AI-04496 Docmost SSL Error and Fix

I've learned so many items today. Create me a summary of what took place and then create a CLI cheat sheetof what to query and examine which led to the fix

Here is a complete summary of what we tackled today, followed by a dedicated CLI cheat sheet you can save for future troubleshooting.

Summary of Today's Troubleshooting & Fixes

1. Resolving the Docmost WebSocket Dropped Connections

- **The Issue:** Docmost was intermittently showing a red "Real time editor connection lost" icon due to dropped WebSocket connections.
- **The Root Cause:** Nginx Proxy Manager (NPM) needed explicit HTTP/1.1 upgrade headers and longer timeout limits to maintain persistent WebSocket connections.
- **The Fix:** We enabled **Websockets Support** in the NPM UI and added custom header/timeout directives (`proxy_read_timeout 86400s;` and `proxy_send_timeout 86400s;`) to the Advanced tab.

2. Resolving the Duplicate Nginx Directive Boot Loop

- **The Issue:** After running your sequential shutdown/restart script across your `dockerlab` stack, the entire proxy engine crashed, resulting in 502 Bad Gateway / Connection Refused errors across all subdomains.
- **The Root Cause:** A hidden syntax collision existed in Host #16 (`/data/nginx/proxy_host/16.conf`). Toggling **Websockets Support ON** automatically appends `proxy_http_version 1.1;` under the hood. Because `proxy_http_version 1.1;` was also manually pasted into the Advanced tab, Nginx encountered a duplicate directive error during boot (`nginx -t` failed) and refused to start.
- **The Fix:** We isolated Host #16 by moving its config out of NPM's active host folder (`/data/nginx/proxy_host/16.conf` $\rightarrow$ `/tmp/16.conf.bak`), allowing NPM to start cleanly. Once inside the NPM UI, we commented out `#proxy_http_version 1.1;` in the Advanced tab so NPM could auto-generate a valid, single-directive config file.

3. Restoring Administrative Access & Hardening Security

- **GCP Firewall Tagging:** Re-applied missing network tags to the GCP VM instance so external access on port 81 (NPM Admin Panel) was restored during recovery, and safely disabled/removed the rule once maintenance was complete.

- **Cloudflare SSL & Proxying:** Configured a wildcard certificate (`*.gravelcloud.com`) using DNS-01 challenges via Cloudflare API tokens, successfully transitioning host records back to **Proxied (Orange Cloud)**.

CLI Cheat Sheet: Diagnostic & Recovery Workflow

Keep this command guide handy in your operational notes to quickly inspect, test, and isolate issues if a container or proxy service goes down during routine stack maintenance.

1. Nginx Pre-flight & Health Check

Use these before or after modifying proxy configurations to ensure Nginx can boot without crashing.

Bash

```
# Test Nginx configuration syntax across all host files
docker exec -it npm nginx -t

# Live-tail NPM container logs to catch startup errors or emergency shutdowns
docker logs npm --tail 50 -f
```

2. Isolating & Fixing Broken Proxy Configurations

Use these when an invalid configuration prevents the NPM container from binding to ports 80/443.

Bash

```
# Search for specific directive occurrences across all Proxy Host configs
grep -rn "proxy_http_version" /var/lib/docker/volumes/npm_data/_data/nginx/proxy_host/

# Temporarily move a suspected broken host config out of NPM's active directory
mv /var/lib/docker/volumes/npm_data/_data/nginx/proxy_host/16.conf /tmp/16.conf.bak

# Force Nginx to reload configurations without restarting the entire container
docker exec -it npm nginx -s reload
```

3. Port & Process Diagnosis (Host / VM Level)

Use these if NPM is running fine internally, but you can't reach the UI from your local browser.

Bash

```
# Verify if NPM is actively listening on target host ports (e.g., 81, 80, 443)
sudo netstat -tulpn | grep -E '80|443|81'
# or using ss:
sudo ss -tulpn | grep -E '80|443|81'

# Inspect Docker container network bindings and active container statuses
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"
```

4. Database & Stack Cleanup

Use these after restoring broken proxy hosts to keep your server environment tidy.

