RabbitMQ Benchmarking in .NET 8

Insights for .NET developers

Václav Bílý | Elliot Verstraelen

Introduction

The development landscape constantly evolves, and .NET developers are increasingly tasked with building robust and scalable applications. As system demands grow, traditional centralized architectures often struggle to keep pace. This is where **event-driven systems and messaging** emerge as a compelling solution, offering several key advantages for .NET developers:

Scalability: Event-driven architectures, enabled by decentralized messaging, excel at handling high volumes of data and concurrent users. This scalability is crucial for modern applications experiencing unpredictable load spikes.
Decentralization: Unlike centralized architectures with single points of failure, decentralized messaging distributes communication across a network. This enhances fault tolerance and ensures continued operation even if individual components experience issues.
Serverless-friendliness: Decentralized messaging aligns perfectly with the serverless paradigm, where developers can focus on application logic without managing infrastructure. This streamlines development and reduces operational overhead.

Beyond these broader benefits, .NET developers have specific reasons to consider decentralized messaging:

Performance Efficiency: A key concern for developers is the potential performance penalty associated with message-based communication compared to familiar REST APIs. This report aims to address this concern by benchmarking the performance of decentralized messaging with RabbitMQ against traditional HTTP Web APIs in a .NET 8 environment.

MassTransit Adoption: MassTransit is a popular library among .NET developers, particularly for building large-scale distributed applications. This report explores the performance implications of using MassTransit with RabbitMQ, providing valuable insights for developers already utilizing this established library.

This report helps .NET developers understand how decentralized messaging performs compared to traditional methods, enabling them to make better choices when creating scalable and strong applications.

Background

This section provides background information about the technologies used in this report.

.NET 8 is a high-performance, cross-platform, and open-source framework, .NET 8 provides a versatile environment for developing and testing the communication approaches compared in this research. .NET is very versatile and is a popular choice for building large applications and systems.

WebAPI architecture serves as the baseline for centralized communication in this study. This standard approach utilizes HTTP requests and responses for communication between applications. The performance of Web API will be compared against decentralized messaging solutions.

RabbitMQ: As a popular open-source message broker, RabbitMQ underpins the decentralized messaging strategies evaluated in this report. It offers a robust infrastructure for asynchronous communication, enabling message queuing, routing, and exchange mechanisms. Leveraging RabbitMQ allows for the implementation of scalable and fault-tolerant communication patterns.

MassTransit: MassTransit is a popular .NET library that provides simplified abstraction above inter-service messaging. It offers abstractions and streamlined message handling, potentially impacting performance.

BenchmarkDotNet: This research uses BenchmarkDotNet for precise performance measurements in the .NET environment, ensuring reliable results through multiple benchmarks and outlier filtering.

Docker (Compose): This report employed Docker Compose to guarantee a consistent testing environment for RabbitMQ benchmarking. We ensured a replicable setup by configuring containers for RabbitMQ and a custom testing Web API endpoint within a Docker Compose YAML file, minimizing external influences on performance measurements.

Altair: We used the Altair Python package to create vega-lite charts and visualizations to present the results.

Benchmarking setup

This section outlines the methodology employed to evaluate the performance of different communication approaches in this research. We designed two distinct benchmarks to isolate and analyze specific aspects:

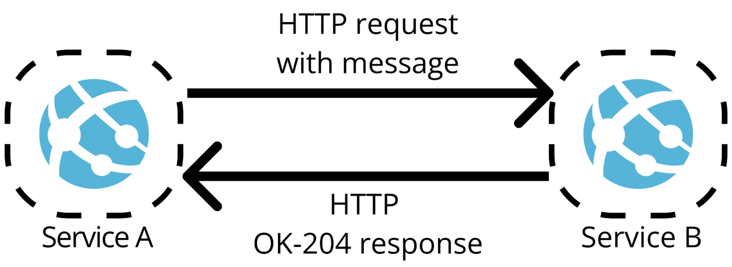
Benchmark 1: Centralized vs. Decentralized Messaging

This benchmark compares the performance of traditional HTTP message transfers via a custom Web API endpoint to message transfers utilizing RabbitMQ as a message broker.

By measuring metrics like latency and throughput, we aimed to assess the potential performance gains (or trade-offs) associated with decentralized messaging in a .NET 8 environment.

