ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

Message Queuing

Narges Mehran, MSc.,
Dr. Dragi Kimovski

Current Topics in Distributed Systems: Internet of Things and Cloud Computing,

SS21

14.04.2021

What is MQ

- Message queue (MQ) is a temporary message storage when the destination application is busy or not connected.
- Message queuing allows applications to communicate by sending messages to each other.
- Message queue provides **asynchronous communications protocol.**
- The sender and receiver of the message do not need to interact with the message queue at the same time.
  - Email is probably the best example of asynchronous communication.

Message queuing - a simple use case

- Imagine that you have a web service,
  - receives many requests every second
  - no request can get lost
  - should not be locked by processing previously-received requests
  - all requests need to be processed by a function that has a high throughput
- Solution:
  - placing a queue between the web service and the processing service is ideal
- The queue will persist with the requests even if their number grows.

Basic architecture of message queue

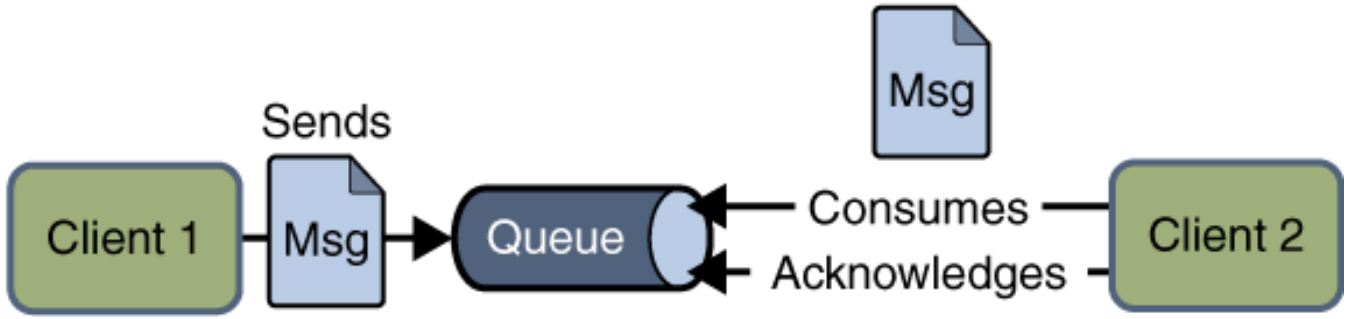
- The basic architecture of a **message queue** is simple:
  - There is a client application, called producer, that creates messages and sends them to the message queue.
  - Another application, called a consumer, connects to the queue and retrieves the messages to be processed.
  - Messages in the queue are stored until the consumer retrieves them.

Basic architecture of message queue

https://www.cloudamqp.com/blog/2014-12-03-what-is-message-queuing.html

Point-to-point messaging

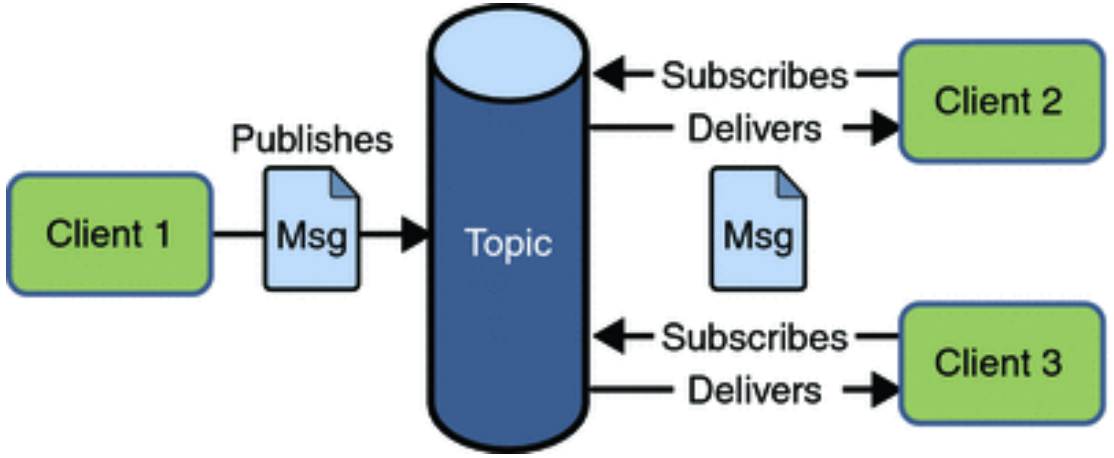
- There is only one client for each message.
- There is no timing dependency for sender and receiver of a message.
- The receiver sends the acknowledgement after receiving the message.



