IS Ficha 1

João Laranjeiro*, Marcelo Gomes*
*University of Coimbra, DEI
Emails: {joaolaranjeiro, marcelogomes}@student.dei.uc.pt

***Abstract*—Efficient data representation is crucial for the performance of microservices, particularly in systems handling large volumes of data. This paper examines the performance of two widely used serialization formats, JSON and XML, with the hypothesis that JSON is faster than XML in both serialization and deserialization, especially with larger datasets. Selecting the faster format is challenging, as performance overhead can significantly affect system throughput in large-scale systems. To test this hypothesis, we conducted a series of controlled experiments comparing the speed of JSON and XML across various data sizes. The findings confirm that JSON consistently outperforms XML in terms of speed, particularly at higher data volumes. These insights provide valuable guidance for developers and architects in selecting data formats for microservices architectures where performance is a key concern.**

I. Introduction

A representação de dados é uma componente fundamental no desenvolvimento de aplicações modernas, especialmente em arquiteturas baseadas em microsserviços, onde diferentes serviços interagem entre si através da serialização e desserialização de dados. No contexto desta tarefa, serão explorados dois formatos de representação de dados amplamente utilizados: JSON (*JavaScript Object Notation*) e XML (*Extensible Markup Language*). Ambos os formatos têm as suas particularidades, vantagens e desvantagens, tornando-os adequados para diferentes cenários de aplicação.

Neste relatório, formulamos a hipótese de que "**JSON é mais rápido que XML em termos de serialização e desserialização, especialmente com grandes volumes de dados.**". Esta suposição baseia-se em características intrínsecas dos formatos, onde o JSON é geralmente considerado mais leve e menos denso que o XML, o que pode impactar a sua rapidez em ambientes onde múltiplos microsserviços precisam de trocar informações de forma rápida. Para validar esta hipótese, utilizaremos a biblioteca Jackson, que é uma das ferramentas mais populares para a manipulação de dados em formato JSON e XML em Java. Através de uma série de testes rigorosos de desempenho, mediremos o tempo necessário para serializar e desserializar dados utilizando ambos os formatos.

Este estudo visa contribuir para a compreensão das implicações na escolha de formatos de representação de dados na velocidade de sistemas distribuídos. Destacando as considerações que os desenvolvedores devem ter em mente ao projetar soluções baseadas em microsserviços. A análise dos resultados obtidos irá permitir não só confirmar ou refutar a hipótese proposta, mas também proporcionar uma visão mais aprofundada sobre os fatores que influenciam a rapidez dos formatos em cenários reais de utilização.

II. Concepts

Neste capítulo, serão abordados os conceitos fundamentais relacionados aos formatos de representação de dados JSON e XML, bem como os princípios da serialização e desserialização, que são cruciais para a compreensão da hipótese a ser testada.

- **JSON (JavaScript Object Notation):** O JSON é um formato de dados baseado em texto que segue a sintaxe de objetos *JavaScript*. Este faz uso de uma estrutura baseada em pares de valores-chave para criar uma estrutura semelhante a um mapa, onde a chave tem como objetivo identificar o par e o valor é a informação atribuída a essa chave. [1]
- **XML (Extensible Markup Language):** O XML é um formato de marcação que permite definir documentos estruturados. Este utiliza uma sintaxe baseada em *tags*, bastante semelhante ao HTML. [1]
- **Serialização:** A serialização é o processo de converter uma estrutura de dados ou objeto num formato que possa ser facilmente armazenado ou transmitido. A serialização é essencial para a comunicação entre microsserviços, uma vez que permite que dados complexos sejam transformados num formato que pode ser enviado através de redes ou armazenado em sistemas de ficheiros. [2]
- **Desserialização:** A desserialização é o processo inverso da serialização, onde os dados serializados são convertidos de volta à sua representação original num objeto ou estrutura de dados utilizável. Esse processo é crucial quando um microsserviço recebe dados de outro serviço e é necessário transformá-los numa forma que possa ser manipulada internamente. [2]

III. Problem Statement

No contexto de um sistema de *e-commerce*, onde a gestão de pedidos, processamento de pagamentos e controlo de inventário são distribuídos entre microsserviços, a serialização e desserialização de dados desempenham um papel crucial na comunicação entre essas partes do sistema. Durante eventos de alta procura, como o *Black Friday*, o volume de pedidos pode aumentar exponencialmente e, para garantir que o sistema possa processar e responder rapidamente a esses pedidos, é essencial que os formatos de dados utilizados na comunicação sejam bastante rápidos. A escolha entre JSON e XML para representar esses dados pode ter um impacto significativo no tempo de processamento.

