# Documento Arquitetural - WebSocket Service (.NET 8) - Versão 3

## 1. Resumo Executivo

A versão v3 consolida os avanços da PoC WebSocket em .NET 8, introduzindo entrega UNICAST por usuário autenticado e autenticação JWT via Header Authorization, mantendo compatibilidade com ?token= para retrocompatibilidade. O objetivo principal foi garantir isolamento de mensagens por usuário, preservar o contrato JSON legado e preparar a base técnica para evolução ao MVP com segurança, rastreabilidade e escalabilidade.

## 2. Contexto e Problema

Na versão anterior (v2), o sistema operava apenas em modo broadcast, enviando mensagens para todos os clientes conectados. Essa abordagem não atendia aos requisitos de isolamento de mensagens e autenticação padronizada. A v3 foi desenvolvida para permitir comunicação unicast (direta e exclusiva para cada usuário autenticado), utilizando o JWT como fonte confiável de identidade e garantindo total compatibilidade com o ambiente legado em produção.

## 3. Objetivos da PoC v3

- Implementar entrega UNICAST com base no campo target\_user\_id.  
- Priorizar autenticação JWT via Header Authorization: Bearer <token>, mantendo suporte ao ?token=.  
- Permitir múltiplas conexões por usuário.  
- Garantir compatibilidade total com o contrato JSON original.  
- Fortalecer rastreabilidade (TraceId, Elastic APM, logs estruturados).  
- Consolidar observabilidade operacional com endpoints de telemetria e métricas.

## 4. Arquitetura Lógica (v3)

Fluxo principal: HTTP /v1/publish → Dispatcher → Redis (opcional, backplane) → WebSocket /ws → Conexões autenticadas (vinculadas à claim 'sub' do JWT).  
  
Contrato JSON:  
{  
 "message": {  
 "data": "IntxxxfSI=",  
 "message\_id": "string",  
 "publish\_time": "string",  
 "body": {}  
 },  
 "subscription": "string"  
}

## 5. Principais Modificações e Arquivos Atualizados

• Program.cs: Autenticação via Header Authorization e fallback por query.  
• WebSocketConnectionManager.cs: Novo índice userId → connectionId; múltiplas conexões por usuário.  
• WebSocketDispatcher.cs: Novo método SendToUserAsync; fan-out intrausuário.  
• JwtValidator.cs: Validação prioritária via header.  
• PublishMessage.cs: Campo TargetUserId adicionado.  
• appsettings.json: Mantida estrutura JWT e Redis.

## 6. Endpoints Disponíveis

POST /v1/publish - Publica mensagem (unicast se TargetUserId presente, senão broadcast).  
WS /ws - Conecta cliente autenticado via Header Authorization ou ?token=.  
GET /metrics - Exibe métricas operacionais.  
GET /deliveries - Lista entregas realizadas.  
GET /acks - Exibe confirmações recebidas.  
GET /clients - Lista clientes ativos e claims.  
GET /swagger - Interface Swagger para testes.

## 7. Contrato e Payloads de Exemplo

{  
 "message": {  
 "data": "VXNlcg==",  
 "message\_id": "msg-uni-005",  
 "publish\_time": "2025-10-27T23:15:00Z",  
 "body": { "event": "private\_notice", "text": "Somente A deve receber (teste final)" }  
 },  
 "subscription": "chat",  
 "target\_user\_id": "a123"  
}

## 8. Autenticação e Segurança JWT

O JWT é validado primeiro no Header Authorization e, caso ausente, no parâmetro ?token=. A claim 'sub' representa o ID do usuário e é usada para mapeamento das conexões. O serviço suporta múltiplas conexões por usuário (ex: duas abas abertas). Tokens inválidos ou expirados retornam 401 Unauthorized.

## 9. Observabilidade e Rastreabilidade

Cada mensagem inclui TraceId, MessageId e TargetUserId. Elastic APM integra-se automaticamente ao pipeline HTTP e WebSocket. Endpoints /metrics, /deliveries e /clients fornecem visibilidade instantânea das atividades. Logs em formato JSON permitem integração com ELK, Grafana e Loki.

## 10. Resultados dos Testes

Unicast → Mensagem entregue apenas ao usuário A.  
Broadcast → Entrega a todos conectados.  
JWT Header → Autenticação correta.  
Fallback Query → Compatibilidade garantida.  
Logs e APM → TraceId correlacionado.

## 11. Riscos e Mitigações

Autenticação: Token inválido - mitigado com ClockSkew 30s.  
Concorrência: múltiplas conexões por usuário - mitigado com ConcurrentDictionary.  
Mensagens perdidas: mitigado via heartbeats (MVP).  
Escalabilidade: resolvido via Redis Backplane opcional.

## 12. Plano de Evolução (MVP)

- Adicionar heartbeats e reconexão automática.  
- Docker Compose integrando API, Redis e Elastic Stack.  
- Health Check Endpoint para readiness/liveness.  
- Integração CI/CD via GitHub Actions.  
- Testes de carga com 1000+ conexões.  
- Integração federada JWT (Keycloak/Auth0/Cognito).

## 13. Conclusão

A versão v3 consolida a PoC em nível funcional pronto para produção: entrega unicast isolada, JWT via Header Authorization como padrão, compatibilidade total com o contrato legado, rastreabilidade ponta a ponta e base sólida para o MVP distribuído e observável.  
  
Autor: Wanderson Ferreira da Silva  
Cargo: Arquiteto  
Data: 27/10/2025