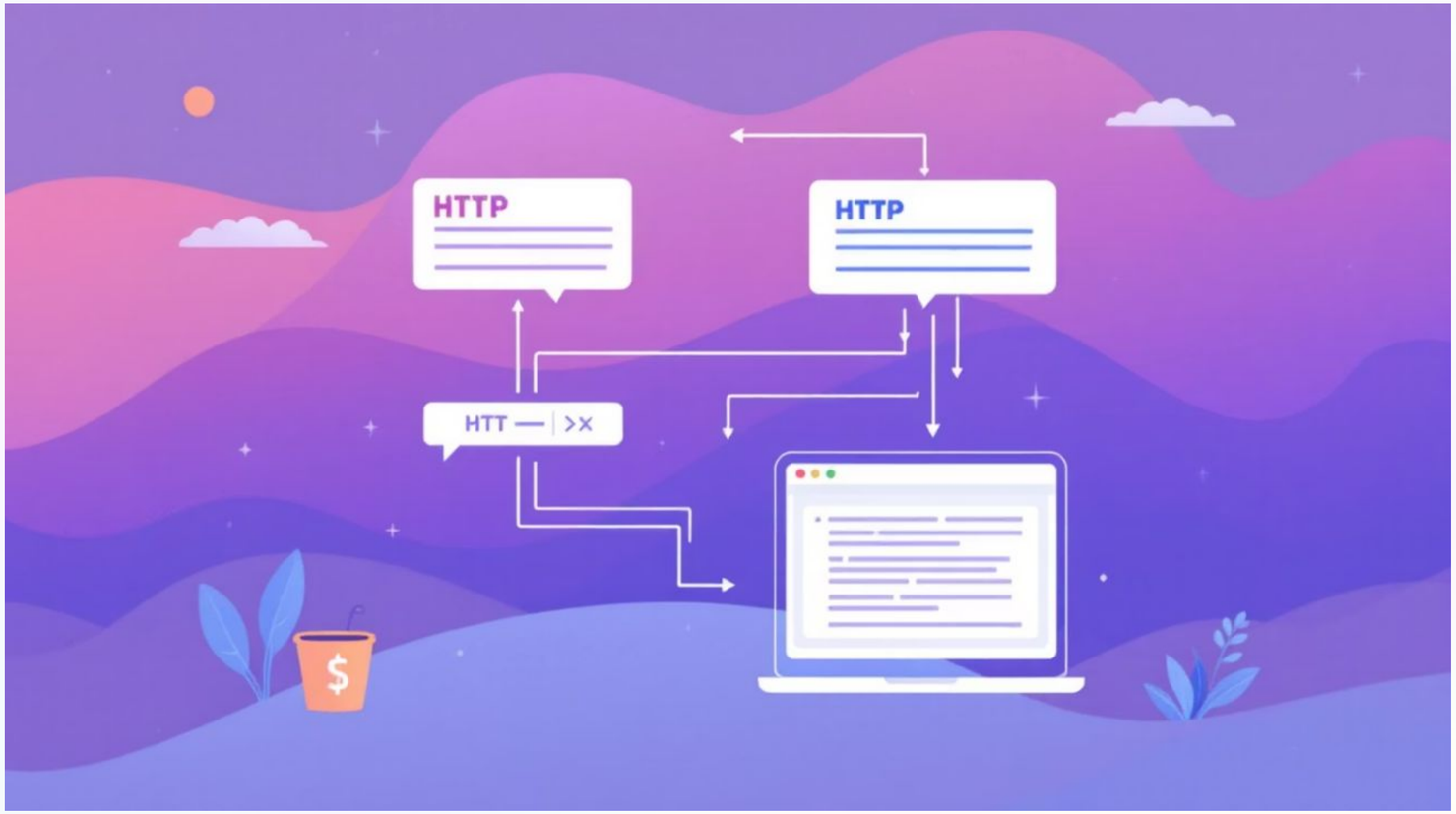
Chatterbox: A Real-Time WebSocket Chat Application

Building a highly concurrent, bi-directional chat system that scales to thousands of users with persistent connections and instant message delivery.

