Data brokers & Cloud messaging services

Pelle Jakovits

April 2022, Tartu
Outline

• Messaging services
  – MQTT
  – AMQP
  – RabbitMQ

• Cloud messaging services
  – AWS MQ
  – AWS SQS
  – AWS SNS
Messaging services

• Services that facilitate the delivery of messages between distributed components and services
• Often used as distributed message queues
• Publish-Subscribe asynchronous pattern of communication
  – N publishers, M subscribers, X message queues
• Examples:
  – MQTT
  – AMQP and RabbitMQ
  – Kafka
  – AWS Simple Queue Service (SQS), AWS Simple Notification Service (SNS)
Using message queues in distributed systems

• There are advantages in using message queues in Cloud, in microservice architecture and in distributed systems
• Message queues allow senders and receivers not have to be active at the exact same time
• Queue acts as a mediator between different processes, that stores the messages for a while
• There can be many different queues
• Each sender and receiver can have its own local queue
• Enables asynchronous communication
• Enables ease of parallelization
Components of queue-based systems

- System components when designing queue based distributed systems
  - Input and Output queues
  - Queue labels, locations of the queues in a database or storage
  - Queue managers
  - Relays (intermediaries)
  - The queue network becomes a new overlay network on top of the actual network
  - Routing issues
  - Routing tables
    - Routing rule management is often manual
Message formats

• There is a fundamental problem with message formats - messages can contain anything

• It is usually impossible to agree on a format in advance - we agree to allow messages in different formats
  – JSON, XML
  – Images, Videos
  – Plain text, encoded binary values

• Dedicated nodes can translate the messages that pass through them
  – In terms of system architecture, these are common applications that use queues
• Publish-subscribe model network protocol that transmits messages between devices
• It is usually set up as a separate server
• Exchanges data between components of distributed systems
• Suitable for asynchronous communication and system scaling
  – N message producers
  – M message consumers
  – Messages are distributed between M consumers - no complicated synchronization required
Simplified MQTT publish subscribe process
• In practice, there are no queues!
  – Instead, use Topics to define how messages are routed

https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/
MQTT - Quality of Service

• **At most once** - Level 0
  – The message is sent at most once or not at all
  – Duplicates not allowed

• **At least once** - Level 1
  – The message is always sent at least once
  – Duplicates allowed

• **Exactly once** - Level 2
  – The message is always sent exactly once
  – No duplicates allowed
MQTT example

https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/
AMQP

• Established in 2006, V1.0 in 2011.
• Advanced Message Queuing Protocol (AMQP)
• An open standard application layer protocol for message-oriented middleware
• Asynchronous, queue-based
• Messages remain in the queue until they are retrieved
• Authentication and encryption based on SASL or TLS
• Implementations:
  – RabbitMQ, Apache ActiveMQ, Azure Service Bus
  – Amazon MQ (ActiveMQ, RabbitMQ managed service)
AMQP entities

- **Exchange** – A message arrival "mailbox"
  - Can be temporary or permanent
  - Messages can't be sent directly to queues.
- **Queue** - ordered collections of messages.
  - Subscribers connect to queues to listen to messages
- **Routing Key** - A key that describes the type of message.
  - Used to filter and route messages
  - For example: "myfloor.livingroom.temperature"
- **Binding** – Defines which messages to route from exchange X to queue Y based on a specified routing key pattern.
  - Route message from exchange A to queue Q, if routing key: "myfloor.livingroom.*"
- **Message** - The message to be sent
  - **Header**: A key-value style metadata, such as a routing key
  - **Body**: A binary object that is the content of the message
AMQP

- Publish-Subscribe pattern of message brokerage

https://support.smartbear.com/readyapi/docs/testing/amqp.html
AMQP

https://pythonhosted.org/nucleon.amqp/amqp.html
Routing messages

• Messages contain a **routing key** that is used to route the message
  • tartu.delta.outdoorlightlevel
  • tartu.delta.floor3.room3040.temp

• **Bindings** can use wildcards * and #.
  – * - substitutes one word
  – # - substitutes multiple words
  – tartu.delta.korrus3.*.Temp

https://www.rabbitmq.com/tutorials/tutorial-five-python.html
AMQP ja MQTT differences