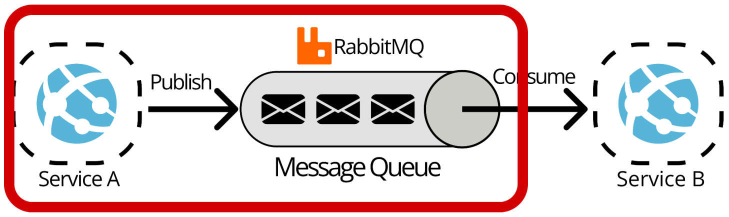
HTTP

Simple HTTP message transfer using Asp.Net Core WebApi. Then, we are awaiting a response acknowledging the received data.



Rabbit - Simple one-way test

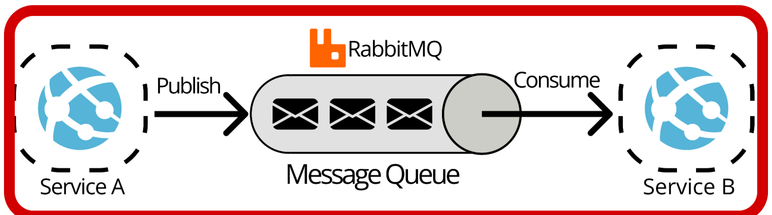
Transfer messages only to the message broker and wait for acknowledgment from the



broker.

Rabbit-RT (round trip)

Transferring message to Service B



Benchmark 2: MassTransit vs. RabbitMQ Client

This benchmark specifically investigates the performance overhead introduced by the MassTransit library.

We compared the performance of sending and receiving messages directly using the official RabbitMQ client library versus utilizing MassTransit, which offers a higher level of abstraction for interacting with RabbitMQ.

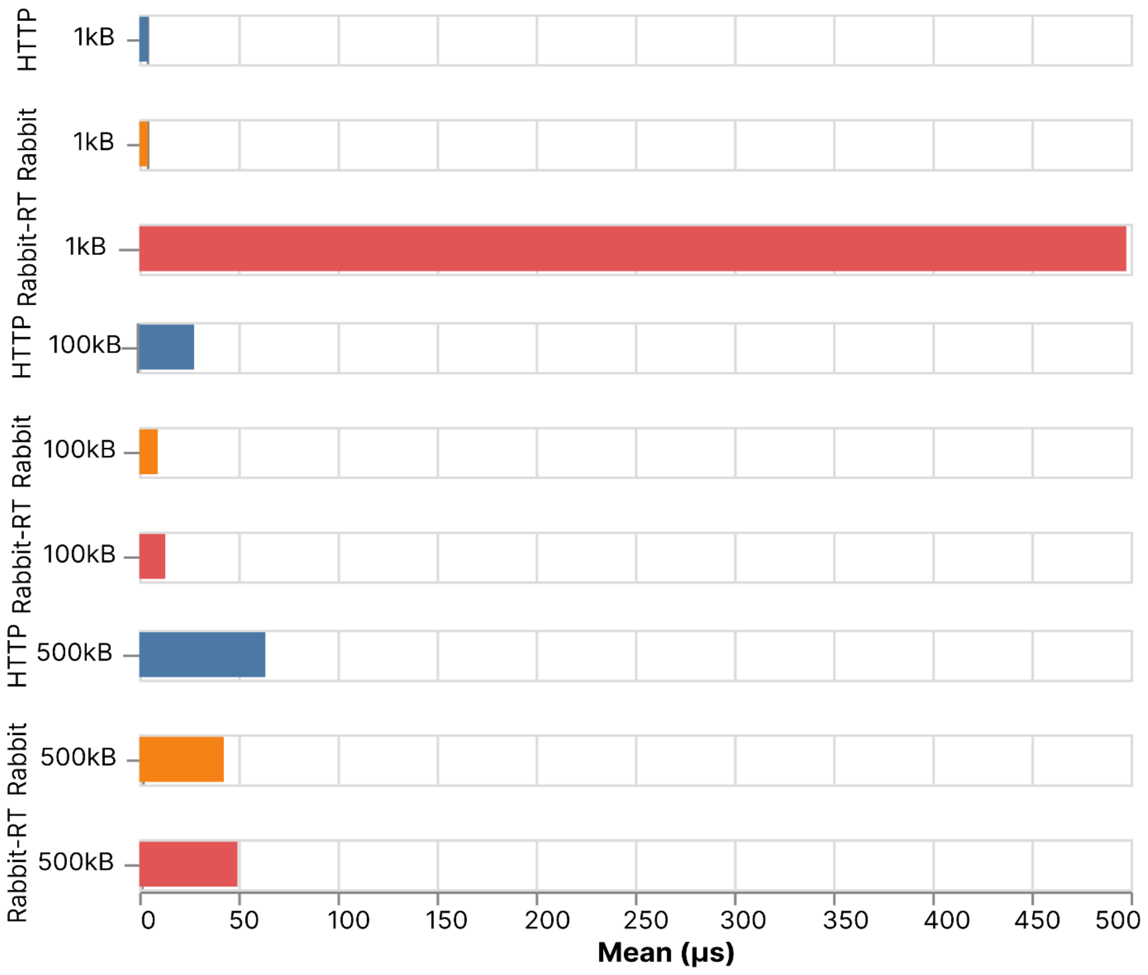
This comparison helps developers understand the potential performance impact of leveraging libraries like MassTransit for simplified message broker interaction.

