Escalabilidade do MQTT

Bruno A. Pires[1], **Arthur C. Estevão**[2]

[1]Universidade Federal do Espírito Santo (UFES)
Vitória – ES – Brazil

2

bruno.pires@edu.ufes.br, arthur.estevao@edu.ufes.br

Abstract. *This project analyzes the scalability of a local MQTT server using a device simulator that sends data via MQTT. Tests include progressively increasing the message payload, analyzing CPU and memory consumption with an increasing number of publications, and evaluating the impact of multiple subscribers. Results will be presented in a report following the SBC format, along with the codes and usage instructions.*

Resumo. *Este projeto analisa a escalabilidade de um servidor MQTT local por meio de um simulador de dispositivos que enviam dados via MQTT. Os testes incluem o aumento progressivo do payload das mensagens, a análise do consumo de CPU e memória do broker com o aumento do número de publicações e a avaliação do impacto de múltiplos subscribers. Os resultados serão apresentados em um relatório conforme o formato da SBC, acompanhados dos códigos e instruções de uso.*

1. Introdução

A Internet das Coisas (IoT) tem se expandido rapidamente, conectando bilhões de dispositivos que comunicam e compartilham dados. O MQTT (Message Queuing Telemetry Transport) é um protocolo de comunicação leve, amplamente utilizado em aplicações de IoT devido à sua eficiência na troca de mensagens entre dispositivos de baixa capacidade de processamento e redes com largura de banda limitada. No entanto, com o crescente número de dispositivos conectados e a demanda por dados em tempo real, torna-se crucial avaliar a escalabilidade de servidores MQTT.

Este projeto tem como objetivo analisar a escalabilidade de um servidor MQTT local utilizando um simulador de dispositivos que enviam dados via MQTT. A escalabilidade será avaliada através de três testes principais: aumento progressivo do payload das mensagens, variação do número de publicações, e incremento do número de subscribers. Ao monitorar o consumo de CPU e memória do broker durante esses testes, buscamos identificar os limites de desempenho e possíveis gargalos.

Os resultados deste estudo fornecerão insights valiosos sobre o comportamento do servidor MQTT sob diferentes condições de carga, contribuindo para o desenvolvimento de soluções mais robustas e escaláveis para aplicações de IoT.

2. Metodologia

Para avaliar a escalabilidade de um servidor MQTT local, desenvolvemos um simulador que envia dados de sensores via MQTT, e monitoramos o consumo de CPU e memória

do broker durante diferentes testes. A metodologia está dividida em três testes principais: aumento progressivo do payload das mensagens, variação do número de publicações, e incremento do número de subscribers. A seguir, detalhamos a abordagem para cada um desses testes.

2.1. Configuração do Ambiente

O simulador foi implementado utilizando a biblioteca Python **paho-mqtt** para gerenciar as conexões MQTT, e scripts Bash para monitoramento dos recursos do sistema. O ambiente foi configurado com os seguintes parâmetros padrão definidos em um arquivo **.env**:

- **numPublishers**: Quantidade de publicadores MQTT.
- **numSubscriber**: Quantidade de subscribers MQTT.
- **simTime**: Tempo de execução dos testes (em segundos).
- **instTime**: Tempo entre instanciação de publicadores e subescritores (em milissegundos).
- **host**: Endereço do broker MQTT.
- **port**: Porta do broker MQTT.
- **topic**: Tópico utilizado para publicação e subscrição.
- **attr**: Chave do payload da mensagem.
- **msgTime**: Tempo entre envios de mensagens (em milissegundos).
- **stdout_arquivo**: Indica se a saída deve ser salva em arquivo.

2.2. Teste de Payload

Neste teste, um cliente MQTT subscrito foi configurado para receber mensagens com payload de tamanho crescente. O script **mqtt.py** foi utilizado para gerenciar a comunicação MQTT. A função **increasing_payload** foi definida para aumentar o tamanho do payload a cada mensagem enviada, iniciando com um caractere e incrementando até encontrar o limite máximo que o broker suporta. O consumo de recursos foi monitorado e registrado.

2.3. Teste de Publisher

Para avaliar o impacto do número de publicações no consumo de recursos do broker, configuramos múltiplos publicadores MQTT utilizando o mesmo script **mqtt.py**. A frequência de publicação foi ajustada através do parâmetro **msgTime**, e o número de instâncias de publicadores foi aumentado gradativamente de acordo com o parametro parâmetro **instTime**. O script **gera-grafico.py** foi utilizado para gerar gráficos do uso de CPU e memória com base nos dados coletados.

2.4. Teste de Publisher e Subscriber

Este teste combina os elementos dos testes anteriores, aumentando gradativamente o numero de subscribers ao invez dos publishers. O objetivo é avaliar o impacto no desempenho do broker MQTT. Utilizamos múltiplas instâncias para simular o ambiente de teste e monitorar os recursos do sistema.

