# Documento Arquitetural - WebSocket Service (.NET 8) - Versão 2

## 1. Resumo Executivo

Este documento consolida a evolução da PoC do serviço WebSocket em .NET 8 (v2), agora com autenticação JWT, filtro por canal (subscriptions), integração opcional com Redis para escalabilidade horizontal, endpoints de observabilidade (/metrics, /deliveries, /clients, /acks) e rastreabilidade via TraceId/Elastic APM. Inclui payloads de exemplo, lista de arquivos criados/alterados, erros/riscos identificados e respectivas mitigações.

## 2. Contexto e Problema

O ambiente Python anterior é tratado como caixa-preta e pode estar apresentando indícios de instabilidade sob carga, ausência de métricas e imprevisibilidade operacional. A PoC v1 comprovou funcionalidade básica. A v2 amplia o escopo para segurança, observabilidade e prontidão para multi-instância.

## 3. Objetivos da PoC v2

- Autenticação com JWT no handshake do WebSocket e autorização por canal (claim subscriptions).

- Backplane Redis (Pub/Sub) para replicação de mensagens entre múltiplas instâncias.

- Exposição de métricas operacionais (/metrics) e trilhas de entrega (/deliveries, /acks, /clients).

- Rastreabilidade por mensagem (TraceId) compatível com Elastic APM.

- Automação de testes ponta a ponta com script próprio.

## 4. Arquitetura Lógica (v2)

Fluxo principal: Pub/Sub (HTTP /v1/publish) → Dispatcher → (Redis Pub/Sub opcional para fan-out) → WebSocket (/ws) → Clientes autorizados por canal. O exemplo de contrato JSON do publish foi mantido:

```

{

"info": {

"\_postman\_id": "1234567890",

"name": "WebSocket",

"schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json",

"\_exporter\_id": "987654321"

},

"item": [

{

"name": "publish",

"request": {

"method": "POST",

"header": [

{

"key": "accept",

"value": "application/json"

},

{

"key": "Content-Type",

"value": "application/json"

}

],

"body": {

"mode": "raw",

"raw": "{\n \"message\": {\n \"data\": \"IntxxxfSI=\",\n \"message\_id\": \"string\",\n \"publish\_time\": \"string\",\n \"body\": {}\n },\n \"subscription\": \"string\"\n}",

"options": {

"raw": {

"language": "json"

}

}

},

"url": {

"raw": "https://app.dev.api.com.br/app-platform-websocket-public-api/v1/publish",

"protocol": "https",

"host": [

"app",

"dev",

"api",

"com",

"br"

],

"path": [

"app-platform-websocket-public-api",

"v1",

"publish"

]

}

},

"response": []

}

]

}

````

## 5. Novos Arquivos e Modificações no Projeto

• Services/RedisService.cs: Gerencia conexão, subscribe/publicação em canal Redis (backplane).

• Services/DeliveryStore.cs: Persistência in-memory de entregas (deliveries) e ACKs.

• Services/JwtValidator.cs: Validação de token e construção do ClaimsPrincipal na conexão WS.

• Services/WebSocketDispatcher.cs (atualizado): Filtro por canal com base em claims; envio com registro de entrega e TraceId.

• Services/WebSocketConnectionManager.cs (atualizado): Armazena Socket + ClaimsPrincipal; expõe /clients.

• Models/PublishMessage.cs (atualizado): Inclui TraceId; mantém schema compatível ao contrato existente.

• Program.cs (atualizado): Mapeia endpoints /v1/publish, /metrics, /deliveries, /clients, /acks; injeta RedisService, Dispatcher, Validator.

## 6. Endpoints Disponíveis

|  |  |  |
| --- | --- | --- |
| **Método/Protocolo** | **Endpoint** | **Descrição** |
| POST (HTTP) | /v1/publish | Publica mensagem no canal indicado; respeita filtro por subscriptions. |
| WS | /ws?token=JWT | Conecta cliente autenticado; token define canais permitidos. |
| GET | /metrics | Métricas operacionais em JSON (conexões ativas, mensagens, publishRequests, timestamp). |
| GET | /deliveries | Lista de entregas realizadas (connectionId, messageId, deliveredAtUtc). |
| GET | /acks | Confirmações de recebimento enviadas pelo cliente ({"ack":"<message\_id>"}). |
| GET | /clients | Clientes conectados e claims associadas (auditoria). |
| GET | /swagger | Interface interativa para testar /v1/publish e consultas. |

## 7. Contrato de Publicação e Payloads de Exemplo

Estrutura obrigatória do corpo do POST /v1/publish:

{ "message": { "data": "IntxxxfSI=", "message\_id": "string", "publish\_time": "string", "body": {} }, "subscription": "string" }

Exemplos completos:

- Canal chat:

{  
 "message": {  
 "data": "Q2hhbm5lbCBjaGF0",  
 "message\_id": "msg-chat-001",  
 "publish\_time": "2025-10-26T12:00:00Z",  
 "body": { "event": "chat\_message", "text": "Mensagem do canal chat" }  
 },  
 "subscription": "chat"  
}

- Canal user-events:

{  
 "message": {  
 "data": "dXNlciBsb2dpbg==",  
 "message\_id": "msg-user-001",  
 "publish\_time": "2025-10-26T12:02:00Z",  
 "body": { "event": "user\_login", "message": "Mensagem do canal user-events" }  
 },  
 "subscription": "user-events"  
}

- Mensagem com ACK (chat):

{  
 "message": {  
 "data": "YWNrbWVzc2FnZQ==",  
 "message\_id": "msg-ack-01",  
 "publish\_time": "2025-10-26T12:15:00Z",  
 "body": { "event": "requires\_ack", "description": "Mensagem que exige confirmação" }  
 },  
 "subscription": "chat"  
}

Formato de ACK enviado pelo cliente WebSocket (texto):

{"ack":"<message\_id>"}

## 8. Segurança e Autorização por Canal

Os clientes WS informam um JWT na querystring (?token=). O servidor extrai claims e filtra a entrega de acordo com a claim 'subscriptions'. São suportados dois formatos: (i) múltiplas claims 'subscriptions'; (ii) claim única 'subscriptions\_json' contendo um array em string.

Exemplos de payload JWT (claims relevantes):

{ "sub":"user-123", "subscriptions": ["chat"] }

{ "sub":"user-456", "subscriptions\_json":"[\"user-events\"]" }

## 9. Observabilidade e Rastreabilidade (TraceId/APM)

Cada publicação recebe/propaga um TraceId (se ausente, o servidor gera). Logs são estruturados e compatíveis com coleta por ELK/APM. O Dispatcher inclui spans do tipo WS-Broadcast, possibilitando rastrear o caminho HTTP → Redis → WebSocket por mensagem.

## 10. Resultados de Teste (v2)

Teste automatizado por script validou:

(i) publicação segmentada por canal; (

ii) recebimento distinto por clientes; (

iii) métricas atualizadas;

(iv) registro de entregas e ACK;

(v) geração de TraceId;

(vi) stress leve com 5 mensagens em sequência sem erros.

Exemplos de respostas típicas:

POST /v1/publish → {"status":"sent","totalMessages":N,"activeConnections":M}

GET /metrics → {"activeConnections":2,"messagesBroadcasted":N,"publishRequests":K,"timestamp":"...Z"}

GET /deliveries → [ { "connectionId":"...", "messageId":"...", "deliveredAtUtc":"...Z" }, ... ]

## 11. Erros/Riscos identificados e Mitigações (consolidado)

|  |  |  |  |
| --- | --- | --- | --- |
| **Etapa / Situação** | **Erro observado** | **Causa raiz** | **Mitigação** |
| Build inicial | CS1061 AddSwaggerGen ausente | Pacote Swashbuckle não instalado | Instalar Swashbuckle.AspNetCore |
| Swagger + OpenAPI | CS1002 ; expected | Sintaxe ao usar WithOpenApi() e refs | Corrigir imports/usar Annotations |
| Execução | Porta incorreta (5000 vs 5087) | ASPNETCORE\_URLS divergente | Definir URL explícita ao rodar |
| JWT | 401/assinatura inválida | Secret/Issuer/Audience divergentes | Gerar token com mesmo segredo e claims |
| Redis | Falha de conexão | Redis inativo/local ausente | Subir redis-server ou docker redis e retry |
| Projeto | Conflito csproj/bin/obj | Script auxiliar gerou artefatos conflitantes | Limpeza de bin/obj e ajuste csproj |
| Autorização | Todos recebendo tudo | Tokens sem 'subscriptions' | Gerar tokens com claims de canal ou forçar bloqueio |

## 12. Plano de Evolução

- Integrar com Identity Provider corporativo (Keycloak/Auth0/Cognito/AAD) para emissão/validação de JWT.

- Containerizar API e Redis; publicar imagens e orquestrar (Kubernetes ou similar).

- Ativar Elastic APM/ELK em staging; criar dashboards para latência/broadcast/erros.

- Executar testes de carga com 100–1000 conexões simultâneas e medir latência ponta a ponta.

- Configurar CI/CD (build, test, scan, docker push, deploy) e política de rollback.

## 13. Conclusões e Recomendações

A PoC v2 cumpre os objetivos de segurança, segmentação por canal e observabilidade, mantendo o contrato JSON original. Com Redis habilitado, o serviço está apto a operar em múltiplas instâncias. Recomenda-se avançar para staging com autenticação federada, testes de carga e monitoramento contínuo antes de promover para produção.

Autor: Wanderson Ferreira da Silva

Cargo: Arquiteto

Data: 24/10/2025