A questão central que este trabalho visa responder é: "**O JSON é realmente mais rápido do que XML em termos de serialização e desserialização, especialmente com grandes**

volumes de dados, num cenário de microsserviços?". A hipótese a ser testada é que "**JSON é mais rápido que XML em termos de serialização e desserialização, especialmente com grandes volumes de dados**", devido à sua sintaxe mais simples e ao menor overhead na representação de dados.

Neste cenário, o sistema de *e-commerce* gere diferentes volumes de pedidos, que variam de 10.000 a 1.000.000, simulando tanto situações comuns quanto picos extremos de vendas.

IV. Experimental Setup

Para realizar os testes de desempenho entre os formatos de representação de dados JSON e XML, foi elaborado um ambiente de testes controlado. O objetivo é medir o tempo de serialização e desserialização de objetos que representam pedidos num sistema de *e-commerce*.

A. Configuração do Ambiente

As experiências foram conduzidas no seguinte ambiente:

- **Linguagem de Programação**: Java, utilizando a biblioteca Jackson para serialização/desserialização em JSON e XML.
- **Formatos de Dados**: JSON e XML, ambos amplamente utilizados em arquiteturas de microserviços.
- **Bibliotecas**:
  - JSON: Jackson 2.15.0
  - XML: Jackson 2.15.0
- **Hardware**:
  - Processador: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
  - RAM: 32 GB
  - Armazenamento: SSD
  - Sistema Operativo: Microsoft Windows 11 Pro

B. Objetos de Dados

Os dados utilizados consistem num objeto `DailyRecord` que contém múltiplos objetos `Order`. O tamanho do objeto `DailyRecord` varia da seguinte forma:

- 10000 pedidos
- 100000 pedidos
- 1000000 pedidos

Cada objeto `Order` contém os seguintes atributos:

- `username` (String)
- `date` (String)
- `id` (String)

C. Procedimento de Teste

Para cada formato de representação de dados (JSON e XML), os seguintes passos foram executados:

1) **Serialização**: O objeto `DailyRecord` é serializado para o respetivo formato.
2) **Desserialização**: Os dados serializados desserializados de volta para a estrutura original do objeto `DailyRecord`.

Este processo é repetido cinco vezes para cada tamanho de dados (10000, 100000 e 1000000 pedidos) e para ambos os formatos, JSON e XML.

D. Métricas Coletadas

Para cada teste, as seguintes métricas foram medidas:

- **Tempo de Serialização**: Tempo necessário para converter o objeto `DailyRecord` para JSON ou XML.
- **Tempo de Desserialização**: Tempo necessário para converter os dados JSON ou XML de volta para o objeto `DailyRecord`.

E. Controlos Experimentais

- A JVM **é reiniciada** entre os diferentes casos de teste, o que significa que o cache e as otimizações do JIT **não** interferem com os resultados da comparação entre JSON e XML. Assim, este cenário aproxima-se a um ambiente de microsserviços real.

V. Results

A. Dados de Desempenho

Os gráficos 1 e 2 apresentam os tempos de serialização e desserialização para os formatos JSON e XML em diferentes tamanhos de dados.

