
BREAKING:
Message brokers
"Why is the guy who invests all your money called a broker?"

Message brokers

Een "message broker" is een "intermediair" tussen zender(s) en ontvanger(s).

Implementatie middels een message protocol:

- Een protocol legt de service vast: hoe om te gaan met berichten, het formaat van berichten, connecties, queues, acknowledgements, etc..
- Daardoor een betere (gestandaardiseerde) samenwerking tussen zender(s), broker en ontvanger(s)

De bekendste message protocollen zijn:

- MQTT (Message Queuing Telemetry Transport, (IBM 1999))
- AMQP (Advanced Message Queuing Protocol, (J.P. Morgan Chase, 2003))

Bekende brokers:

- Mosquitto, HiveMQ (MQTT)
- RabbitMQ (AQMP, MQTT, …)
- Kafka (eigen binary protocol)

Waarom een message broker

- **Los koppelen** van (producers en receivers) services. Zorgt voor onafhankelijkheid van je services
- **A-synchrone** communicatie, snellere respons
- **Horizontaal schaalbaar**: meerdere ontvanger bij veel berichten
- **Betrouwbaar**: berichten kunnen worden opgeslagen.
- "Quality of Service" (QoS) levels
- **Flexibel**: werken met verschillende topics / queues
- **Interoperabiliteit**: veel ondersteunde talen, cloud omgevingen, micro services

Waarom een message broker - praktijk van een webshop

Klant plaatst een bestelling.

Zonder message broker:

- De code op de webserver gaat aan de slag om dit te verwerken (betaling, voorraad bijwerken, factuur maken, …)
- **Pas nadat** de verwerking is gelukt volgt een bevestiging.
- Bij ergens een fout: alles opnieuw

Met een message broker:

- De bestelling wordt naar de message broker gestuurd en de klant krijgt **direct** een bevestiging van de bestelling
- De queue met de bestelling wordt verwerkt en eventuele notificaties volgen
- Bij ergens een fout: bestelling blijft in de queue tot de fout is hersteld
- Loskoppeling aanvraag en verwerking

Nog een voorbeeld: een "smart city"

Diverse slimme sensoren (bv. voor verkeersdrukte, luchtkwaliteit, neerslag, temperatuur, …) die veel data produceren wat verwerkt moet worden

Zonder message broker:

- overbelasting centrale verwerkings-server?
  - Kans op data-verlies
  - wachttijd voor sensoren?
- 1 onderdeel kapot: hele keten functioneert niet meer

Met een message broker:

- Verschillende topics/queues voor verkeersdrukte, luchtkwaliteit en het weer
- Verschillende consumers (receivers) voor het verwerken van de topics/queues
- Geen dataverlies bij uitval van 1 of meerdere services
- Loskoppeling sensoren en verwerking

Het MQTT protocol

Een "Eenvoudig" publish/subscribe (pub/sub) protocol.

- publish naar *topic*
- subscribe *topic*

Eigenschappen topic:

- case sensitive
- / als "level" scheider: hierarchische structuur
- hoeft niet met / te beginnen

Voorbeelden van topics:

- huis/keuken/temperatuur
- huis/badkamer/temperatuur
- garage/licht

Bekende broker voor MQTT: mosquitto

Wildcards

Om makkelijk te *subscriben* voor meerdere topics kun je 2 soorten wildcards gebruiken: de **+** en het **#**-je.

+ : matcht 1 level

Bijvoorbeeld:

/huis/**+**/temperatuur

- /huis/keuken/temperatuur
- /huis/badkamer/temperatuur
- /huis/…/temperatuur

: matcht 1 of meerdere levels

/huis/**#**

- /huis/keuken/temperatuur
- /huis/hal/camera
- ...

MQTT: Quality of Service

Default: **QoS 0** ("At most once", "Fire and forget")

- Bericht wordt niet bewaard.
- Als een subscriber niet online is verliest deze het bericht
- Geschikt voor niet-kritische updates (bijv. temperatuurmetingen elke x seconde).

QoS 1 ("At least once")

- Broker ontvangt bericht, gaat naar subscribers en een PUBACK wordt verstuurd naar de publisher
- Als de bevestiging (PUBACK) niet wordt ontvangen, wordt het bericht opnieuw verstuurd.
- Kan leiden tot dubbele ontvangst van berichten.

