EP1 - Daniel Leal

Protocolo MQTT

Reads e Writes

- Arquitetura simples
  - Recebe input
  - Faz um parse da mensagem
  - Age de acordo
  - Responde

Comandos

- Os tipos comandos são identificados convertendo os 4 bits mais significativos em decimal
- Maior parte dos comandos recebe uma resposta

- CONNECT
  - Possui valor 1
  - Recebe um CONNACK como resposta
- SUBSCRIBE
  - Possui valor 8
  - Recebe um SUBACK como resposta
    - Último bit indica sucesso ou falha na “conexão” com o tópico

Comandos

- PUBLISH
  - Possui valor 3
  - Ao receber, identifica o tópico e repassa a mensagem para os subscribers
- PINGREQ
  - Possui valor 12
  - Recebe um PINGRESP como resposta

Tópicos

- Setup:
  - Criação da pasta root
  - Deleção de todos os arquivos existentes na pasta root
    - Funções de limpeza:
      - eraseTopicFilesAndFolders - recursivamente delete todos os arquivos dentro de uma pasta e chama removeTopicPipeFile no callback
      - removeTopicPipeFile - deleta a pasta do tópico

Subscribers

- É criada uma pasta com o nome do tópico, caso não exista
- Pipe é criado dentro da pasta que possui o nome do tópico
  - Nome do arquivo do pipe é o pid do processo
- Recebe um novo processo onde consome do tópico
- Processo original continua respondendo PINGREQ

Producers

- Mensagem original é escrita em todos os pipes do tópico
  - writeToTopic -> itera sobre todos os arquivos dentro da pasta do tópico e escreve a mensagem

Testes e resultados

- Os testes foram feitos utilizando o servidor dentro um container docker e utilizando o comando docker stats para colher as informações de uso de CPU e rede.
- Máquina:
  - i7-8565U 1.8 GHz
  - 8 GB de RAM
  - SSD 256GB
- Rede (Wi-fi)
  - 200 MB download (150MB em teste)
  - 100 MB upload (110MB em teste)

Sem clientes

Contexto:

- Para o teste sem clientes, o servidor ficou no ar até o fim da coleta sem receber conexões

100 clientes

Contexto:

- Para o teste com 100 clientes, foi feito um script com um loop conectando 100 clientes simultaneamente
- Até o fim da coleta, publishers escreviam no mesmo tópico que todos os subscribers

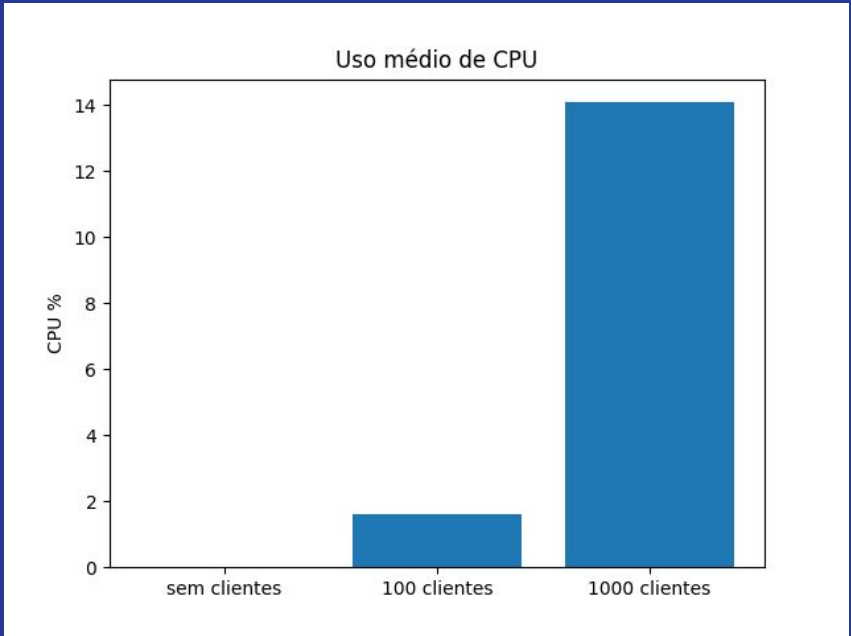
1000 clientes

Contexto:

- Para o teste com 1000 clientes, foi feito um script com um loop conectando 1000 clientes simultaneamente
- Até o fim da coleta, publishers escreviam no mesmo tópico que todos os subscribers