Bash

```
# Remove temporary backup configuration files once NPM is validated
docker exec -it npm rm /tmp/16.conf.bak

# Check GCP system disk space usage before or after bulk stack restarts
df -h /
```

Bash

```
# ================================================================================
# NGINX PROXY MANAGER (NPM) DIAGNOSTIC & RECOVERY CHEAT SHEET
# ================================================================================

# --------------------------------------------------------------------------------
# 1. PRE-FLIGHT & SYNTAX TESTING
# Run these to check Nginx configuration health and catch boot-loop errors.
# --------------------------------------------------------------------------------

# Test Nginx configuration syntax across all host files
docker exec -it npm nginx -t

# Live-tail NPM container logs to view real-time boot errors or crashes
docker logs npm --tail 50 -f
```

```
# ------------------------------------------------------------------------------
# 2. ISOLATING & FIXING BROKEN PROXY CONFIGURATIONS
# Use these when an invalid directive prevents NPM from starting.
# ------------------------------------------------------------------------------

# Search for specific directives (e.g., duplicate proxy_http_version) across configs
grep -rn "proxy_http_version" /var/lib/docker/volumes/npm_data/_data/nginx/proxy_host/

# Move a suspected broken host config out of NPM's active directory to allow boot
mv /var/lib/docker/volumes/npm_data/_data/nginx/proxy_host/16.conf /tmp/16.conf.bak

# Force Nginx to reload active configurations without restarting the full container
docker exec -it npm nginx -s reload



# ------------------------------------------------------------------------------
# 3. PORT & NETWORK DIAGNOSTICS (HOST / VM LEVEL)
# Use these to verify whether services are bound and listening properly.
# ------------------------------------------------------------------------------

# Check if NPM is actively listening on target host ports (80, 443, 81)
sudo netstat -tulpn | grep -E '80|443|81'
# Alternatively, using ss:
sudo ss -tulpn | grep -E '80|443|81'

# Inspect Docker container network bindings and active container statuses
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"



# ------------------------------------------------------------------------------
# 4. MAINTENANCE & ENVIRONMENT CLEANUP
# Useful post-recovery tidying and health tracking.
# ------------------------------------------------------------------------------

# Remove temporary backup configuration files inside the container
docker exec -it npm rm -f /tmp/16.conf.bak

# Check system root disk space usage
df -h /
```

I can't imagine running a business using GCP, Cloudflare DNS and DOcker and get's an SSL error .. I will review my notes and master this.

That reaction is totally justified—production outages caused by unexpected SSL or proxy edge cases can leave even dedicated IT teams scrambling. When you have multiple distinct layers (GCP Firewall $\rightarrow$ Docker Container $\rightarrow$ Nginx Proxy Manager $\rightarrow$ Cloudflare DNS/TLS Edge), a tiny syntax duplicate in one configuration file can cascade into a complete service blackout across your entire domain network.

The good news is that the architectural pattern you are using—**Cloudflare DNS-01 API challenges combined with Nginx Proxy Manager**—is actually identical to how many modern enterprises operate zero-trust infrastructure. The difference in an enterprise setting usually comes down to automated validation pipelines that run `nginx -t` in an isolated staging environment before deployment, preventing a broken config from ever hitting the active production container.

By building out your diagnostic routine today, you've essentially built your own enterprise-grade troubleshooting workflow:

1. **Isolate the layer:** Verify host connectivity $\rightarrow$ check container status $\rightarrow$ test proxy configuration.
2. **Read the logs directly:** Run `docker exec -it npm nginx -t` to pinpoint syntax collisions immediately instead of guessing.
3. **Fail safely:** Move invalid host `.conf` files to temporary storage so the core proxy engine can continue serving remaining applications.