Publish/Subscribe messaging

- In publish/subscribe messaging approach, one message is published, which is then delivered to all clients.

- Some of the characteristics are:
  - There can be multiple subscribers for a message.
  - The publisher and subscriber have a timing dependency.
  - A client that subscribes to a topic can consume only messages published after the client has created a subscription, and the subscriber must continue to be active for it to consume messages.



Some popular high-throughput messaging systems

Distributed messaging system	Description
Apache Kafka	Kafka was developed at LinkedIn corporation and later it became a sub-project of Apache. Apache Kafka is based on broker-enabled, persistent, distributed publish-subscribe model. Kafka is fast, scalable, and highly efficient.
RabbitMQ	RabbitMQ is an open source distributed robust messaging application. It is easy to use and runs on all platforms.
JMS (Java Message Service)	JMS is an open-source API that supports creating, reading, and sending messages from one application to another. It provides guaranteed message delivery and follows publish-subscribe model.
ActiveMQ	ActiveMQ messaging system is an open-source API of JMS.
ZeroMQ	ZeroMQ is broker-less peer-peer message processing. It provides push-pull, router-dealer message patterns.
Kestrel	Kestrel is a fast, reliable, and simple distributed message queue.

https://www.tutorialspoint.com/

ZeroMQ

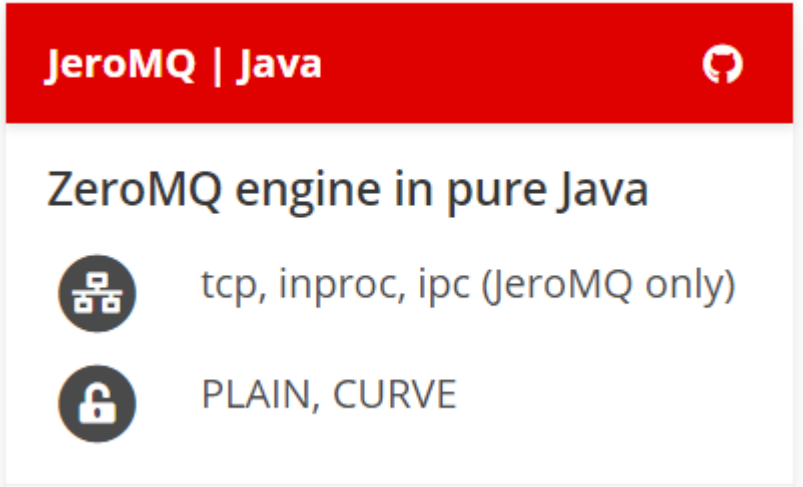
- ZeroMQ (also spelled ØMQ, 0MQ or ZMQ) is a high-performance asynchronous messaging library,
  - aimed at use in distributed or concurrent applications.

- It provides a message queue, but unlike message-oriented middleware, a ZeroMQ system can run *without a dedicated message broker*.

ZeroMQ

- ZeroMQ supports
  - common messaging patterns (pub/sub, request/reply, client/server and others)
  - over a variety of transports (TCP, in-process, inter-process communication, multicast, WebSocket and more),
  - making inter-process messaging as simple as inter-thread messaging.
  - This keeps your code clear, modular and extremely easy to scale.

- ZeroMQ is developed by a large community of contributors.
- There are third-party bindings for many popular programming languages and native ports for C# and Java.



Socket API

- Sockets are the de facto standard API for network programming.

- Conventional sockets present a synchronous interface to either connection-oriented reliable byte streams (SOCK_STREAM), or connection-less unreliable datagrams (SOCK_DGRAM).
  - ✓ZeroMQ sockets present an abstraction of an asynchronous message queue, with the exact queueing semantics depending on the socket type in use.

