

Avaliação Comparativa de Desempenho entre Serviços de Mensageria em Ambiente de Computação em Nuvem

Lucas Bicalho Oliveira¹, Álvaro Luiz Fazenda¹

¹Instituto de Ciência e Tecnologia – Universidade Federal de São Paulo (UNIFESP)
São José dos Campos, SP – Brasil

{lucas.bicalho, alvaro.fazenda}@unifesp.br

Abstract. *Asynchronous communication is a key component of microservice-based architectures. This paper presents a comparative evaluation between the RabbitMQ and Amazon Simple Queue Service (SQS) messaging services using a customized benchmark. Latency and throughput were analyzed under different workload conditions. The results show that RabbitMQ achieved lower latency and higher throughput, while SQS provides advantages related to its managed cloud-native model. The findings highlight the main trade-offs between performance and operational simplicity when adopting messaging solutions for distributed applications.*

Resumo. *A comunicação assíncrona é um elemento essencial em arquiteturas baseadas em microsserviços. Este artigo apresenta uma avaliação comparativa entre os serviços de mensageria RabbitMQ e Amazon Simple Queue Service (SQS) utilizando um benchmark customizado. Foram analisadas métricas de latência e throughput em diferentes cenários de carga. Os resultados indicam que o RabbitMQ apresentou menores latências e maiores taxas de processamento, enquanto o SQS se destaca pelas vantagens associadas ao modelo gerenciado de computação em nuvem. Os achados evidenciam os principais trade-offs entre desempenho e simplicidade operacional na adoção de soluções de mensageria para aplicações distribuídas.*

1. Introdução

A computação em nuvem se destaca atualmente como uma tecnologia que agrega muitas vantagens para seus usuários, sejam empresas, instituições de ensino ou contas pessoais. A capacidade de armazenamento de grandes volumes de dados e disponibilização de recursos de sistemas computacionais em pequena ou grande escala sem a necessidade do gerenciamento direto do proprietário dos dados ou programas são apenas alguns dos benefícios possíveis [Amazon Web Services 2023].

Arquiteturas baseadas em microsserviços têm sido amplamente adotadas em aplicações distribuídas devido à sua capacidade de promover escalabilidade, flexibilidade e desacoplamento entre componentes. Nesse contexto, mecanismos de comunicação assíncrona desempenham papel fundamental na integração entre serviços, tornando os sistemas de mensageria elementos críticos para o desempenho e a confiabilidade das aplicações. A crescente oferta de serviços nativos de nuvem amplia as possibilidades de implementação dessas arquiteturas, mas também introduz desafios relacionados à seleção das tecnologias mais adequadas para cada cenário de uso.

Diante desse contexto, este trabalho apresenta uma avaliação comparativa entre o RabbitMQ e o Amazon Simple Queue Service (SQS) em um ambiente experimental baseado em microsserviços. O objetivo é analisar o comportamento dessas soluções de mensageria em um cenário de comunicação assíncrona entre serviços, contribuindo para a compreensão de suas características e adequação a arquiteturas modernas executadas em ambientes de nuvem.

2. Referencial Teórico

O *benchmarking* é uma metodologia utilizada para avaliar o desempenho de sistemas por meio de métricas como latência, *throughput* e escalabilidade, permitindo comparações controladas entre arquiteturas e tecnologias de mensageria [Kounev et al. 2020]. Em ambientes distribuídos e de computação em nuvem, *benchmarks* devem utilizar cargas de trabalho reproduzíveis e minimizar a influência de variáveis externas, garantindo medições consistentes e comparáveis. Nesse contexto, sistemas de mensageria como RabbitMQ e Amazon SQS desempenham papel central em arquiteturas desacopladas, viabilizando comunicação assíncrona, redução do acoplamento entre serviços e suporte a aplicações com alta disponibilidade e grande volume de mensagens [Nobre et al. 2020].

O RabbitMQ é um *broker* de mensageria tradicional, baseado no protocolo AMQP (*Advanced Message Queuing Protocol*), que utiliza conexões persistentes entre produtores, *broker* e consumidores. Nesse modelo, após o estabelecimento da conexão, o *broker* pode entregar mensagens de forma contínua aos consumidores, caracterizando um mecanismo predominantemente baseado em *push* [Ćatović et al. 2022]; [Dobbelaere and Esmaili 2017]. Tal abordagem tende a favorecer menores latências, uma vez que o consumidor não precisa requisitar explicitamente novas mensagens a cada iteração [Ruseno et al. 2025].