Resultados

Uso de CPU

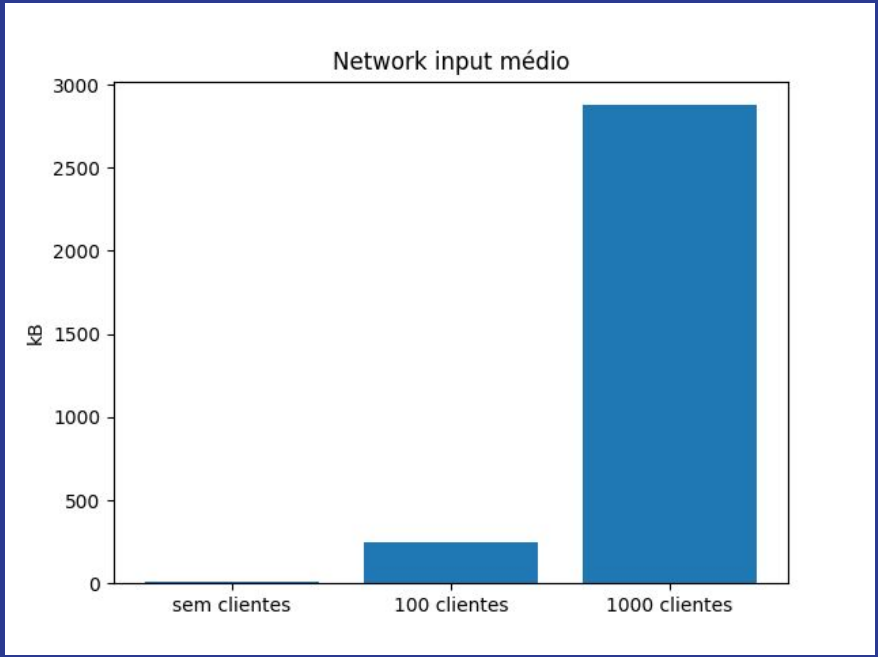


Intervalos de confiança 95%

- Sem clientes: [0, 0]
- 100 clientes: [1,57, 1,64]
- 1000 clientes: [13,33, 14, 84]

Resultados

Network Input

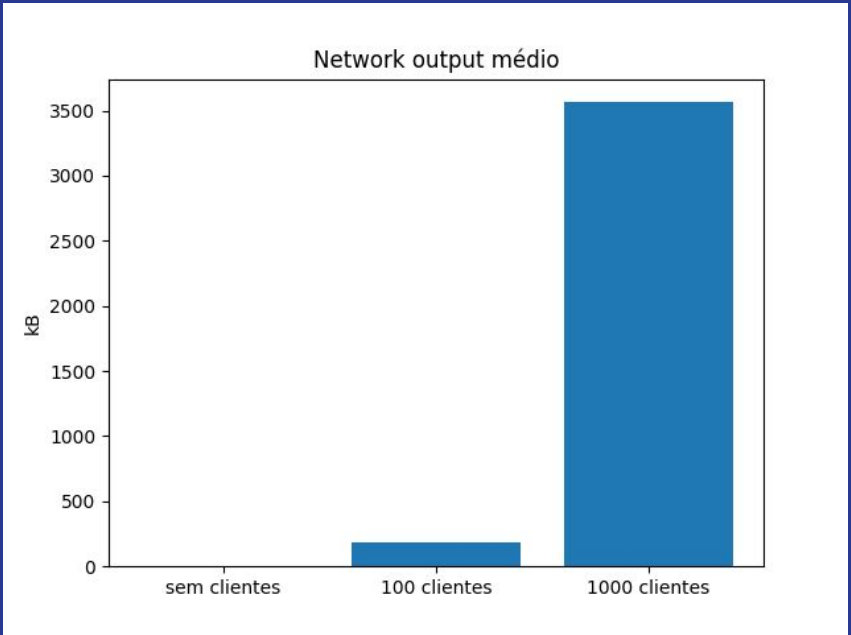


Intervalos de confiança 95%

- Sem clientes: [6.53kB, 6.81kB]
- 100 clientes: [208.02kB, 290.26kB]
- 1000 clientes: [2330.53kB, 3422.13kB]

Resultados

Network Output



Intervalos de confiança 95%

- Sem clientes: [0kB, 0kB]
- 100 clientes: [153.05kB, 217.97kB]
- 1000 clientes: [2882.18kB, 4247.28kB]

Resultados

Considerações

- Os testes se comportaram como esperado: o aumento da carga foi perceptível nos dados capturados, quanto maior a carga, maior o uso de CPU e rede.
- Para o teste sem clientes, temos um uso imperceptível de processamento e algum uso de rede para input, provavelmente pacotes que o container recebe pelo próprio docker mas o imperceptível uso para output comprova que o servidor não estava se comunicando com clientes

Resultados

Considerações

- Comparando os testes com 100 e 1000 clientes, há um crescimento de mais ou menos 13x no uso de CPU e mais de 10x para uso de rede, o que faz bastante sentido, dado que:
  - Há 10x mais processos e 10x mais arquivos para iterar e escrever
  - Há 10x mais clientes para enviar a mensagem, o que foi bem traduzido no uso superior de rede para output do que input, pois uma mensagem recebida era enviada para 1000 clientes.