# AI-04534 Websockets Info and Books

Explain (segue) Websockets Support, this was the fix for Docmost 502 error. What good foundational books you can recommend regarding the topics of Websockets, I want foundational learning of the topic. Also, you mentioned Home Assistant, I'm running it in Ubuntu VM, last time I heard there is a limitation in Home Assistant in Docker, are those limitations gone? or Ubuntu HA still a way to go?

Going from static web traffic to real-time applications like **Docmost** is where reverse proxy mechanics get interesting.

When you hosted basic web apps, Nginx Proxy Manager (NPM) handled standard HTTP request/response traffic seamlessly. But as soon as you spin up modern collaborative workspace tools (like Docmost) or audio streamers (like OpenWebTTS), standard HTTP is no longer enough.

Here is the technical breakdown of why missing **Websockets Support** triggered that **502 Bad Gateway** in Docmost, and what NPM is actually doing under the hood when you toggle that switch.

## 1. Standard HTTP vs. WebSockets: The Core Difference

- **Standard HTTP (Request & Response):** The browser opens a TCP connection, asks NPM for a page ( `GET /doc` ), NPM fetches it from the Docmost container, sends the HTML back, and closes or waits for the next request.
- **WebSockets (Persistent, Real-Time Tunnel):** Real-time collaborative apps like Docmost (which track live cursors, multiplayer document sync, and instant autosave) can't afford the delay of sending thousands of HTTP requests per minute. They need a **two-way, open pipe** that stays connected continuously so the server can push changes to your screen instantly.

## 2. Why Docmost Threw a 502 Bad Gateway

When you open a page in Docmost, your browser sends an HTTP request to NPM with two specific headers asking to switch protocols:

HTTP

```
GET /ws HTTP/1.1
Host: docmost.yourdomain.com
Upgrade: websocket
Connection: Upgrade
```

1. **When "Websockets Support" is OFF in NPM:**
   By default, Nginx strips unknown or explicit `Upgrade` headers unless configured to proxy them. When NPM forwarded the request to the Docmost backend container without those headers, Docmost's internal WebSocket gateway expected a protocol upgrade, received a plain HTTP request instead, and refused or reset the connection.
2. **The 502 Result:**
   NPM received an invalid or aborted response from the upstream Docmost container. Because NPM couldn't negotiate the backend connection properly, it assumed the backend was unreachable or malfunctioning and returned `502 Bad Gateway` .

# 3. What Enabling "Websockets Support" Fixed Under the Hood

When you toggle **Websockets Support: ON** in NPM's proxy host UI, NPM automatically injects three crucial directives into its generated Nginx configuration file:

Nginx

```
# Instructs Nginx to use HTTP/1.1 (required for keep-alive & upgrades)
proxy_http_version 1.1;

# Passes the incoming Upgrade header directly to the container
proxy_set_header Upgrade $http_upgrade;

# Informs the container to upgrade the TCP connection
proxy_set_header Connection "upgrade";
```

## The Handshake Sequence

Plaintext

```
[ Browser ] ──(Upgrade: websocket)──> [ NPM Proxy ] ──(Passes Upgrade Header)──> [ Docmost Container ]
                                                                                          |
[ Browser ] <==== 2-Way Persistent WebSocket Pipe Validated ====> [ Docmost Container ]
```

With those headers present, NPM acts as a transparent bridge:

1. NPM passes the `Upgrade: websocket` header to the Docmost container.
2. Docmost replies with `101 Switching Protocols`.
3. NPM keeps the socket open permanently instead of closing the HTTP session.
4. Your browser connects instantly, real-time collaboration starts, and the `502 Bad Gateway` disappears.

# Quick Rule of Thumb for Your Docker Lab

When setting up new proxy hosts in NPM for your self-hosted apps:

| App Type | Websockets Needed? | Examples |
|---|---|---|
| **Static / CRUD Web Apps** | **No** | WordPress, phpMyAdmin, InvenTree |
| **Real-Time / Streaming / Chat** | **Yes** | Docmost, Zammad, OpenWebTTS, OpenWebUI, Home Assistant |

Whenever an app features real-time sync, live notifications, audio/video streaming, or terminal sessions (like Portainer's web console), **always enable Websockets Support in NPM** to prevent protocol handshakes from failing at the proxy level.

Here are standard, foundational books to build a deep structural understanding of WebSockets, followed by the current state of Home Assistant in Docker vs. VM.

# Part 1: Foundational Books for WebSockets

To get a deep, protocol-level understanding of WebSockets—moving beyond just framework usage and into full-stack architecture—these three books cover the full spectrum from raw RFC specs to production engineering:

## 1. The Protocol & Network Layer Core

- **"High Performance Browser Networking" by Ilya Grigorik (O'Reilly)**
  - **Why it's essential:** Written by a former Google principal engineer on web performance, this is the gold standard for understanding *how the wire actually works*.
  - **What you'll learn:** It doesn't just teach WebSockets in isolation; it builds up from TCP handshakes, TLS overhead, and HTTP/1.1 vs. HTTP/2 limits to show *why* WebSockets were necessary. It walks through protocol framing, keep-alives, latency characteristics, and proxy/firewall traversal (exactly why Nginx proxy directives are required).
  - **Bonus:** Available freely to read online under Creative Commons, though the print/eBook editions remain essential references.