- Conventional sockets transfer streams of bytes or discrete datagrams,
  - ✓ZeroMQ sockets transfer discrete messages.

Socket Type

- REQ
- REP

- SUB
- PUB

- PULL
- PUSH

- DEALER
- ROUTER

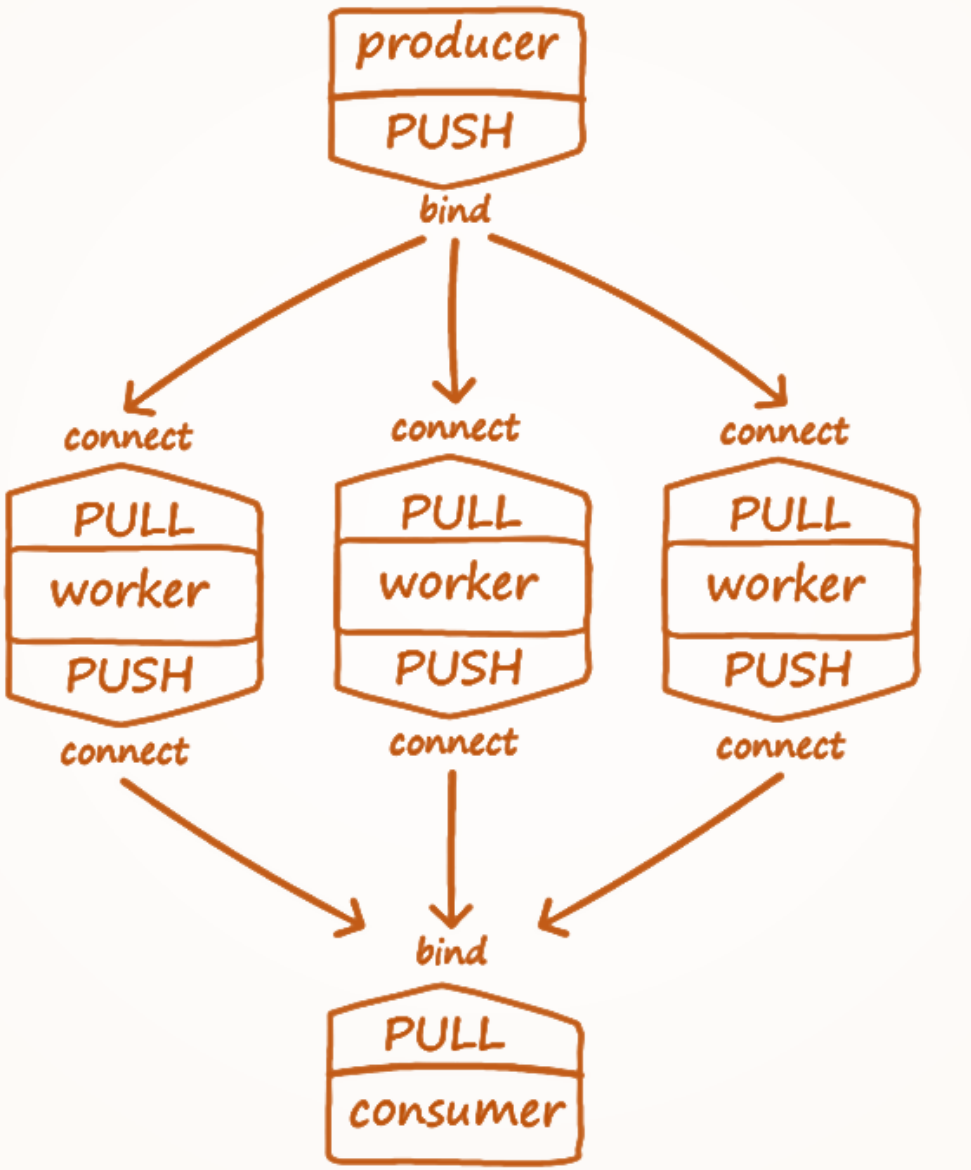
ZeroMQ API

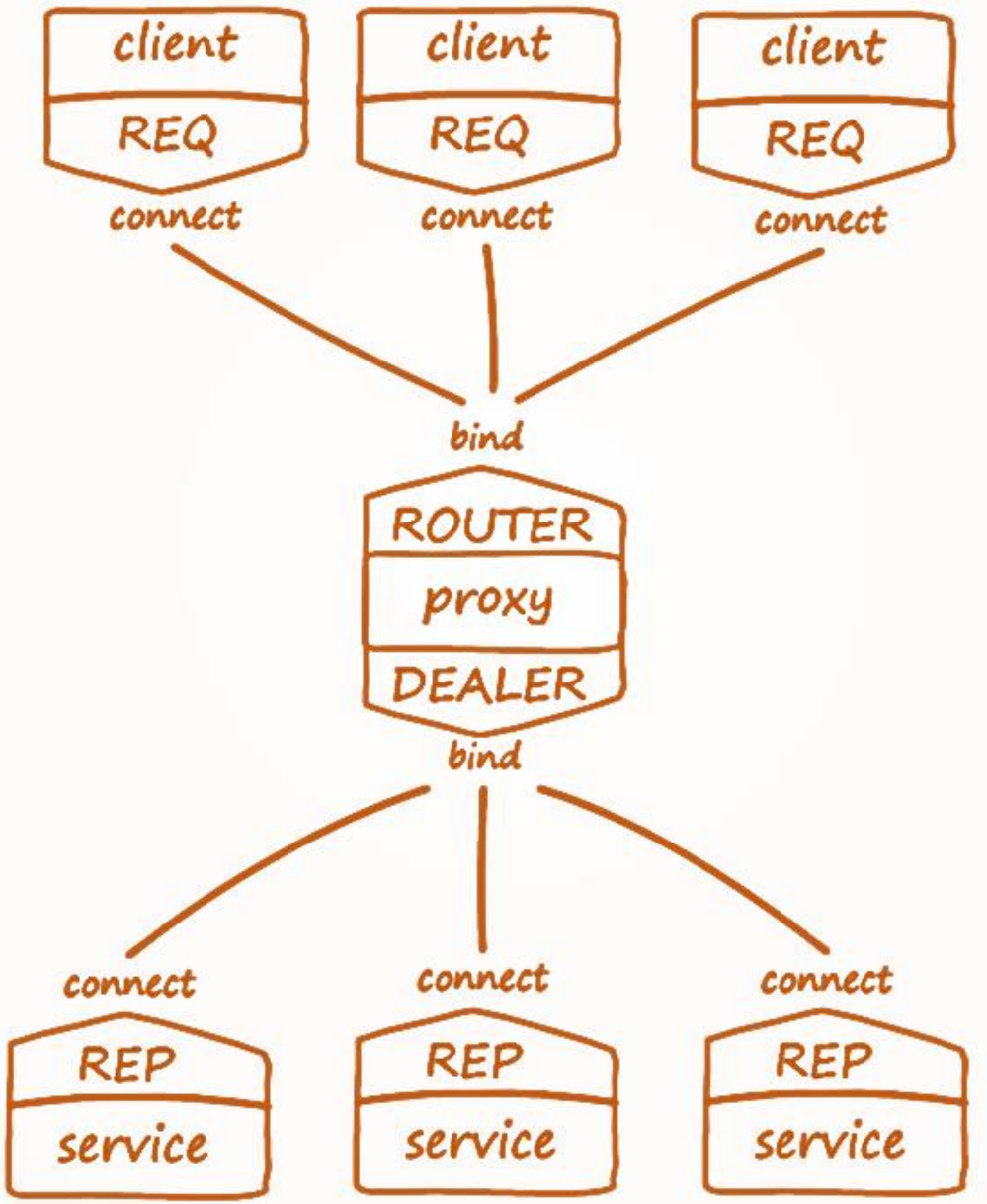
- Objects: context, socket, message
- Steps: create context > bind/connect > send/receive

```
# server program
context = zmq.Context(1) # A zmq Context creates sockets via its ctx.socket method.
socket = context.socket(zmq.PUB)
socket.bind("tcp://*:5557")
socket.send("some event")
```

```
# client program
context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.connect("tcp://server:5557")
event_data = socket.recv()
```