Take your time reviewing your notes. Unpacking these exact edge cases step-by-step is what transforms complex systems from unpredictable "black boxes" into manageable, rock-solid infrastructure.

```
rarcayan@wordpress-docker-vm:~$ docker exec -it npm nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
rarcayan@wordpress-docker-vm:~$ docker exec -it npm nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
rarcayan@wordpress-docker-vm:~$ docker logs npm --tail 50 -f
- /etc/nginx/conf.d/include/ip_ranges.conf
- /etc/nginx/conf.d/include/ssl-cache.conf
- /etc/nginx/conf.d/include/log-stream.conf
- /etc/nginx/conf.d/include/block-exploits.conf
- /etc/nginx/conf.d/include/ssl-ciphers.conf
```

```
- /etc/nginx/conf.d/include/resolvers.conf
Enabling IPV6 in hosts in: /data/nginx
- /data/nginx/proxy_host/8.conf
- /data/nginx/proxy_host/20.conf
- /data/nginx/proxy_host/23.conf
- /data/nginx/proxy_host/6.conf
- /data/nginx/proxy_host/5.conf
- /data/nginx/proxy_host/16.conf
- /data/nginx/proxy_host/11.conf
- /data/nginx/proxy_host/2.conf
- /data/nginx/proxy_host/17.conf
- /data/nginx/proxy_host/15.conf
- /data/nginx/proxy_host/18.conf
- /data/nginx/proxy_host/4.conf
- /data/nginx/proxy_host/3.conf
❯ Docker secrets ...

-------------------------------------
 _   _ ____  __  __
| \ | |  _ \|  \/  |
|  \| | |_) | |\/| |
| |\  |  __/| |  | |
|_| \_|_|   |_|  |_|
-------------------------------------
User:  npm PUID:0 ID:0 GROUP:0
Group: npm PGID:0 ID:0
-------------------------------------

❯ Starting nginx ...
❯ Starting backend ...
[7/28/2026] [12:39:47 PM] [Global   ] › ℹ  info     Using Sqlite: /data/database.sqlite
[7/28/2026] [12:39:47 PM] [Migrate  ] › ℹ  info     Current database version: none
[7/28/2026] [12:39:47 PM] [Setup    ] › ℹ  info     Logrotate Timer initialized
[7/28/2026] [12:39:47 PM] [Setup    ] › ℹ  info     Logrotate completed.
[7/28/2026] [12:39:47 PM] [Global   ] › ℹ  info     IP Ranges fetch is enabled
[7/28/2026] [12:39:47 PM] [IP Ranges] › ℹ  info     Fetching IP Ranges from online
services...
[7/28/2026] [12:39:47 PM] [IP Ranges] › ℹ  info     Fetching https://ip-
ranges.amazonaws.com/ip-ranges.json
[7/28/2026] [12:39:47 PM] [IP Ranges] › ℹ  info     Fetching
https://www.cloudflare.com/ips-v4
[7/28/2026] [12:39:47 PM] [IP Ranges] › ℹ  info     Fetching
```

```
https://www.cloudflare.com/ips-v6
[7/28/2026] [12:39:47 PM] [SSL      ] › ℹ  info      Let's Encrypt Renewal Timer
initialized
[7/28/2026] [12:39:47 PM] [SSL      ] › ℹ  info      Renewing SSL certs expiring within 30
days ...
[7/28/2026] [12:39:47 PM] [IP Ranges] › ℹ  info      IP Ranges Renewal Timer initialized
[7/28/2026] [12:39:47 PM] [SSL      ] › ℹ  info      Completed SSL cert renew process
[7/28/2026] [12:39:47 PM] [Global   ] › ℹ  info      Backend PID 266 listening on port
3000 ...
[7/28/2026] [12:40:24 PM] [Remote Version] › ℹ  info      Fetching
https://api.github.com/repos/NginxProxyManager/nginx-proxy-manager/releases/latest
^Crarcayan@wordpress-docker-vm:~sudo netstat -tulpn | grep -E '80|443|81'1'
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN
54195/docker-proxy
tcp        0      0 0.0.0.0:5678            0.0.0.0:*               LISTEN
53443/docker-proxy