QoS 2 ("Exactly once")

- Broker en abonnee voeren een vierstaps bevestigingsproces uit om te garanderen dat het bericht exact één keer wordt ontvangen.
- Bericht wordt pas verwijderd nadat het correct is afgeleverd.

Worden MQTT berichten bewaard op de broker?

Default niet.

Maar "persistent" te maken bij QoS = 1 of QoS = 2:

Create subscription client met:

- client ID
- en met `clean_session=False` (default True)

Bewaartermijn beinvloedbaar met TTL (client side, MQTT), "expiry setting" (broker)

Authenticatie?

Op de server (in dit geval voor Mosquitto)

In /etc/mosquitto/mosquitto.conf:

```
allow_anonymous false

password_file /etc/mosquitto/passwords
```

Generate password:

```
mosquitto_passwd -c /etc/mosquitto/passwords username
```

TLS encryptie

Maak CA (Certification Authority) certificaten in /etc/mosquitto/ca_certificates:

- `openssl genrsa -out ca.key 2048`
- `openssl req -new -x509 -days 3650 -key ca.key -out ca.crt -subj "/CN=MQTT CA"`

Kopieer naar /etc/mosquitto/ca_certificates

Maak broker certificaten in /etc/mosquitto/certs

- `openssl genrsa -out server.key 2048`
- `openssl req -new -key server.key -out server.csr -subj "/CN=mosquitto"`
- `openssl x509 -req -days 3650 -in server.csr -CA "path"/ca.crt -CAkey \ "path"/ca.key -CAcreateserial -out server.crt`

TLS encryptie

Config server (/etc/mosquitto/mosquitto.conf):

```
listener 8883

cafile /etc/mosquitto/ca_certificates/ca.crt

certfile /etc/mosquitto/certs/server.crt

keyfile /etc/mosquitto/certs/server.key
```

Config clients (pub/sub):

```
client.tls_set("ca.crt")
```

RabbitMQ

- RabbitMQ is, net als Mosquitto, een ***message broker***
- Kan berichten ontvangen, *queue*-en en doorsturen
- Werkt met een exchanges en queues:
  - een ***exchange*** is nodig om de juiste queues aan te wijzen
  - een queue bestaat pas als deze gebonden is aan een exchange (binding)
- zender: ***producer***
- ontvanger: ***consumer*** (ook wel *receiver*)
- Ook A-synchrone communicatie
- Default protocol: Advanced Message Queuing Protocol (AMQP). Maar ook bv. Streaming Text Oriented Messaging Protocol (STOMP), MQTT, …). Dmv plug-ins.
- Veel programmeertaal ondersteuning voor clients
- Past ook goed in "serverloze" architectuur (cloud omgevingen)

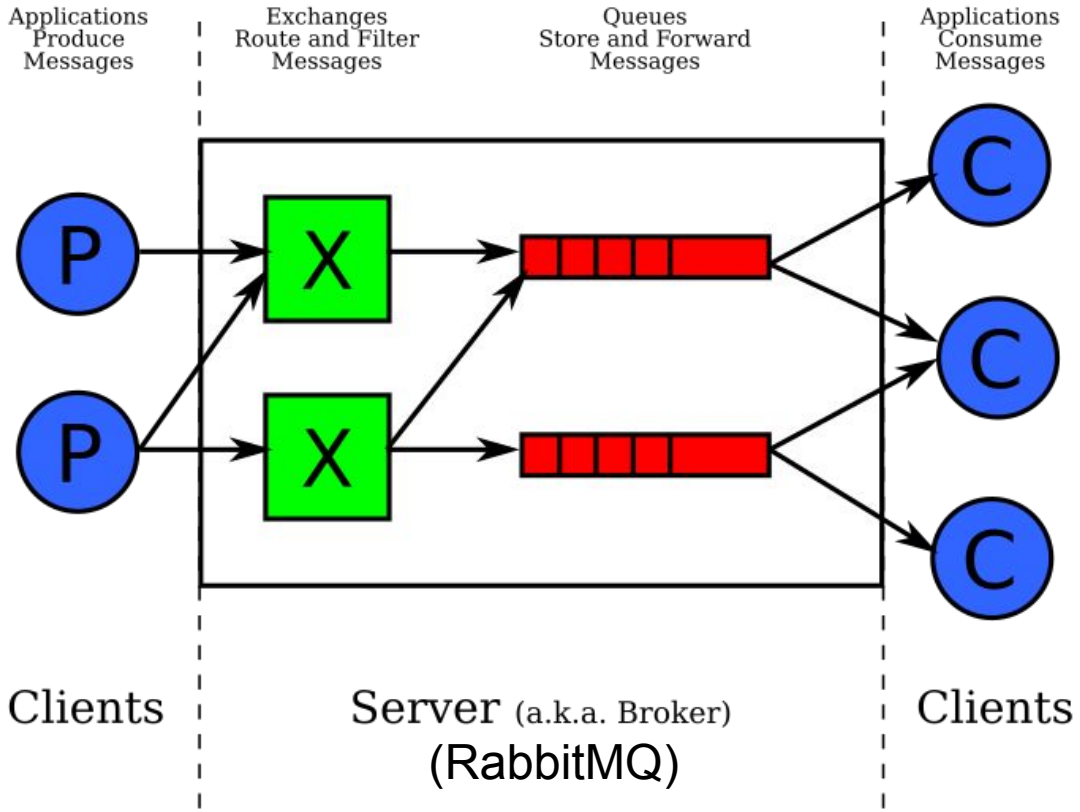
"Install"

```
docker run -it --rm --name rabbitmq -p 5672:5672 \
        -p 15672:15672 rabbitmq:4.0-management
```