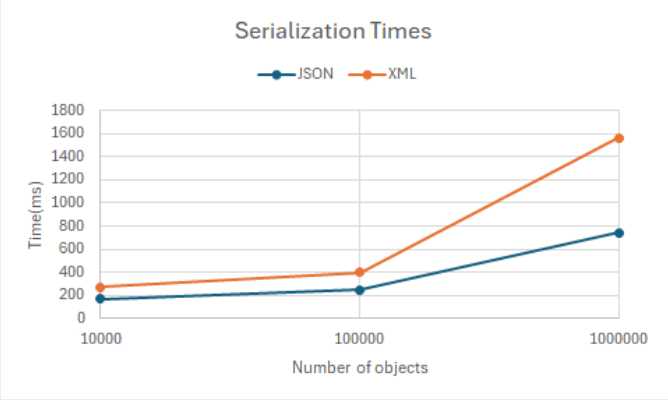

Fig. 1: Tempos de serialização para JSON e XML

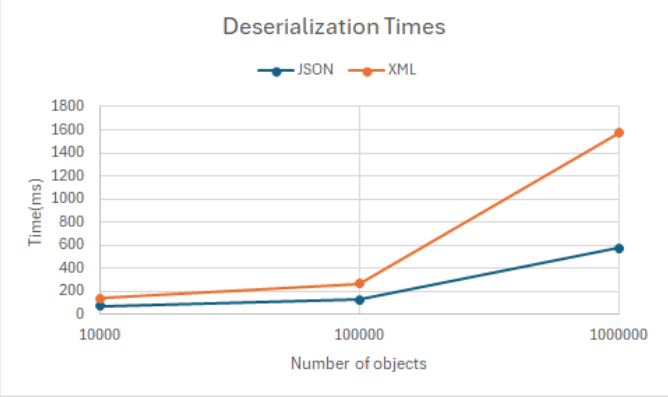

Fig. 2: Tempos de desserialização para JSON e XML

B. Dados do Impacto do Reset da JVM

Nos testes realizados, foi analisado o impacto de reiniciar ou não a JVM entre as execuções dos testes de serialização

e desserialização. As Figuras 3 e 4 apresentam os resultados comparativos.

```
JSON:
   (2024/10/05 09:45:32) 1000: serialization: 246.0737ms | deserialization: 48.607ms
   (2024/10/05 09:45:48) 1000: serialization: 221.0876ms | deserialization: 64.2772ms
   (2024/10/05 09:46:02) 1000: serialization: 226.7196ms | deserialization: 59.0907ms
   (2024/10/05 09:46:17) 1000: serialization: 241.9947ms | deserialization: 59.9498ms
   (2024/10/05 09:46:44) 1000: serialization: 291.1899ms | deserialization: 89.5547ms
XML:
   (2024/10/05 09:47:00) 1000: serialization: 359.5817ms | deserialization: 108.2594ms
   (2024/10/05 09:47:15) 1000: serialization: 301.3685ms | deserialization: 115.4493ms
   (2024/10/05 09:47:31) 1000: serialization: 296.6995ms | deserialization: 99.0839ms
   (2024/10/05 09:47:46) 1000: serialization: 303.9642ms | deserialization: 108.1522ms
   (2024/10/05 09:48:01) 1000: serialization: 301.0714ms | deserialization: 110.6489ms
```

Fig. 3: Resultados com reset da JVM

```
JSON:
   (2024/10/03 17:35:51) 1000: serialization: 237.2567ms | deserialization: 66.5202ms
   (2024/10/03 17:35:51) 1000: serialization: 8.266ms | deserialization: 9.9012ms
   (2024/10/03 17:35:51) 1000: serialization: 5.5539ms | deserialization: 5.6714ms
   (2024/10/03 17:35:51) 1000: serialization: 7.2547ms | deserialization: 5.1583ms
   (2024/10/03 17:35:51) 1000: serialization: 4.9127ms | deserialization: 5.2007ms

XML:
   (2024/10/03 17:36:04) 1000: serialization: 305.3511ms | deserialization: 97.3235ms
   (2024/10/03 17:36:05) 1000: serialization: 11.8206ms | deserialization: 18.4826ms
   (2024/10/03 17:36:05) 1000: serialization: 10.3592ms | deserialization: 9.2197ms
   (2024/10/03 17:36:05) 1000: serialization: 9.4329ms | deserialization: 7.7505ms
   (2024/10/03 17:36:05) 1000: serialization: 7.6429ms | deserialization: 7.8768ms
```

Fig. 4: Resultados sem reset da JVM

VI. Discussion

Os gráficos 1 e 2 apresentados na secção de resultados demonstram claramente as diferenças de velocidade entre JSON e XML, tanto para serialização quanto para desserialização, em diferentes volumes de dados simulando um cenário de microsserviços de *e-commerce*.