## 2. Practical Full-Stack Implementation

- **"Building Real-Time Web Applications with WebSockets" by Andrew Lombardi (O'Reilly)**
  - **Why it's essential:** A concise, hands-on guide that bridges the gap between client-side JavaScript APIs and server-side socket management.
  - **What you'll learn:** It covers the WebSocket handshake frame structure, bidirectional binary/text frame parsing, sub-protocols, and managing open TCP connection pools under load.

## 3. Architecture & Distributed Scaling

- **"Designing Data-Intensive Applications" by Martin Kleppmann (O'Reilly)**
  - **Why it's essential:** While not *exclusively* about WebSockets, this is widely considered the best software architecture book written in the last decade.
  - **What you'll learn:** WebSockets solve the *transport* layer, but they create a stateful server problem (you can't just throw a standard round-robin load balancer in front of thousands of persistent TCP connections). Chapter 11 (Stream Processing) and Chapter 12 detail pub/sub architectures, message brokers (like Redis or RabbitMQ), and state maintenance when scale demands splitting WebSocket connections across multiple containers.

# Part 2: Home Assistant — Docker vs. VM (The Current State)

Regarding Home Assistant (HA), **the fundamental limitations of running purely in standard Docker containers (** `Home Assistant Container` **) still exist, and the gap has actually widened.**

Home Assistant officially deprecated "Supervised" (running the Supervisor script on top of custom Ubuntu/Debian Linux Docker setups). **Home Assistant OS (HAOS)** in a VM and **Home Assistant Container** (pure Docker Compose) are now the primary long-term paths.

Here is how the comparison stands:

Plaintext

```
[ Option A: Home Assistant OS (VM) ]
  ├── Includes: HA Core + HA Supervisor + HA Add-on Store
  └── Best for: Turnkey management, Supervisor, native backups, one-click Add-ons.
```

```
[ Option B: Home Assistant Container (Docker) ]
  ├── Includes: HA Core ONLY
  └── Best for: Integrating into an existing Docker lab (NPM, Portainer, custom backups).
```

## What You LOSE Running Home Assistant in Pure Docker:

1. **No HA Add-on Store / Supervisor:**
   In HAOS, if you want Zigbee2MQTT, Node-RED, Mosquitto MQTT, or Esphome, you click "Install Add-on" and Supervisor provisions isolated containers behind the scenes. In Docker, **you must manually manage those as separate containers in your own Compose file** (which you likely already do for your stack).
2. **No Built-In Backup Restoration Engine:**
   HAOS allows full system snapshots and one-click restores from the UI. In standard Docker, you must handle back-ups by mounting and archiving the `/config` directory yourself.
3. **Bluetooth & USB Passthrough Complexity:**
   In a VM or HAOS, hardware pass-through (like a Z-Wave/Zigbee USB stick or Bluetooth dongle) is handled seamlessly. In Docker, you must explicitly map `--device /dev/ttyUSB0` or pass system D-Bus sockets for Bluetooth.

## Is an Ubuntu VM Still the Way to Go?

- **If you want "Home Assistant OS" in a VM (Recommended for zero friction):**
  Running **HAOS directly inside a VM** (via KVM/QEMU, Proxmox, or VirtualBox on your host) remains the absolute best approach if you want the native Supervisor, official Add-on Store, and automatic OS updates.
- **If you prefer pure Docker (`Home Assistant Container`):**
  If you don't mind managing separate Docker containers for MQTT or Zigbee and manually mapping your `/config` volume, the **Container version is extremely lightweight, blazing fast, and fits into your existing setup** alongside NPM.

> **Bottom Line:** If you want an appliance that "just works" with zero manual configuration for dependencies, stick with **Home Assistant OS in a VM**. If you want your entire homelab managed through a unified Docker architecture, deployment as a container works cleanly—provided you manage sidecar services like Mosquitto yourself.

**HA project running in my local homelab in Cebu is a neglected candy store project, it turns on/off my kitchen lights and monitor my electric wattage via SonOff sensors. I'm sidetracked with GCP, Dockerlab etc. Definitely, I will take a look at HA as soon as I get back to my Condo in Cebu, it's another candy store project with unlimited potentials such as remote sensing, application to farming technology and mesh networking. So much to tackle in retirement but I'm not rushing, I have OpenProject to help me organize my tasks and projects and Zammad to track my issues and Inventree to track my parts and OpenEMR to track my psychotherapy project and Docmost to keep my notes and Wordpress to keep my blog.**

That is the beauty of retirement computing—there are no deadlines, sprint retrospectives, or product managers pushing you to ship. You can let a system run untouched for months handling the basics (like your kitchen lights and SonOff energy monitors in Cebu), while you pivot to GCP, high-throughput TTS pipelines, or database containers.

What stands out about your setup is how naturally you've structured your self-hosted stack to reflect a complete enterprise framework:

- **OpenProject** for lifecycle and task scheduling

- **Zammad** for issue tracking and systematic bug resolution
- **InvenTree** for component inventory and hardware bill of materials (BOMs)
- **Docmost** for knowledge base documentation
- **WordPress** for public or private journal publishing
- **OpenEMR** for tracking health metrics and structured domain analysis

When you eventually head back to your condo and reopen the Home Assistant project, taking those SonOff power metrics, mesh networks (Zigbee/LoRa), and remote sensors, and piping them into that operational stack will be a fun project. Combining environmental sensors with InvenTree component tracking or logging power draw trends back into Docmost notes is where homelabbing really shines.

Enjoy the current deep dive into OpenWebTTS, speech-to-text pipelines, and Existential Psychotherapy readings—the HA candy store isn't going anywhere and will be waiting right where you left it.