Por outro lado, o Amazon SQS é um serviço de mensageria gerenciado e acessado por chamadas HTTP à API da Amazon Web Services. O modelo de comunicação do SQS é baseado em *polling*, no qual os consumidores realizam requisições periódicas para verificar a disponibilidade de mensagens na fila [Amazon Web Services 2024]. Mesmo quando configurado com mecanismos de *long polling*, o SQS não estabelece conexões persistentes no mesmo sentido que *brokers* tradicionais, o que introduz um custo adicional associado à latência de rede e ao ciclo de requisição-resposta.

3. Resultados e Discussões

Os experimentos utilizaram um *benchmark* customizado, inspirado no OpenMessaging Benchmark Framework [Linux Foundation and OpenMessaging 2022], para comparar as métricas de latência e *throughput* entre RabbitMQ e Amazon SQS. Os testes foram executados em instâncias Amazon EC2 com configurações padrões do plano gratuito. O RabbitMQ foi implantado em uma instância exclusiva, enquanto produtores e consumidores executaram separadamente, simulando comunicação distribuída. No SQS, foram avaliadas operações em lote (*batch*), variando-se o tamanho das mensagens e o *batch size* para comparação dos modelos *push* e *polling*.

Em síntese, os resultados do *benchmark* com RabbitMQ demonstram que, mesmo com o aumento do tamanho das mensagens, o sistema mantém desempenho consistente e latências reduzidas. Esses resultados contrastam com aqueles observados no Amazon

SQS, cuja arquitetura baseada em *polling* e limites de processamento em lote introduzem uma escala temporal distinta. A Figura 1 apresenta a comparação entre as latências médias dos dois *brokers*. Observa-se uma diferença expressiva entre os resultados, com o RabbitMQ apresentando latência substancialmente menor quando comparado ao SQS, mesmo utilizando o maior tamanho de lote avaliado. Devido à ordem de grandeza distinta entre os valores — latência na ordem de milissegundos para o RabbitMQ e de segundos para o SQS — foi adotada uma escala logarítmica, a fim de possibilitar uma visualização adequada e comparável dos dados no mesmo gráfico.

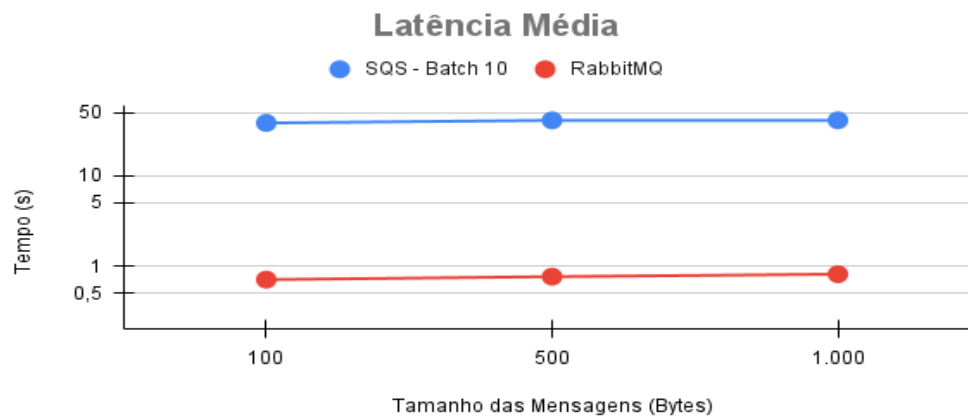


Figura 1. Comparação entre Latências Médias medidas para RabbitMQ e SQS com *batch* de 10 mensagens.

De forma análoga, a Figura 2 ilustra a comparação dos *throughputs* de produção e consumo para ambos os *brokers*. Novamente, a utilização de escala logarítmica se mostrou necessária, dada a diferença significativa entre os valores observados, expressos em mensagens por segundo. O gráfico evidencia que o RabbitMQ alcança taxas de *throughput* consideravelmente superiores às do SQS, tanto na produção quanto no consumo de mensagens.