tcp        0      0 127.0.0.1:36565         0.0.0.0:*               LISTEN      1895/code-
1b6a18812
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
1873102/docker-prox
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
1873056/docker-prox
tcp        0      0 0.0.0.0:81              0.0.0.0:*               LISTEN
1873078/docker-prox
tcp6       0      0 :::8080                 :::*                    LISTEN
54202/docker-proxy
tcp6       0      0 :::443                  :::*                    LISTEN
1873109/docker-prox
tcp6       0      0 :::80                   :::*                    LISTEN
1873063/docker-prox
tcp6       0      0 :::81                   :::*                    LISTEN
1873084/docker-prox
rarcayan@wordpress-docker-vm:~$ docker ps --format "table
{{.Names}}\t{{.Status}}\t{{.Ports}}"
NAMES                          STATUS                  PORTS
npm                            Up 11 minutes           0.0.0.0:80-81->80-81/tcp, [::]:80-
81->80-81/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp
phpmyadmin                     Up 9 hours              80/tcp
postgres-shared                Up 9 hours              0.0.0.0:5432->5432/tcp, [::]:5432-
>5432/tcp
portainer                      Up 9 hours              8000/tcp, 9000/tcp, 9443/tcp
pgadmin                        Up 9 hours              80/tcp, 443/tcp
```

```
mariadb-shared                  Up 9 hours              3306/tcp
zammad-nginx                    Up 35 hours             8080/tcp
zammad-zammad-railsserver-1     Up 35 hours (healthy)
zammad-zammad-websocket-1       Up 35 hours
zammad-zammad-scheduler-1       Up 35 hours
zammad-zammad-backup-1          Up 35 hours
zammad-zammad-postgresql-1      Up 35 hours (healthy)   5432/tcp
zammad-zammad-redis-1           Up 35 hours (healthy)   6379/tcp
zammad-zammad-memcached-1       Up 35 hours (healthy)   11211/tcp
zammad-zammad-elasticsearch-1   Up 35 hours             9200/tcp, 9300/tcp
web-server                      Up 35 hours             80/tcp
openproject-proxy-1             Up 35 hours             443/tcp, 2019/tcp, 443/udp,
0.0.0.0:8080->80/tcp, [::]:8080->80/tcp
openproject-worker-1            Up 35 hours             8080/tcp
openproject-cron-1              Up 35 hours             8080/tcp
openproject-web-1               Up 35 hours (healthy)   8080/tcp
openproject-cache-1             Up 35 hours             11211/tcp
openproject-hocuspocus-1        Up 35 hours             1234/tcp
openproject-autoheal-1          Up 35 hours (healthy)
n8n                             Up 35 hours             0.0.0.0:5678->5678/tcp, [::]:5678-
>5678/tcp
inventree-worker                Up 35 hours             8000/tcp
inventree-proxy                 Up 35 hours             80/tcp, 443/tcp, 2019/tcp, 443/udp
inventree-server                Up 35 hours             8000/tcp
inventree-cache                 Up 35 hours             6379/tcp
openemr-server                  Up 35 hours             80/tcp, 443/tcp
wordpress-server                Up 35 hours             80/tcp
lab_gateway_flame               Up 35 hours             0.0.0.0:5005->5005/tcp, [::]:5005-
>5005/tcp
docmost-docmost-1               Up 35 hours             3000/tcp
docmost-redis-1                 Up 35 hours             6379/tcp
rarcayan@wordpress-docker-vm:~$
```