2.5. Execução dos Testes

Os testes foram realizados por meio dos scripts **teste_payload.sh**, **teste_pub_sub.sh** e **teste_publisher.sh**, que executam os respectivos arquivos **.py**. Esses arquivos carregam as configurações do arquivo **.env**, instanciam a classe **MQTT** e iniciam os processos de publicação e assinatura conforme os parâmetros definidos. Durante a execução dos scripts **.py**, o uso de memória e CPU do Mosquitto é monitorado e registrado no arquivo **mem.dat**. Posteriormente, esses dados são utilizados para criar um gráfico através do script **gera-grafico.py**.

2.6. Análise dos Dados

Os dados coletados foram analisados para identificar o impacto de cada variável testada no desempenho do broker MQTT. Utilizamos gráficos gerados pelo script **gera-grafico.py** para visualizar o uso de CPU e memória durante os testes. Estes resultados foram comparados para determinar os limites de escalabilidade do broker e identificar possíveis gargalos de desempenho.

3. Análise dos Resultados

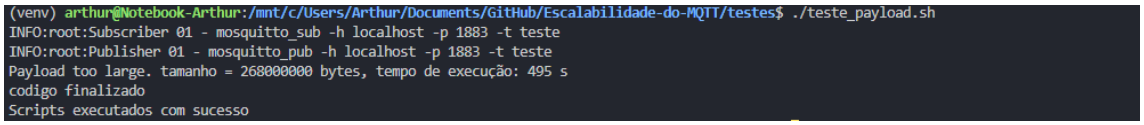
O primeiro teste realizado foi o teste de Payload, cujo objetivo era determinar o número máximo de caracteres que podem ser enviados pelo publisher ao Mosquitto. A metodologia consistiu em aumentar o tamanho do payload em incrementos de 1 MB por mensagem e verificar, ao final, o tamanho máximo enviado com sucesso. Um exemplo de mensagem publicada pode ser observado na Listagem 1.

Listing 1. Exemplo de código JSON 1 MB

```
{
    "teste_payload": "XXXXXXX",
}
```

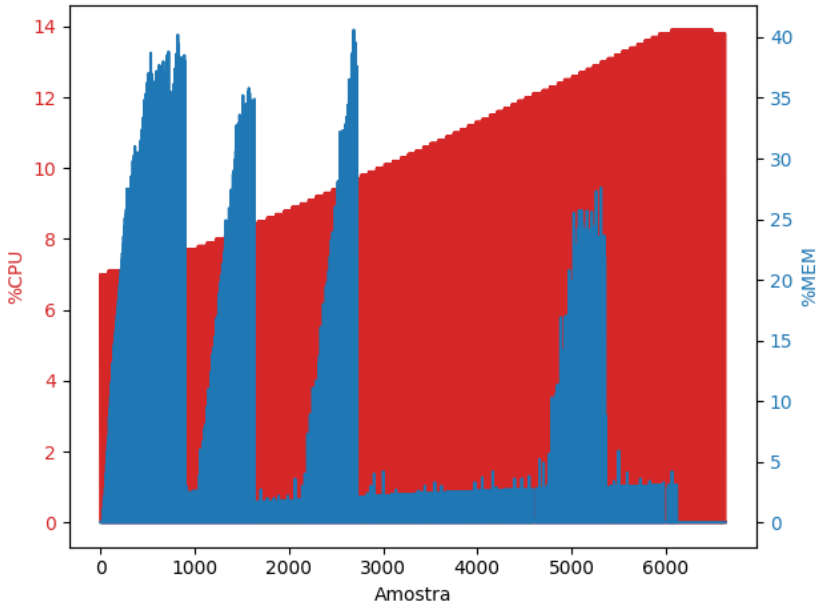
O estresse do broker levou 8 minutos e 15 segundos, consumindo 298 MB de memória (Figura 1). É importante salientar que isso não significa que toda a memória do broker foi utilizada. Conforme demonstrado na Figura 2, o broker se estabiliza em um baixo percentual de uso de memória após o estresse do payload.

Figura 1. Payload



Fonte: Produção do próprio autor.

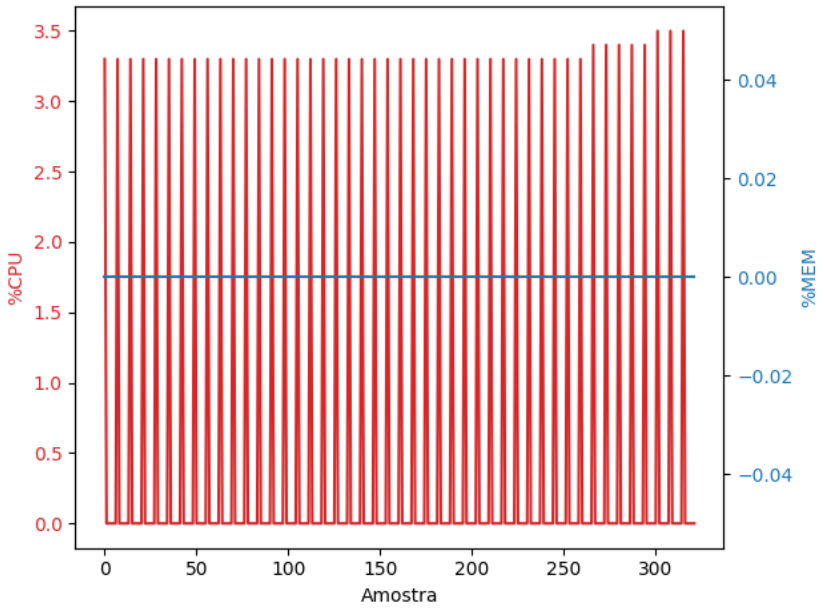
Figura 2. Payload



Fonte: Produção do próprio autor.