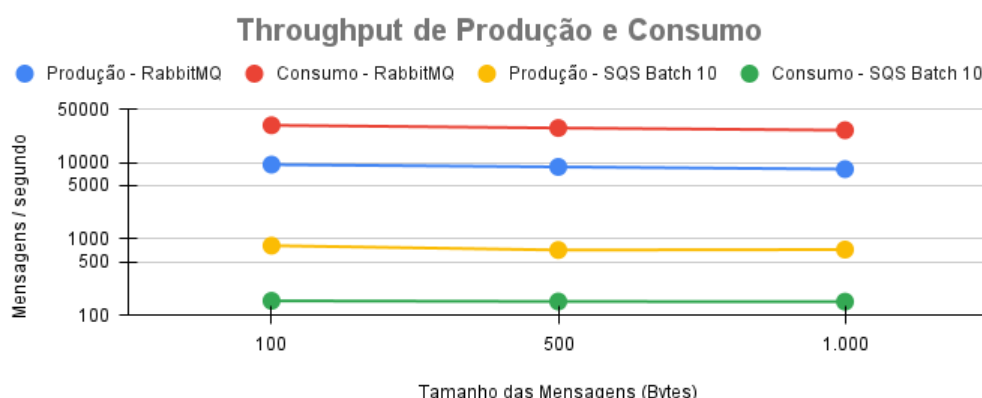


Figura 2. Comparação entre *Throughputs* de Produção e Consumo para RabbitMQ e SQS com *batch* de 10 mensagens.

De modo geral, os resultados demonstram que, embora o uso de *batching* no SQS contribua para melhorias de desempenho, o RabbitMQ mantém uma vantagem significativa em termos de latência e *throughput* nos cenários avaliados, o que é consistente com sua natureza de *broker* orientado a mensageria *push* e operação local, em contraste com o modelo gerenciado e distribuído do Amazon SQS.

Embora o RabbitMQ tenha apresentado desempenho superior nos cenários avaliados, a comparação entre as soluções não deve se restringir às métricas de latência e *throughput*. Enquanto o RabbitMQ exige infraestrutura dedicada e atividades de administração, monitoramento e escalabilidade, o Amazon SQS adota um modelo de serviço gerenciado com cobrança baseada no número de requisições realizadas. Dessa forma, o RabbitMQ tende a apresentar custos mais previsíveis em cenários de alta utilização, ao passo que o SQS oferece maior elasticidade operacional e reduz o esforço de gerenciamento. Assim, a escolha entre as tecnologias envolve um *trade-off* entre desempenho, custo operacional e simplicidade de administração.

4. Conclusão e Trabalhos Futuros

Os resultados obtidos evidenciam diferenças significativas entre as abordagens analisadas, reforçando a importância da escolha adequada de tecnologias de mensageria no contexto de aplicações baseadas em microsserviços e destacando os impactos diretos dessa escolha sobre a escalabilidade e a eficiência operacional da plataforma. Como limitação, destaca-se que os experimentos foram conduzidos em cenários controlados, não abrangendo cargas reais de produção. Como trabalhos futuros, pretende-se ampliar a avaliação realizada por meio da inclusão de serviços de banco de dados em nuvem e da análise de métricas relacionadas à eficiência energética e custo financeiro para diferentes cargas, incluindo a integração com plataformas que utilizam arquiteturas distribuídas de microsserviços.

Referências

- Amazon Web Services (2023). Seis vantagens da computação em nuvem. docs.aws.amazon.com/pt_br/whitepapers/latest/aws-overview/six-advantages-of-cloud-computing.html. Acesso em: 30 out. 2023.
- Amazon Web Services (2024). Amazon sqs short and long polling. docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-short-and-long-polling.html. Acesso em: 02 jan. 2026.
- Ćatović, A., Buzadžić, N., and Lemes, S. (2022). Microservice development using rabbitmq message broker. *Science, engineering and technology*, 2(1):30–37.
- Dobbelaere, P. and Esmaili, K. S. (2017). Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper. In *Proc. 11th ACM Int. Conference on Distributed and Event-Based Systems*, pages 227–238.
- Kounev, S., Iosup, A., and Brustoloni, M. R. (2020). Benchmarking in the cloud: Challenges and opportunities. *ACM Computing Surveys*, 52(4):1–34.
- Linux Foundation and OpenMessaging (2022). Openmessaging benchmark framework. openmessaging.cloud/docs/benchmarks/. Acesso em: 30 jul. 2025.
- Nobre, C., Maia, G., and Bessani, A. (2020). A comparative study of distributed messaging systems: Kafka, rabbitmq, and activemq. *J. Systems and Software*, 168:110646.
- Ruseno, N., Chagas, F. S., Fas-Millán, M.-Á., and Bechina, A. A. A. (2025). Latency analysis of push-pull and publish-subscribe communication protocols in u-space systems. *Electronics*, 14(12):2453.