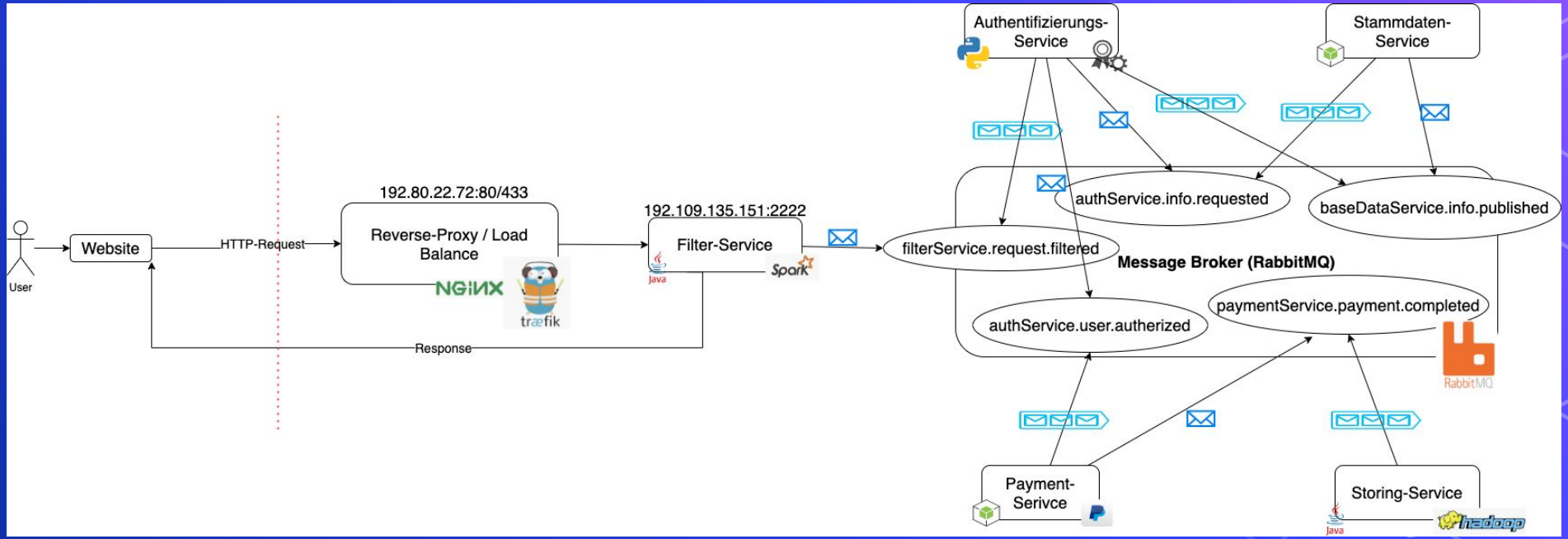
Fanout

3. RabbitMQ - Exchange Typen



Topic

3. Buchungswebsite detailliert



4. Bottleneckanaylse

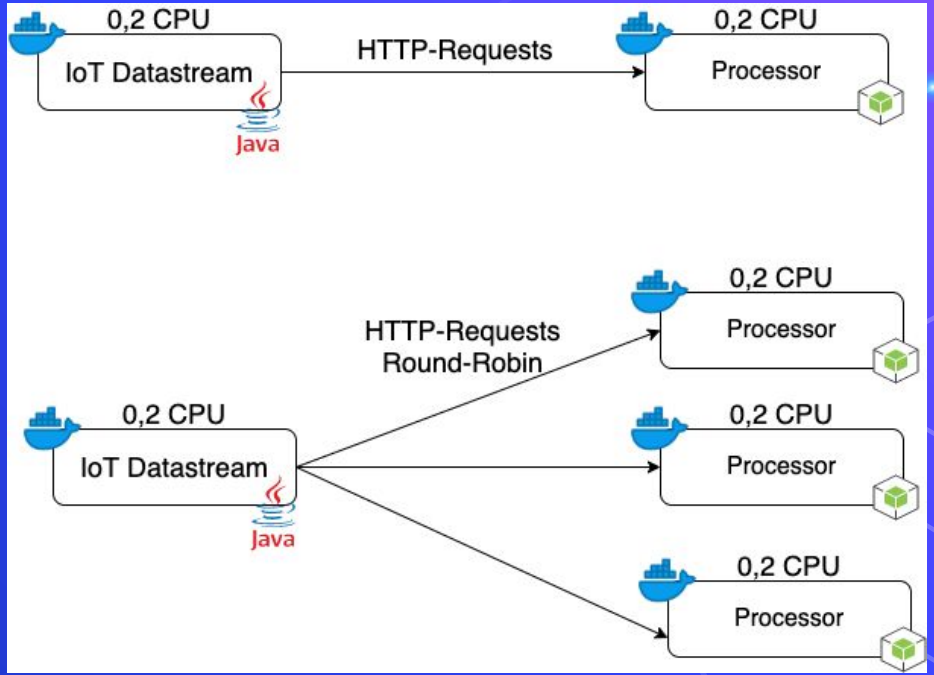
Wozu das Ganze?