O segundo teste executado foi o teste de publisher para analisar o consumo de CPU e memória do broker de acordo com o aumento do número de publicações. Analisando o gráfico da Figura 3, é possível observar que a utilização da memória do broker se manteve constante e baixa. No entanto, a porcentagem de uso da CPU mostra que, a cada instância de um novo publisher, há um pico de uso que se estabiliza logo em seguida. Apesar dos picos, o valor máximo alcançado não ultrapassa 3,5% de uso da CPU.

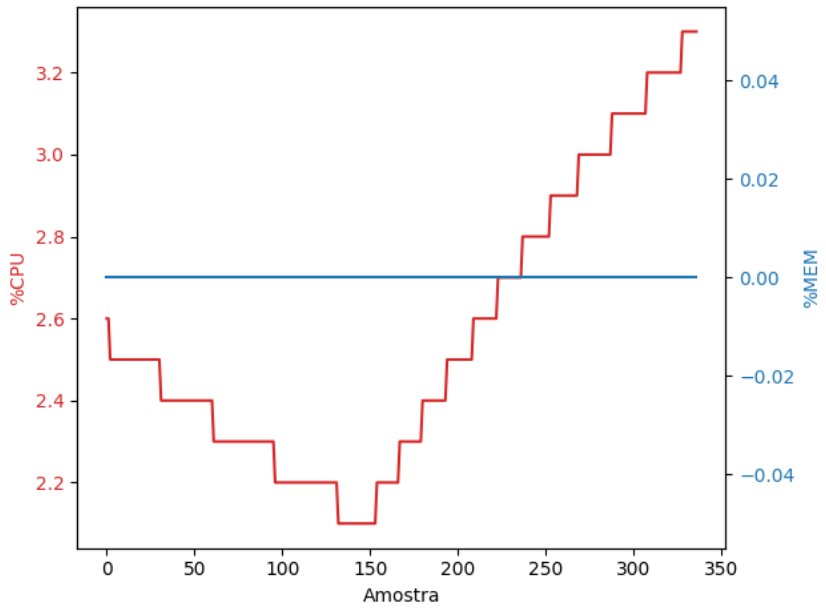
Figura 3. Publisher



Fonte: Produção do próprio autor.

O terceiro teste, que analisou o consumo de CPU e memória do broker com o aumento do número de inscritos, teste de publisher e subscriber. Os resultados da simulação são mostrados na Figura 4, a simulação foi executada instanciando 300 subscribers a cada 500 ms e 1 publisher em sequência, necessitando de cerca de 150 amostras para que todos fossem instanciados. É possível observar que, até a finalização da instanciação de todos os publishers e subscribers, a porcentagem de uso da CPU diminuiu. No entanto, ao iniciar as publicações, houve um aumento linear da porcentagem de uso da CPU.

Figura 4. Publisher Subscriber



Fonte: Produção do próprio autor.

4. Conclusão

Este projeto explorou a escalabilidade de um servidor MQTT local através de testes rigorosos. Os testes incluíram o aumento progressivo do payload das mensagens, a análise do consumo de CPU e memória do broker com o aumento do número de publicações, e a avaliação do impacto de múltiplos subscribers. Os resultados obtidos revelaram insights significativos sobre o desempenho do broker MQTT em diferentes cenários de carga.

No teste de payload, observou-se que o broker suporta eficientemente o aumento do tamanho das mensagens sem comprometer significativamente o uso de memória, alcançando um pico de 298 MB durante o estresse máximo, representado na Figura 1. Este teste demonstrou a capacidade do broker de lidar com cargas substanciais de dados sem comprometer a estabilidade.

No teste de publisher, foi constatado que o broker mantém uma utilização estável e baixa da memória, enquanto a CPU apresenta picos temporários durante a instanciação dos publishers e subscribers, conforme ilustrado na Figura 3. Apesar dos picos de uso da CPU ao instanciar novos publishers, o impacto geral permanece controlado, com um máximo de 3,5% de uso da CPU.

O teste combinado de publisher e subscriber, conforme mostrado na Figura 4, revelou que o aumento do número de inscritos inicialmente reduz a carga de CPU, mas

introduz um aumento linear após o início das publicações. Isso destaca a importância de gerenciar dinamicamente o número de clientes conectados para otimizar o desempenho do broker MQTT em cenários de alta demanda.

Em suma, os resultados desses testes fornecem uma base sólida para o desenvolvimento de estratégias de dimensionamento e otimização de servidores MQTT, essenciais para aplicações IoT que exigem escalabilidade e eficiência operacional. Mais detalhes da execução dos códigos podem ser observados no readme.md do projeto.

Referências