The Challenge of Real-Time Communication



Traditional Request-Response Limitations

Conventional HTTP architectures establish a new connection for each request, creating latency and preventing persistent, bi-directional communication needed for real-time applications.

- Each request requires TCP handshake
- Server cannot push updates to clients
- High overhead for frequent updates
- Firewall and NAT traversal complications

Enter WebSockets: Persistent Bi-Directional Communication

Persistent Connection

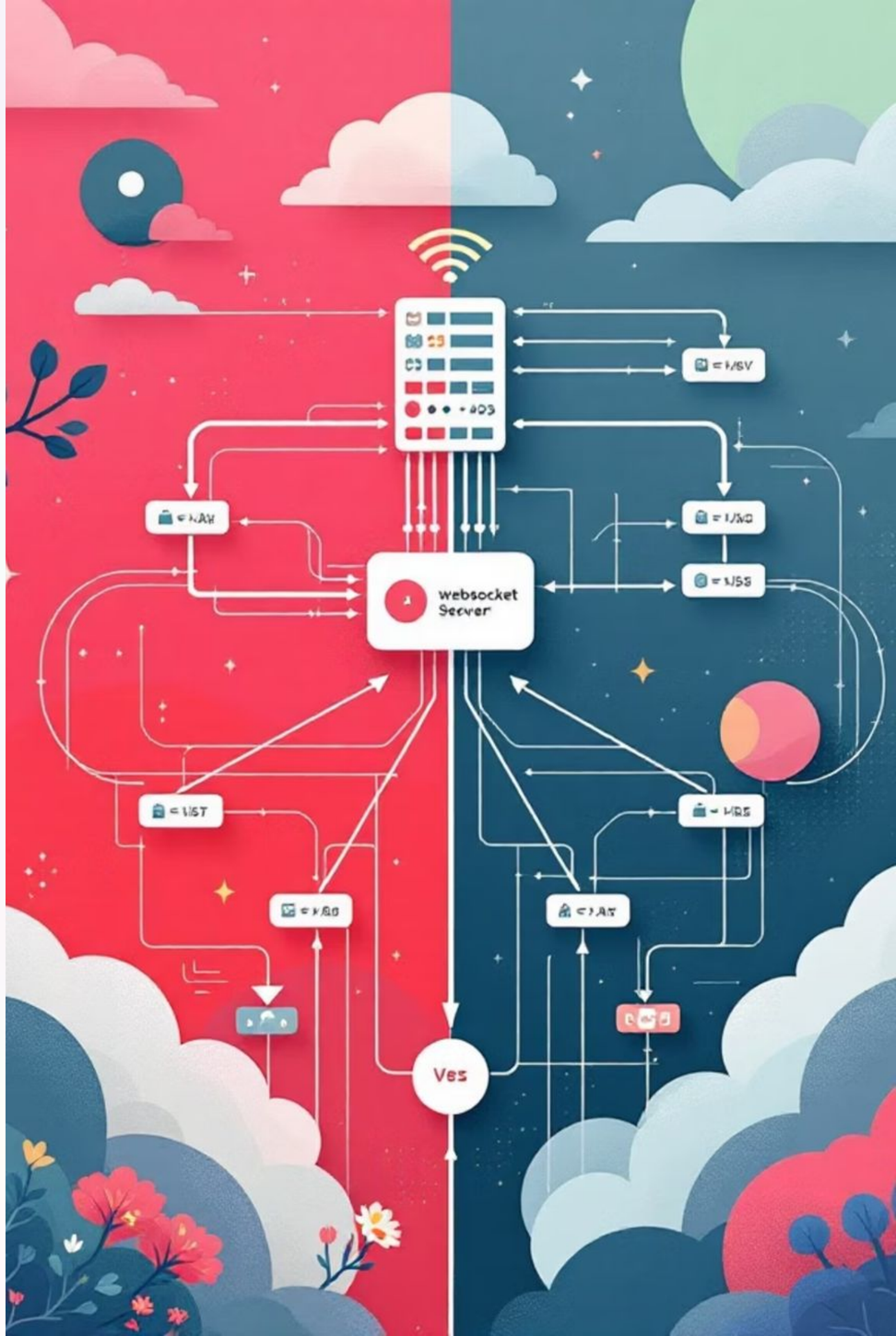
Single TCP connection stays open, eliminating repeated handshakes and reducing latency significantly.