port 5672: RabbitMQ (AMQP protocol)

port 15672: RabbitMQ admin interface

In een plaatje



Exchanges

Berichten niet rechtstreeks naar de queue: via een "exchange"

Exchange: zorgt dat bericht in de goede queue(s) terecht komt (*routing*)

Routering (*routing rule*) van messages door de exchange hangt af van:

- exchange type (straks meer):
  - direct
  - topic
  - fanout
  - headers
- routing key: het "adres" voor het bericht: naar welke queue(s) moet het bericht gestuurd worden
- header attributen

De link tussen een exchange en de queue: *binding*

exchange types

Direct exchange type:

- Default.
- Bericht wordt naar een **specifieke** queue gestuurd middels de routing key
- Geen publish/subscribe maar point to point: 1 zender naar 1 ontvanger.
  - 1 consument per bericht
- Impliciet: load balancing!

Topic:

- Lijkt op topic van MQTT
- Berichten wordt naar **1 of meerdere** queues gestuurd middels de routing key
- Routing key bestaat uit woorden gescheiden door een . (punt)
- Bindings kan met wildcards om 1 bericht naar meerdere queues te sturen.
- Voorbeeld: log notificatie systeem

exchange types

Fanout exchange type

- Elk bericht gaat naar alle queues, lege routing key
- Consumer bind z'n eigen queue
- Voorbeeld: "live updates", bv. voor een sport event

Header exchange type

- routering niet met routing key maar op basis van client header parameters, bv.:
  - "headers": {"severity": "warning", "department": "support"}
  - "headers": {"severity": "warning", "department": "admin"}
- Geen gebruik van een routing key
- Consumer bind "zijn queue" aan de hand van filtering op de header parameters
  - queue_bind(arguments={"department": "support", "severity": "warning", "x-match": "any"})

Blijft alles bewaard?

Berichten worden standaard bewaard maar hoe zit het met een herstart van de broker?

Drie soorten:

- Exchange Durability (durable=True)
- Queue Durability (durable=True)
- Message Persistence (delivery_mode=2) -> performance..

Overeenkomsten en verschillen

MQTT	AMQP
Puur publish/subscribe (minder overhead)	Point-to-point, publish/subscribe
shared subscriptions (MQTT 5.0), creatief naar verschillende topics publiceren..	Native support load balancing
Wildcard topics	Uitgebreide routering opties
Default worden berichten niet bewaard (maar QoS)	Data wordt standaard bewaard (maar "durable" nodig voor herstarts)
IoT, sensornetwerken, embedded systemen, weinig bandbreedte	Complexe routeringen, bedrijfskritische apcplicaties, financiële systemen, genoeg resources
Header 2 bytes. Max payload 256MB	Header 8 bytes. Max payload 2GB
Compact, lichtgewicht, weinig overhead	Complexer, meer overhead
TLS te veel overhead/cpu-usage voor lichtgewicht clients?	