

Computação Distribuída – Guião 3

LEI / Universidade de Aveiro

Diogo Gomes & Nuno Lau & Mário Antunes & Alfredo Matos

Abril 2023

Introdução

Um *Message Broker* é um middleware tipicamente utilizado em sistemas distribuídos que necessitam de fazer EAI (*Enterprise Architecture Integration*). Muitas empresas (*Enterprises*) são fornecidas por fabricantes de software distintos que precisam de ser integrados para completar os processos de negócio da empresa. A integração é mais difícil quando os softwares comunicam através de protocolos distintos e usam serialização de dados distintas.

Os *Message Broker* são usados nestas situações para traduzir a comunicação entre estes softwares.

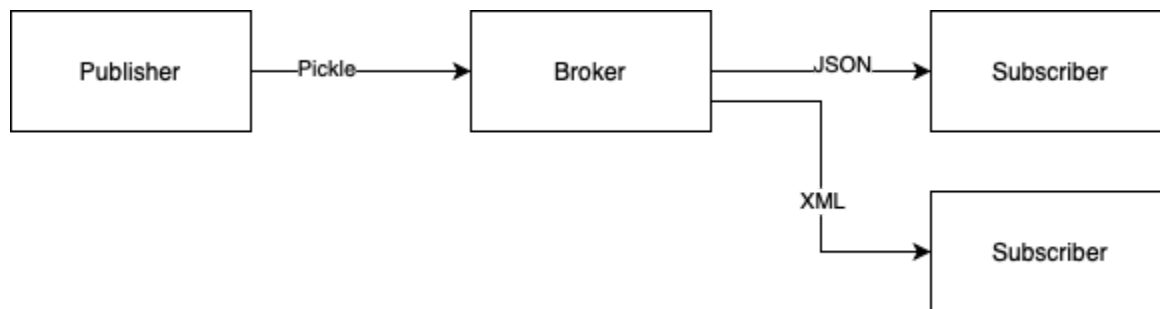


Figura 1

No nosso projeto temos um conjunto de produtores (*Producers*) de dados que precisam de ser consumidos por consumidores (*Consumers*) que suportam mecanismos de serialização distintos. A comunicação é suportada por um modelo de *publish-subscribe* em que os produtores são *Publishers* e os consumidores são *Subscribers*. A tarefa deste projeto consiste no desenvolvimento de um *Message Broker* rudimentar que resolva esta situação.

Objetivos

O objetivo deste trabalho é o desenvolvimento de um *Message Broker* capaz de interligar produtores (*Producers*) e consumidores (*Consumers*) através de um protocolo **PubSub** comum e de três mecanismos de serialização distintos (XML, JSON e Pickle).

Para além do *Message Broker* é necessário o desenvolvimento de um middleware que torne o processo de comunicação transparente para os produtores e consumidores.

Requisitos da solução

O protocolo PubSub deve ser definido por cada grupo e documentado no ficheiro `PROTOCOLO.txt`. Este protocolo deve ser implementado sobre o protocolo **TCP** com o **Message Broker** a fazer uso do porto **5000** para estabelecer ligação com consumidores/produtores. Este protocolo deve incluir:

- Mensagem de **Subscrição** de um tópico
- Mensagem de **Publicação** num tópico
- Mensagem de **Pedido de Listagem** de tópicos
- Mensagem de **Cancelamento de Subscrição** de um tópico

Tanto os **produtores como os consumidores** deverão ser capazes de **sinalizar** ao *Message Broker* o mecanismo de **serialização que suportam**, sendo da competência do *Message Broker* garantir a troca de mensagens de todos para todos independentemente do mecanismo de serialização suportado.

Uma característica dos *Message Brokers* é a sua capacidade de **reter mensagens** (persistir) que podem ser entregues aos consumidores assim que estes se ligam. Um consumidor que se ligue ao *Message Broker* e subscreva um tópico onde previamente foram publicadas mensagens, **deve receber a última mensagem publicada**.

O *Message Broker* deve resolver sem problemas situações em que produtores ou consumidores desligam a sua conexão.

Junto com este enunciado são fornecidas duas aplicações python (`producer.py` e `consumer.py` cuja implementação está em `clients.py`) que fazem uso de uma classe que implementa um esqueleto de middleware (`middleware.py`) e de broker (`broker.py`). **Não deve fazer qualquer alteração em `clients.py`, `producer.py` nem em `consumer.py`**, mas deve alterar o código de `middleware.py` e `broker.py` de acordo com as suas necessidades.

A resolução correta do exercício pressupõe (por nível de dificuldade crescente):

- 1 *Producer* – *Message Broker* – 1 *Consumer* (1 único mecanismo de serialização)
- 1 *Producer* – *Message Broker* – 1 *Consumer* (2 mecanismos de serialização)
- 1 *Producer* – *Message Broker* – N *Consumers* (3 mecanismos de serialização)
- N *Producers* – *Message Broker* – N *Consumers* (3 mecanismos de serialização)

O *Message Broker* deve ainda implementar tópicos hierárquicos (em árvore). A subscrição de um tópico implica a subscrição implícita de todos os subtópicos desse tópico. Exemplo: a subscrição do tópico *weather* implica a subscrição de *temperature*, *humidity* e *pressure*, no caso de *weather* ter como subtópicos *temperature*, *humidity* e *pressure*. A estrutura hierárquica dos tópicos deve usar a seguinte nomenclatura: `/weather`, `/weather/temperature`, `/weather/humidity`, `/weather/pressure`, etc.. E deve suportar futuros tópicos não previstos e com profundidade crescente.

Na realização deste projeto apenas podem utilizar bibliotecas nativas do Python.

Prazo

8 de Maio de 2023

Entrega através do Github Classroom (automática)

GitHub Classroom

- Este projeto é realizado em **grupos de 2** alunos.
- Para resolver este guião deverá começar por aceitar o mesmo em <https://classroom.github.com/a/-kCKQsWz>
- Ao aceitar o guião será criado um repositório online a partir do qual deve fazer um clone local (no seu computador).
- Deverá enviar as suas alterações periodicamente para o repositório e manter-se atento ao canal #cd em <https://detiuaveiro.slack.com>
- Antes do prazo, o seu código deverá passar os testes automáticos (*tab* “Actions” no seu repositório github)

Protocolo

O seguinte diagrama demonstra o protocolo a implementar. No entanto, não é especificado o formato das mensagens, tal fica ao cuidado de cada grupo.

Message Broker

