Rogério da Cunha Correia

Tamyris Cabral Gomes Rocha

Wilson Gomes Nogueira Júnior

Link do Repositório: https://github.com/wilsongn/Whatslike

Relatório de Entrega: API de Mensagens (Whatslike)

1. Arquitetura da Solução

Esta solução de mensageria utiliza uma arquitetura desacoplada baseada em API REST, Kafka e Cassandra.

O fluxo de dados principal foi implementado da seguinte forma:

1. O cliente obtém um **JWT** para autenticação.
2. A **API REST** recebe a mensagem (POST), valida e publica um evento no tópico `messages` do Kafka.
3. O **Router Worker** (consumidor Kafka) processa o evento, persiste a mensagem no Cassandra e atualiza seu status.
4. A **Listagem** de mensagens consulta o Cassandra diretamente para otimizar a leitura.

2. Funcionalidades da API (Envio e Listagem)

A autenticação é padronizada via `Bearer JWT`, que deve conter o `tenant_id` nas claims.

Envio de Mensagem

- `POST /v1/messages`
- **Entrada:** `conversaId`, `usuarioRemetenteId`, `direcao` (ex: `outbound`) e `conteudo` (JSON). O campo `criadoEm` é opcional.
- **Comportamento:** A API valida a entrada e publica o evento no Kafka. A chave da mensagem no Kafka é a `conversaId` para garantir a ordem.
- **Retorno:** `202T Accepted` com `{ mensagemId, conversaId }`.

Listagem de Mensagens

- `GET /v1/conversations/{conversaId}/messages?bucket=…&fromSeq=…&limit=…`
- **Saída:** Retorna uma lista paginada de mensagens, ordenada pela sequência, incluindo o status (ex: `delivered`).

- **Bucket:** O `bucket` é uma estratégia de particionamento temporal (mensal, `yyyymm`). Se não for informado na query, a API calcula o bucket atual.

  Nota sobre Design Futuro: O design já prevê idempotência de envio e lógica de reentrega (conforme tabelas `idempotencia_mensagem` e `entregas`), que podem ser ativadas em iterações futuras.

3. Processamento de Eventos (Kafka + Consumidor)

Tópico e Produtor (API)

- **Tópico:** `messages` (configurado com 6 partições em dev).
- **Chave:** `conversaId`. Isso garante que todas as mensagens de uma mesma conversa sejam processadas em ordem dentro da mesma partição.
- **Configuração do Produtor:** Usamos `acks=all` e `EnableIdempotence=true` para garantir a durabilidade e evitar duplicatas na publicação.

Consumidor (Router Worker)

- **Grupo:** `router-worker`.
- **Controle de Offset:** `EnableAutoCommit=false`. O commit é feito manualmente.
- **Lógica de Processamento:** Para cada evento consumido, o worker executa:
  1. Persistência da mensagem no Cassandra (calculando a `sequencia` da conversa/bucket).
  2. Atualização do status (ex: `sent` para `delivered`).
  3. Commit do offset no Kafka.

4. Decisão de Banco de Dados: Cassandra

Justificativa da Escolha O domínio da aplicação exige alto *throughput* de escrita (essencialmente *append-only*) e leituras rápidas por chave (a timeline de uma conversa). O Cassandra foi escolhido por suas vantagens nesse cenário:

- **Escala Horizontal:** O throughput cresce de forma previsível ao adicionar novos nós.
- **Ordenação Nativa:** A chave primária `PRIMARY KEY ((organizacao_id, conversa_id, bucket), sequencia)` garante que as mensagens sejam armazenadas e lidas em ordem natural por `sequencia` dentro de cada conversa/bucket.
- **Latência de Escrita:** As escritas são extremamente baratas (commit log + memtable), e as leituras por chave são otimizadas.
- **Séries Temporais:** O uso de TTL e estratégias de compactação (como TWCS) é ideal para o ciclo de vida dos dados de mensagens.

Trade-offs Assumidos Não há JOINs nem chaves estrangeiras. A modelagem é orientada à consulta (CQRS) e exige desnormalização. Para consultas analíticas complexas, os dados deverão ser projetados para um Data Warehouse.

Entidades Principais (Macro)

- `mensagens`: Armazena a timeline (sequencia, conteudo, status).
- `idempotencia_mensagem`: Tabela para LWT (`IF NOT EXISTS`) para evitar duplicatas de envio.
- `entregas`: Log de tentativas e reentregas por conector.

5. Resumo dos Endpoints

- `POST /v1/messages` — Envia uma nova mensagem (requer JWT).
- `GET /v1/conversations/{conversaId}/messages` — Lista mensagens de uma conversa (paginada).

6. Teste prático

Para conseguir o token de autenticação:

POST `/v1/auth/token` — emite um JWT para chamadas autenticadas.

Requisição (exemplo)

```
{
"organizacaoId": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa",
"usuarioId": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb",
"nome": "Alice",
"password": "dev"
}
```

Utilizando o EndPoint /v1/messages:

Usamos o seguinte body:

```
{
    "conversaId": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb",
    "usuarioRemetenteId": "cccccccc-cccc-cccc-cccc-cccccccccccc",
    "direcao": "outbound",
    "conteudo": { "tipo": "text", "texto": "alô, mundo!" }
}
```

Simulando uma mensagem do usuário “`cccccccc-cccc-cccc-cccc-cccccccccccc`” para o usuário “`bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb`“.

Tivemos o retorno com o code 202 - Accepted, com o seguinte body:

```
{
    "mensagemId": "c25cf0ca-f2c4-4a72-bc1a-80b4ee555702",
    "conversaId": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb"
}
```

No nosso worker que está esperando uma mensagem do tópico “messages”, executa a tarefa de persistir a mensagem:

```
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\wilso\source\repos\Whatslike\Chat.RouterWorker\bin\Debug\net8.0
info: Chat.RouterWorker.RouterWorkerService[0]
      Persistido conversa=bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb seq=1 offset=0
```

Após isso, se fizermos o GET no endpoint v1/conversations/bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb/messages?limit=50 , passando como parametro o usuário e o limite de mensagens para ser retornada temos:


```
{
    "items": [
        {
            "mensagemId": "c25cf0ca-f2c4-4a72-bc1a-80b4ee555702",
            "conversaId": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb",
            "sequencia": 1,
            "direcao": "outbound",
            "usuarioRemetenteId":
"cccccccc-cccc-cccc-cccc-cccccccccccc",
            "conteudo": {
                "tipo": "text",
                "texto": "alô, mundo!"
            },
            "status": "delivered",
            "criadoEm": "2025-11-06T05:05:11.937+00:00"
        }
    ],
    "bucket": 202511,
    "count": 1
}
```


Destaque para o status “`delivered`“

Como executar:

cd .\Chat.RouterWorker\
docker compose up -d --build

cd .\Chat.Api\
docker compose up -d --build

cd .\Chat.Api\

dotnet run --project .\Chat.RouterWorker\Chat.RouterWorker.csproj

dotnet run --project .\Chat.Api\Chat.Api.csproj

(Não conseguimos fazer o docker-compose geral a tempo)