A. Serialização

No gráfico de tempos de serialização (Fig. 1), observamos que, para volumes pequenos de dados, tanto JSON quanto XML apresentam tempos de resposta semelhantes. No entanto, à medida que o número de objetos aumenta, JSON demonstra-se significativamente mais rápido. Isso corrobora a hipótese inicial de que JSON, devido à sua estrutura mais simples e menor overhead, é mais rápido para serializar grandes volumes de dados.

B. Desserialização

Em termos de desserialização (Fig. 2), o padrão é semelhante. O JSON apresenta um crescimento mais suave no tempo de desserialização em comparação com XML. Novamente, para volumes pequenos, as diferenças são mínimas, contudo, à medida que o número de objetos aumenta, o tempo de desserialização de XML cresce rapidamente. Para volumes de 1.000.000 de objetos, o tempo de desserialização de XML é mais do dobro do de JSON.

C. Impacto do Reset da JVM

Durante os testes, observámos que os tempos de execução após a primeira iteração eram significativamente menores, como mostra a Figura 4. Após investigação, identificámos que este comportamento estava relacionado com o mecanismo de compilação Just-In-Time (JIT) e o sistema de caching da JVM [3] [4]. Inicialmente, utilizávamos um loop *for* para executar as cinco iterações dos testes sem reiniciar a execução, o que permitiu à JVM otimizar as execuções subsequentes. Para obter resultados mais consistentes e eliminar o efeito do JIT e do caching, passámos a reiniciar a máquina entre cada iteração.

Além disso, esta abordagem de reiniciar o ambiente entre cada teste simula de forma mais precisa um cenário de microsserviços, onde cada serviço é executado de forma isolada e não se beneficia diretamente das otimizações decorrentes de execuções contínuas.

VII. Conclusion

Este estudo teve como objetivo avaliar a velocidade de JSON em comparação com XML em termos de serialização e desserialização dentro de um cenário de microsserviços de *e-commerce*, especialmente durante eventos de alta procura, como o Black Friday. Os resultados obtidos demonstraram que JSON se mostrou consistentemente mais rápido que o XML, confirmando a hipótese inicial.

Além disso, a investigação sobre o impacto do reset da JVM salientou a importância de controlar as variáveis do ambiente de teste. A decisão de reiniciar a JVM entre as iterações foi fundamental para garantir a consistência dos resultados, refletindo um cenário mais realista de microsserviços, onde cada componente opera de forma isolada. Esta abordagem não apenas validou a eficácia da comparação entre os formatos, mas também enfatizou a relevância de considerar o ambiente de execução ao medir o desempenho.

Em suma, os resultados deste estudo sublinham a preferência por JSON como formato de dados em contextos que exigem alta velocidade na comunicação, especialmente em sistemas distribuídos de grande escala. Para futuras investigações, seria interessante explorar outras variáveis, como o impacto de diferentes bibliotecas de serialização e desserialização, bem como testar o desempenho em diferentes linguagens de programação e ambientes de execução, a fim de expandir a compreensão sobre as melhores práticas na escolha de formatos de dados em microsserviços.

References

[1] Amazon Web Services, "The difference between json and xml," 2023, accessed: 2024-10-05. [Online]. Available: https://aws.amazon.com/pt/compare/the-difference-between-json-xml/

[2] Hazelcast, "Serialization," 2023, accessed: 2024-10-05. [Online]. Available: https://hazelcast.com/glossary/serialization/

[3] Oracle, "Understanding the jit compiler," 2023, accessed: 2024-10-05. [Online]. Available: https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/unde

[4] IBM, "Jit compiler reference," 2023, accessed: 2024-10-05. [Online]. Available: https://www.ibm.com/docs/en/sdk-java-technology/8?topic=reference-jit-compiler