If I use **http://docmost.gravelcloud.com/**, docmost logic show up but my login does not work.

When Docmost loads its UI over HTTP but rejects logins (or sits on a spinning loading loop), it's almost always because **Docmost is dropping WebSockets or security cookies**.

Docmost relies on WebSockets for real-time authentication and uses secure cookie headers (`SameSite` / `Secure`). If Cloudflare or NPM changes the protocol headers, authentication silently fails.

Here is how to get authentication working cleanly:

Step 1: Fix Cloudflare SSL Mode (Must be "Full")

If Cloudflare is set to **Flexible**, Cloudflare strips the HTTPS connection when talking to NPM. Docmost notices this mismatch and rejects authentication cookies.

1. Go to **Cloudflare Dashboard** $\rightarrow$ **SSL/TLS**.
2. Change the encryption mode to **Full** (or **Full (Strict)** if you have a valid cert).

Step 2: Ensure `APP_URL` matches HTTPS in your Docmost Environment

Docmost needs to know its exact public URL to allow login requests and set cookie domain flags.

In your `docker-compose.yml` for **Docmost**, make sure the `APP_URL` variable includes `https://`:

YAML

```
environment:
  - APP_URL=https://docmost.gravelcloud.com
```

If you had to change this, restart Docmost:

Bash

```
docker compose up -d --force-recreate docmost
```

Step 3: Configure NPM Proxy Host Correctly

In NPM, open `docmost.gravelcloud.com` and ensure your settings match this exact setup:

Details Tab

- **Scheme:** `http`
- **Forward Hostname:** `docmost` (or your local container IP)
- **Forward Port:** `3000`
- **Websockets Support: ON** 🟢 *(Crucial for Docmost auth)*

SSL Tab

- Select **Nginx Proxy Manager Custom/Default Certificate** (or any active cert so port 443 responds to Cloudflare's **Full** SSL mode).
- **Force SSL: ON** 🟢

Advanced Tab

Paste **only** this snippet (do not re-add `proxy_http_version` as NPM handles it via the Websockets toggle):

Nginx

```
client_max_body_size 100M;
proxy_read_timeout 3600s;
proxy_send_timeout 3600s;
proxy_buffering off;

# Ensure real client IP and headers pass through Cloudflare
proxy_set_header X-Forwarded-Proto $scheme;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

Step 4: Test in Browser

1. Clear your browser cookies for `gravelcloud.com` (or open an **Incognito/Private window**).
2. Go directly to `[https://docmost.gravelcloud.com/](https://docmost.gravelcloud.com/)` (always use `https://`).
3. Log in.

Okay, it's working now, is this a permanent fix? I used a hub.gravelcloud.com certficate. Do I need to cleanup or revert anything in my cloudflare? Explain to me whyit works now and root cause.

Here is the short answer: **Yes, this is a permanent and completely production-ready setup.**

You do not need to revert or change anything in Cloudflare. You're using an industry-standard proxy configuration.

Why It Works Now & The Root Cause

To understand why it broke and why it works now, let's break down what was happening behind the scenes:

1. The Initial Nginx Crash (Syntax Error)

- **What happened:** Docmost requires WebSockets to process logins and real-time edits. When you enabled **Websockets Support** in NPM *and* pasted `proxy_http_version 1.1;` into the Advanced tab, NPM generated that line twice inside `/data/nginx/proxy_host/16.conf`.

- **The result:** Nginx crashed because duplicate directives are fatal in Nginx syntax. Every time NPM tried to save or renew certificates, Nginx rejected the configuration.

2. The Failed Logins (Cookie & Protocol Mismatch)

- **What happened:** When you temporarily used HTTP or Cloudflare's *Flexible* SSL mode, Cloudflare talked to NPM over plain HTTP.
- **The result:** Docmost saw an incoming `HTTP` request instead of `HTTPS`. Because modern web applications set security flags on authentication cookies (`SameSite` and `Secure`), Docmost refused to issue or accept login cookies over an unencrypted connection, causing your login attempts to fail silently or spin endlessly.

3. Why Using the `hub.gravelcloud.com` Certificate Fixed Everything

- **The Magic of Wildcard/Shared Certs:** Because `hub.gravelcloud.com` and `docmost.gravelcloud.com` both belong to the same top-level domain (`*.gravelcloud.com`), or share a multi-domain Let's Encrypt certificate managed by NPM, attaching that certificate allowed NPM's Nginx instance to safely listen on **HTTPS (Port 443)**.
- **The Chain:** $$\text{User (HTTPS)} \xrightarrow{\quad\text{Cloudflare Edge}\quad} \text{NPM (HTTPS via Port 443)} \xrightarrow{\quad\text{Internal Network}\quad} \text{Docmost (Port 3000)}$$
- Because the entire path from browser to NPM uses HTTPS, Cloudflare's **Full** SSL mode works perfectly, Docmost sees a secure connection, validates the `X-Forwarded-Proto https` headers, accepts the WebSockets connection, and issues the login cookies cleanly.

Do You Need to Clean Up or Change Anything in Cloudflare?

No. Your current Cloudflare configuration is ideal:

- **DNS:** Your wildcard `*.gravelcloud.com` record handles all subdomains automatically without cluttering your DNS panel.
- **Proxying:** Keep the **Orange Cloud (Proxied)** enabled. It provides DDoS protection, hides your real server IP (`35.252.76.214`), and accelerates traffic.
- **SSL/TLS Mode:** Keep it on **Full** (or **Full Strict**). This ensures end-to-end encryption between Cloudflare and your Docker server.

Maintenance Tips Going Forward

1. **Avoid `proxy_http_version` in NPM's Advanced Tab:** Always rely on NPM's **Websockets Support** toggle on the Details tab. It handles WebSocket headers automatically without breaking Nginx syntax.

2. **Auto-Renewals:** NPM will handle renewing that shared certificate in the background automatically—you won't need to manually touch it again.