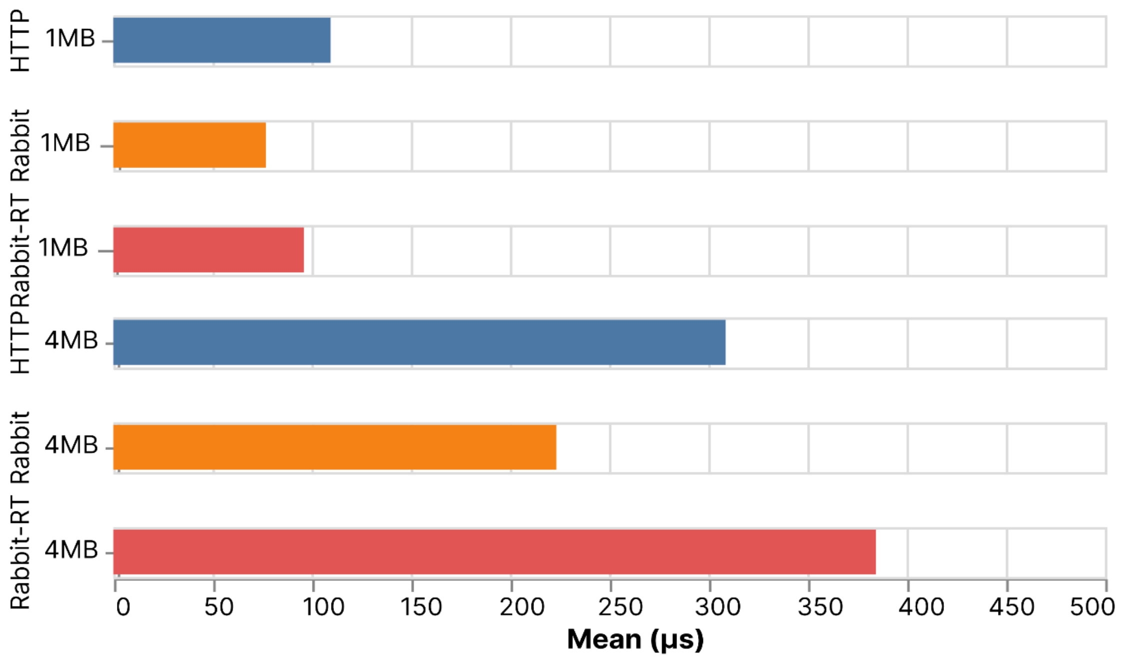
Results

All results can be found in the repository on GitHub in the Results directory.