4. Bottlenackanalyse

- Probleme bei (Blocking)-HTTP:
- Variable RAM Auslastung durch Warten auf Responses
- Failover bei Hochskalierung
- Potentieller Verlust von Daten
- Missverhältnis der Geschwindigkeiten bei Producer und Processor

4. Bottleneckanalyse

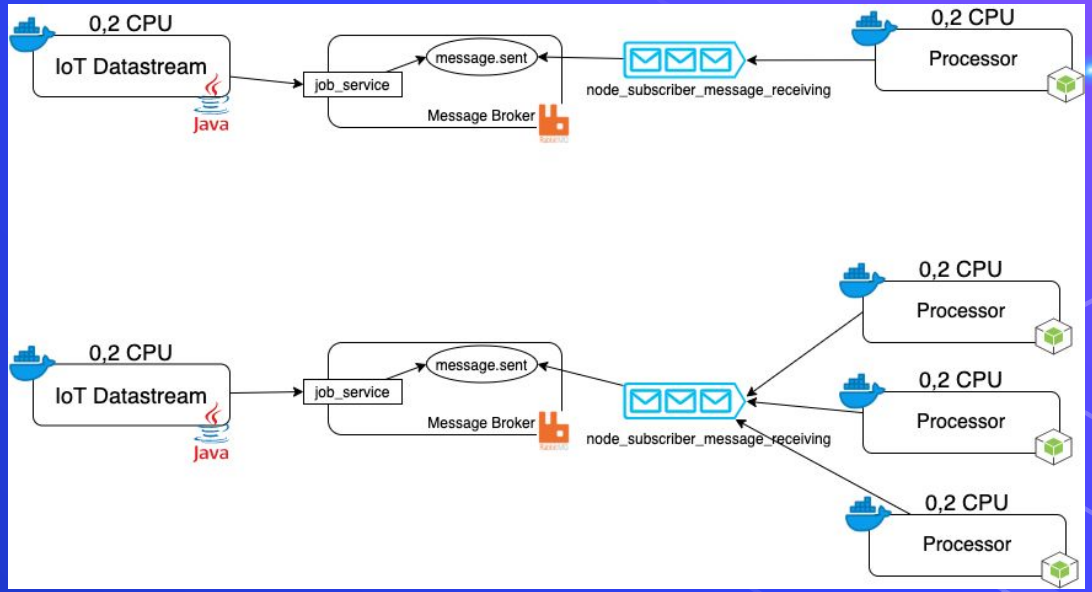
- Sende X HTTP Request mit Daten an HTTP-Endpunkt
- Intervall von X Millisekunden
- ThreadPoolExecutor mit 100 laufenden Threads
- LinkedBlockingQueue sammelt ausstehende Requests
- NodeJS Service berechnet 200 UUIDs bei jeder Request und sendet dann Response zurück



=> Wie groß wird die Queue-Size?

4. Bottleneckanalyse

- Sende X Nachrichten an Exchange “job_service” mit Topic “message_sent”
- Intervall von X Millisekunden
- Warte auf Publisher-Confirm



4. Bottleneckanalyse

Setting (10k requests, 1ms interval)	**max-queue-size**	**Max-response-time**	**Total time needed**
HTTP - 1 Processor	5800	64s	120s / 122s
HTTP - 3 Processors	~20	0.2s	90s / 91s
RabbitMQ - 1 Subscriber	~20	0.01s	39s / 69s
RabbitMQ - 3 Subscribers	~20	0.01s	39s / **39s**

5. Fazit

- Flexibilität durch topic-Exchanges
- Teilweise Auslagerung der Fehlerlogik
- Zentralisierung des Load-Balancing
- Sehr gut geeignet für skalierende Microservice-Infrastrukturen
- Hohe Ausfallsicherheit (Cluster, Cold-Storage, Publisher-Confirm)

Bitte!

Bei Fragen:

jan.loewenstrom@studenten.hs-bremerhaven.de