• Exchanges - Data is not sent directly to the queue, but to the Exchange
• Routing must be explicitly defined between Exchanges and queues
• Queues are not the same as MQTT topics
  – AMQP queues are like mailboxes where messages with a specific routing keys are delivered
  – AMQP routing key ~= MQTT topic
    • But binding MUST exist for a message to be delivered to a AMQP queue.
RabbitMQ

• Direct implementation of the AMQP protocol
• Open Source (https://github.com/rabbitmq)
• Many plugins available (federation, mqtt, scheduled delivery, email delivery, priority queues)
• Advanced features:
  – Clustering
  – Federation
  – Web management interface
  – User level access control
RabbitMQ exchange types

- **Direct exchange** - The exchange directs messages to queues whose routing key is identical to the message key.

- **Fanout exchange** - Messages are sent to all related queues. The message forwarding key is ignored.

- **Topic exchange** - Redirects messages to related queues, if the message routing key matches the pattern specified in the pairing routing key.

- **Header exchange** – Routing messages based on matching values inside message headers
Direct Exchange

http://www.imperative-design.net/insights/rabbitmq-amqp-messaging-standard/
Fanout Exchange

http://www.imperative设计.net/insights/rabbitmq-amqp-messaging-standard/
Topic Exchange

Producers can send messages to a topic exchange using a routing key. The exchange then routes the message to one or more queues based on the routing key pattern. Consumers can then listen to these queues to receive messages.

Routing key: "first.green.fast"

Queue patterns:
- "*.green.*"
- "*.red.fast"
- "*.fast"

Queues are connected to consumers to handle the messages.

http://www.imperative-design.net/insights/rabbitmq-amqp-messaging-standard/
Header exchange

http://www.imperative-design.net/insights/rabbitmq-amqp-messaging-standard/
RabbitMQ Management interface

- Manage users and their permissions
- Set up vHosts – Virtual hosts
- Set up clusters
- Display statistics
- Define policies: messages expiry, message max length, queue size, delivery attempts, etc.
RabbitMQ Federation

• Enables to set up message routing between different brokers
• Federation is NOT clustering
• **Federated exchanges**: Published messages are replicated across multiple nodes or clusters
• **Federated queues**: distributed queues, consumers can consume from any location
Federated Exchange

- One directional or multi-directional replication of messages between exchanges.
- Each exchange has its own local queues.

https://www.rabbitmq.com/federated-exchanges.html
Federated Queues

- Sharing or migrating messages between queues located in different brokers
- "Real" distributed queues –
  - Distributes messages across consumers connected to any of the federated brokers
  - If there is no local consumer, consumer on another broker can receive the message instead
- A federated queue will retrieve messages from other federated queues, if the local queue is empty
- If there are local consumers, they have priority on the next message if queue is empty.
Cloud messaging services

- Managed and scalable Data brokers
- Fully managed messaging platforms
- As an example, let's take a look at multiple different types of Cloud messaging services in AWS
  - Amazon MQ
  - Amazon SQS
  - Amazon SNS
Amazon MQ

- Managed message broker service
- Compatible with typical message brokers
  - AMQP, MQTT, OpenWire, and STOMP
- Managed Apache ActiveMQ or RabbitMQ
  - Setting up a message broker, software upgrades, security updates, fault detection and recovery.
  - Integrated monitoring and logging with AWS CloudWatch, CloudTrail
- Billed for broker instance runtime and storage usage
Amazon Simple Queue Service

• Managed message queues for microservices, distributed systems, and serverless applications
• 1 million requests free
• Messages are persisted if no consumer available
  – Messages can be stored in SQS up to 14 days (4 is default)
• Consumers poll messages from SQS
• Message delivery is guaranteed
• Amazon SQS replicates messages across multiple servers for durability
• SQS is an internally scaled system – user does not need to know or manage any clustering.
SQS Standard vs FIFO queues

**Standard Queue**
- Advertises nearly unlimited number of API calls per second
- **At-Least-Once Delivery** – A message is delivered at least once. But may produce duplicates
- **Best-Effort Ordering** – Messages can be sometimes be delivered out of order.