Benchmark 1 - Comparing RabbitMQ against HTTP

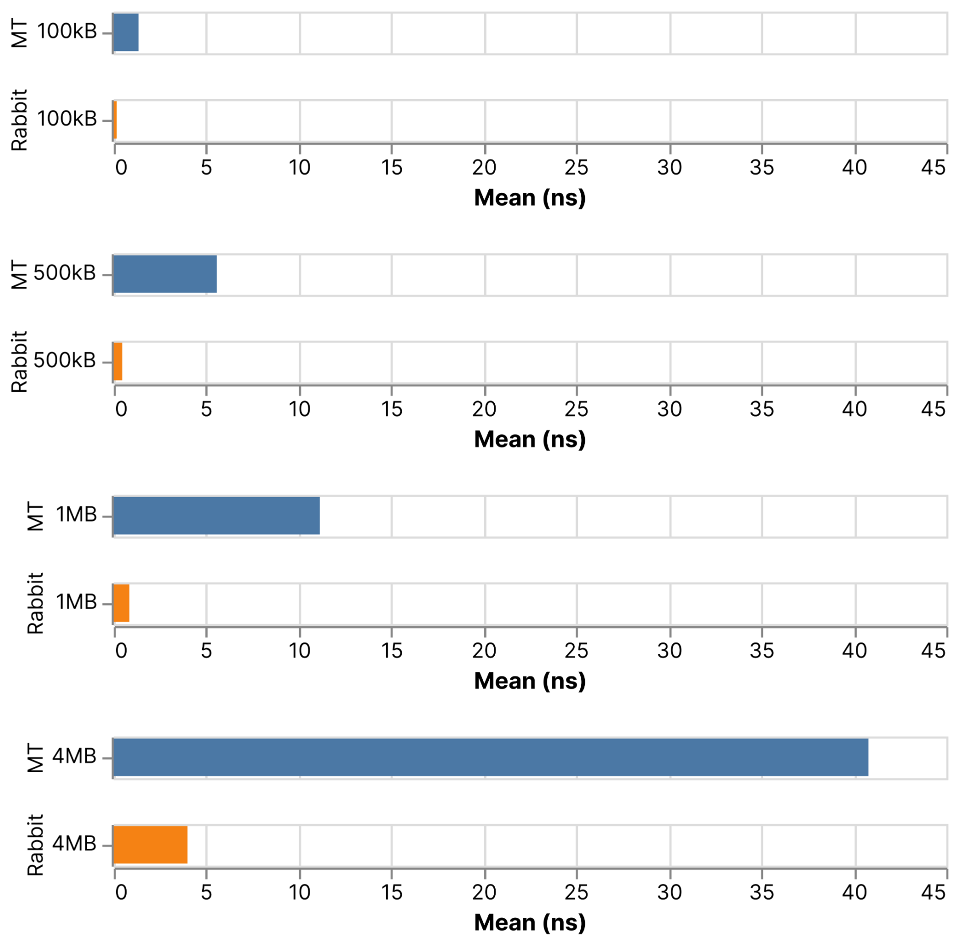
The results of this benchmark show very comparable results of RabbitMQ to HTTP with relatively low differences in microseconds. Moreover, more favoring RabbitMQ. The only outlier is when transmitting a 1kB message, we haven’t been able to explain this behavior. Or results are observable in the chart below.


HTTP
1kB
Rabbit
1kB
Rabbit-RT
1kB
HTTP
100kB
Rabbit
100kB
Rabbit-RT
100kB
HTTP
500kB
Rabbit
500kB
Rabbit-RT
500kB
0
50
100
150
200
250
300
350
400
450
500
Mean (μs)


HTTP
1MB
Rabbit
1MB
Rabbit-RT
1MB
HTTP
4MB
Rabbit
4MB
Rabbit-RT
4MB
0
50
100
150
200
250
300
350
400
450
500
Mean (μs)

Benchmark 2 - RabbitMQ-Client against MassTransit + RabbitMQ

The results of this benchmark show relatively significant differences between the RabbitMQ.Client and MassTransit are used to transmit messages. But the results should not probably be worrying .NET developers, because the differences are in matters of nanoseconds, and the pros of using a library such as MassTransit bring many more benefits for developers (maintainability, abstraction, interchangeable transfer services, etc.) that outweigh the slight penalty of a few nanoseconds.



Future considerations

Extending this testing/benchmarking is viable in a few ways.

First, more services could be added for comparison in the first benchmark because HTTP WebApi does not cover a wide range of currently used technologies such as GraphQL or GRPC. To provide more context on the viability of using event-driven systems.

Second, a possible extension could be to simulate more traffic in the background for message broker(s) to get closer to the real-world use cases. But it’s hard to do in a deterministic and reproducible way. Then, we could test more interesting scenarios and configurations of the message brokers cluster or federations.

Conclusion

This report investigated the performance benefits of decentralized messaging compared to traditional Web APIs in .NET 8 development. Our findings demonstrate that RabbitMQ, a popular message broker, offers significant performance improvements, particularly for asynchronous communication scenarios. This highlights the suitability of RabbitMQ for building scalable and responsive applications that handle high message volumes.

The report also explored the impact of the MassTransit library, which simplifies interaction with RabbitMQ. While MassTransit introduces a measurable overhead, it's important to note that it falls within the nanosecond range. This overhead is negligible in real-world applications dealing with regular messages smaller than 1MB. Therefore, MassTransit's convenience and developer productivity benefits outweigh the minimal performance impact for most use cases.

This research underscores the advantages of adopting decentralized messaging with RabbitMQ for .NET 8 development. RabbitMQ's performance gains and MassTransit's ease of use provide a compelling combination for building robust and scalable distributed applications.

Resources

- Dot.Net
- ASP.NET Core | Open-source web framework for .NET
- RabbitMQ
- BenchmarkDotNet
- MassTransit
- Vega-Altair
- Docker Compose overview