Bi-Directional Flow

Both server and client can send messages independently without waiting for requests from the other end.

Low Overhead

Minimal framing overhead compared to HTTP headers, allowing efficient, high-frequency message transmission.



Technology Stack

FastAPI Backend

Async Python framework for building high-performance WebSocket endpoints with automatic API documentation and type hints.

WebSocket Protocol

Persistent connection handling and pub-sub messaging patterns for real-time broadcasting to all connected clients.

PostgreSQL

Relational database storing user accounts, authentication credentials, and complete chat history with ACID compliance.

Terminal Client

Interactive command-line interface written in Python for connecting, sending messages, and receiving real-time broadcasts.

System Architecture Overview

Authentication Layer

Handles user registration, login, and session management.

Validates credentials and creates secure sessions.

WebSocket Manager

Central hub managing all connections, tracking active users, and broadcasting messages to subscribers.

Data Storage

Persists chat history, user profiles, and metadata for retrieval and audit purposes.

WebSocket Manager: The Heart of the System

Core Responsibilities

The manager orchestrates the entire real-time experience, efficiently routing messages and maintaining connection state across thousands of concurrent users.

Connection Management

Tracks all active WebSocket connections, handles new connections, and gracefully closes stale sessions.

Message Broadcasting

Receives messages from any client and instantly relays them to all connected participants in the chat room.

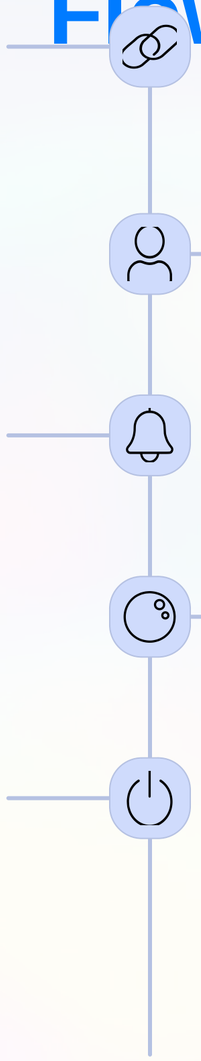
Pub-Sub Pattern

Implements publish-subscribe model where clients subscribe to channels and publisher broadcasts to relevant subscribers.

Client-Server Interaction Flow

Connection

Client establishes WebSocket connection to server endpoint, transmitting authentication credentials via headers or initial message.



Authentication

Server validates user credentials, creates session record, and returns success response or error message.

Subscription

Client subscribes to chat room channels it wants to receive messages from, joining the broadcast group.

Chat

Messages typed by user are sent to server, which logs to database and broadcasts to all subscribers instantly.

Disconnection

When client closes connection, server removes session, unregisters from broadcast groups, and marks session as expired.

Key Technical Features

Asynchronous Processing

Non-blocking I/O using asyncio enables handling thousands of concurrent connections on a single server instance through cooperative multitasking.

Secure Authentication

JWT tokens for session management, password hashing with bcrypt, HTTPS/TLS encryption for all WebSocket connections.

Message Persistence

Complete chat history stored in database with timestamps, user references, and message content for retrieval and auditing.

Input Validation

Sanitises all incoming messages, validates data structure, and prevents injection attacks through strict type checking.

Rate Limiting

Prevents abuse by limiting message frequency per user, protecting against denial-of-service attacks and spam.

Multi-Room Support

Users can join multiple chat rooms simultaneously, each with independent access controls and message broadcasting.

Conclusion & Future Enhancements

Achievements

Scalable WebSocket implementation

Handles thousands of concurrent connections efficiently using async architecture

Real broadcast capability

Messages delivered to all participants within milliseconds

Production-ready features

Authentication, persistence, security, and input validation

Minimal client overhead

Lightweight terminal interface for testing and validation

Future Roadmap

Message History API

Retrieve past conversations with pagination and search functionality

User Presence Tracking

Show online/offline status and last seen timestamps for contact list

Private Messaging

One-to-one encrypted conversations outside public chat rooms

Horizontal Scaling

Distributed deployment across multiple servers with message broker integration