**FIFO queue**
- Up to 3,000 messages per second
- **Exactly-Once Processing** – A message is delivered exactly once and persisted until consumer deletes it. Duplicates are not allowed.
- **First-In-First-Out Delivery** – Message receive order is same as delivery order.
Amazon SQS pricing

- Pay per million requests
- Every Amazon SQS action counts as a request
- Each 64 KB chunk of a payload is billed as 1 request
- API actions for sending, receiving, deleting, and changing visibility of messages from FIFO queues are charged at FIFO rates.
- Also pay for data transfer out of AWS. $0.05 to $0.09 per GB

<table>
<thead>
<tr>
<th>Standard Queues (per Million requests)</th>
<th>FIFO Queues (per Million requests)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First 1 Million Requests/Month</strong></td>
<td>Free</td>
</tr>
<tr>
<td><strong>From 1 Million to 100 Billion Requests/Month</strong></td>
<td><strong>$0.40</strong></td>
</tr>
<tr>
<td><strong>From 100 Billion to 200 Billion Requests/Month</strong></td>
<td><strong>$0.30</strong></td>
</tr>
<tr>
<td><strong>Over 200 Billion Requests/Month</strong></td>
<td><strong>$0.24</strong></td>
</tr>
</tbody>
</table>
Amazon SNS

• Simple Notifications Service
• Managed service for pub/sub messaging, SMS, email, and mobile push notifications
• 1 million requests free
• Application-to-application messaging
• Application-to-person notifications
• SNS pushes messages to consumers (no polling)
• Message delivery is NOT guaranteed
• pay-as-you-go model with no up-front costs
Amazon SNS

Publisher
Publish messages from distributed systems, microservices, and other AWS services

Amazon SNS
Fully-managed pub/sub messaging and event-driven computing service

SNS Topic
Decouple message publishers from subscribers with topics

Message Filtering & Fanout
Filter messages according to subscription filter policies, and deliver them to subscribers

Dead-letter Queue
If an endpoint is unavailable, messages can be held in a dead-letter queue for analysis or reprocessing

Subscribers
Receive messages in subscribing serverless functions, queues, microservices, delivery streams and more

AWS Lambda
Amazon SQS
Amazon Kinesis
Data Firehose
HTTP/S
Email

Pelle Jakovits
SNS pricing

- Every publish, publish batch, topic owner operation, and subscription owner operation counts as a request.
- For Standard topics, each 64 KB chunk of a payload is billed as 1 request. FIFO topics, it is 256 KB.
- Publish and publish batch API requests are $0.36 per 1 million and $0.0204 per GB of payload data.
- Subscription messages are $0.012 per 1 million and $0.0012 per GB of payload data.
- Also may have to pay for data transfer out of AWS. $0.05 to $0.09 per GB.
# SNS vs SQS

<table>
<thead>
<tr>
<th></th>
<th>SNS</th>
<th>SQS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Persistence</strong></td>
<td>No persistence</td>
<td>Up to 14 days</td>
</tr>
<tr>
<td><strong>Access pattern</strong></td>
<td>Pushing messages</td>
<td>Pulling messages</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Pub/Sub system</td>
<td>Queue</td>
</tr>
<tr>
<td><strong>QoS</strong></td>
<td>No guarantees, if target is unavailable.</td>
<td>Can guarantee to be delivered at least once</td>
</tr>
<tr>
<td><strong>Use cases</strong></td>
<td>• Publish messages to many consumers</td>
<td>• Work queues – workers take work from a queue. Messages divided between them</td>
</tr>
<tr>
<td></td>
<td>• Publishing to external services: email, SMS</td>
<td>• Queued</td>
</tr>
<tr>
<td></td>
<td>• Trigger Lambda functions</td>
<td></td>
</tr>
</tbody>
</table>
SQS + SNS

https://medium.com/awesome-cloud/aws-difference-between-sqs-and-sns-61a397bf76c5

Pelle Jakovits
Conclusions

• Data brokers and distributed message queues can simplify scaling distributed systems.
  – "Simply" put a message queues between the components of distributed systems.
• Enables asynchronous communication patterns
• Durable queues can help to avoid missing messages
• QoS features help to guarantee message delivery
That’s all for today

• This week’s practice session
  – RabbitMQ

• Next week’s lecture
  – Internet of Things
THANK YOU FOR YOUR ATTENTION

jakovits@ut.ee