Communication models



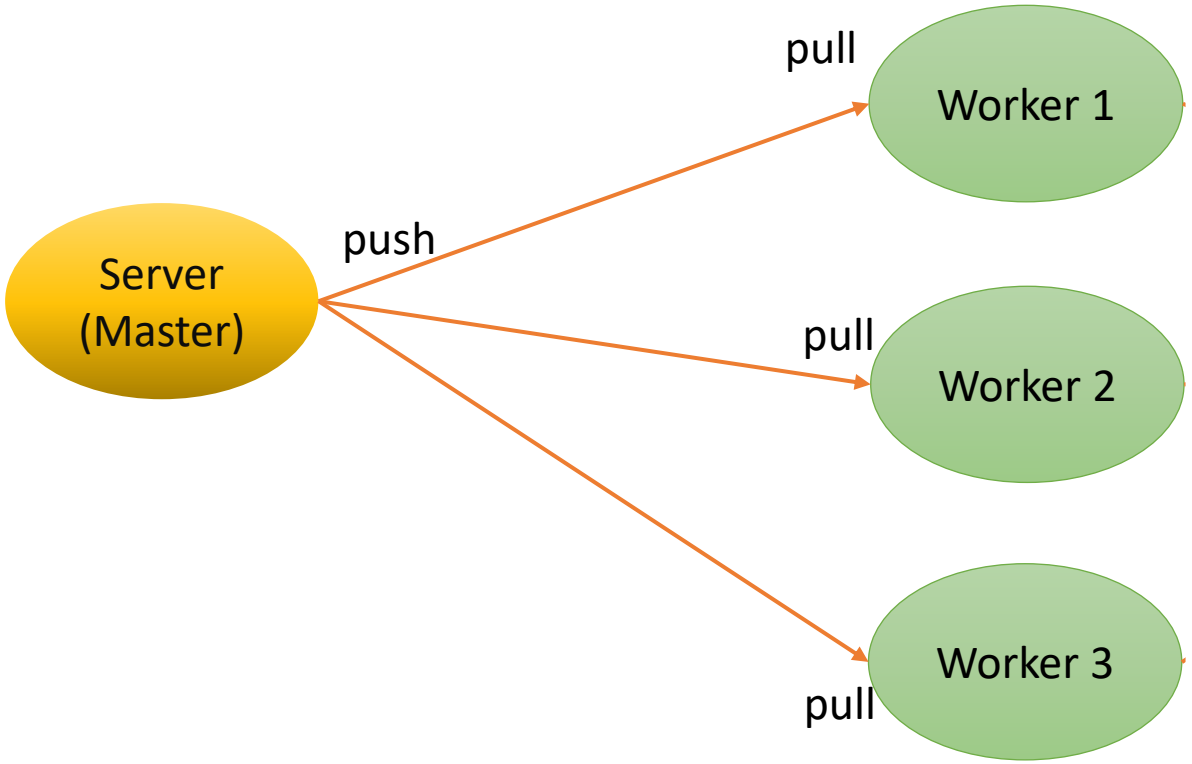


Getting started with JeroMQ

- Firstly, please install and configure Maven:
  - https://maven.apache.org/install.html#windows-tips
- Secondly please download Jeromq:
  - https://github.com/zeromq/jeromq/archive/refs/heads/master.zip
- Please open the client and server programs of *hwserver.java* and *hwclient.java* from the path:
  - jeromq\src\test

Task

- Please setup a simple queue (push-pull) with ZeroMQ,
- Please start coding in Java.



Task (cont.)

- The server should send a message (any message!) to each connected worker for example, every half a second.
- The connection to the workers may look like the following:

Worker 1	Worker 2	Worker 3
Connected to server. received job sending job 0 received job sending job 3 received job sending job 6 received job sending job 9 received job sending job 12 received job sending job 15 received job sending job 18	Connected to server. received job sending job 1 received job sending job 4 received job sending job 7 received job sending job 10 received job sending job 13 received job sending job 16 received job sending job 19	Connected to server. received job sending job 2 received job sending job 5 received job sending job 8 received job sending job 11 received job sending job 14 received job sending job 17 received job sending job 20

References

- https://zeromq.org/
- https://zguide.zeromq.org/docs/