# Supporting TeslaMate

Maintenance, Backups & Ongoing Operations

*Companion to the TeslaMate Install Guide*

## Summary


**What this document is**

Everything needed to keep a TeslaMate installation healthy after it's up and running: automatic OS patching, updating the containers, backing up and restoring your data, and routine housekeeping.

**Who it's for**

Anyone who completed the TeslaMate Install Guide and now owns an installation that needs occasional care and feeding.

**An assumption, stated up front**

> **This document assumes:**
> You're hosting on Vultr.com and managing your domain through Porkbun.com — the same worked examples used in the Install Guide. The concepts here apply the same way on any other cloud provider or registrar; only the exact menu paths and screens will differ.

**High-level steps in this document**

- 1. What Needs Ongoing Attention
- 2. Automatic OS Security Updates
- 3. Updating TeslaMate, Grafana & Other Containers
- 4. Backing Up Your Installation
- 5. Restoring From a Backup
- 6. Upgrading a Pinned Container to a New Major Version
- 7. Disk & Docker Housekeeping
- 8. Vultr Hosting Notes
- 9. Porkbun Domain Notes
- 10. Reconnecting Your Tesla Account
- 11. Getting Help
- Appendix A: Full Backup Script
- Appendix B: DNS Event Viewer Script
- Appendix C: Change Log

# Supporting TeslaMate

## 1. What Needs Ongoing Attention

| Item | How it's handled |
|---|---|
| OS security patches | Automatic (Section 2) — no regular action needed once configured |
| Container image updates (TeslaMate, Grafana) | Manual, whenever you choose (Section 3) — these use a "latest" tag that only changes when you tell it to |
| Backups | Manual or scripted (Section 4) — run before any update, and periodically otherwise |
| Postgres / Mosquitto version | Pinned, deliberately — only changes if you choose to upgrade them (Section 6), which is rare |
| Disk space / unused Docker data | Occasional housekeeping (Section 7) — check every few months |

## 2. Automatic OS Security Updates

If you followed the Install Guide, Ubuntu is already set up to patch itself automatically, including rebooting when a kernel update requires it. Worth periodically confirming it's still active:

```
dpkg -l | grep unattended-upgrades
systemctl list-timers | grep apt        # apt-daily.timer + apt-daily-upgrade.timer
should both be active
```

To see what it's actually done recently:

```
cat /var/log/unattended-upgrades/unattended-upgrades.log
```

To find out specifically when the OS has rebooted itself for a patch:

```
zgrep -h "WARNING Found /var/run/reboot-required" /var/log/unattended-
upgrades/unattended-upgrades.log* | sort
```

**Note:**

Timestamps in these logs reflect whatever timezone the server was set to at the time each line was written. If you ever change the server's timezone, older log entries won't retroactively relabel themselves — keep that in mind if you're ever comparing timestamps across a timezone change.

### 2.1 What automatic updates do not cover

Two gaps are worth knowing about, because neither is obvious from the package name.

The first is ordinary updates. Ubuntu's default allowed-origins list includes the plain release pocket and the security pockets. The release pocket is frozen on the day the release ships and never receives anything afterwards, so in practice what installs automatically is security patches only. Everyday bug fixes wait in the -updates pocket, which is excluded by default. That is a reasonable default — just not the same as "fully up to date".

The second matters more. Nothing from download.docker.com is covered, because it is not an Ubuntu origin and never appears in the allowed list. The packages you installed from Docker's repository — docker-ce,

docker-ce-cli, containerd.io and the compose and buildx plugins — move only when you run apt yourself. Note especially that containerd.io carries both containerd and runc, the two components that actually create and run your containers, and neither shows up in docker --version. It is easy to leave them behind for months without noticing.

Confirm what your own server includes:

```
grep -h "^Unattended-Upgrade::Allowed-Origins" -A6 \
  /etc/apt/apt.conf.d/50unattended-upgrades
unattended-upgrade --dry-run --debug | grep -i "allowed origins"
```

### 2.2 Checking versions and advisories yourself

Because of that second gap, something has to tell you when a Docker component falls behind. Two questions are worth answering on a schedule, and they are not the same question:

Is a newer version available? That is a release-notes question, and being behind is often fine — a pinned image not moving is usually the point of pinning it.

Is a published security advisory unfixed in what I am running? That is a different question with a different answer, and it is the one that should shorten your timeline. A version can be months behind and carry no outstanding advisory; a version one patch release behind can carry several.

The second question has a free, machine-readable answer. OSV.dev accepts a package name and a version over plain HTTPS and returns the advisories that version is subject to — no account, no API key, no client library. Docker's components are indexed there as Go modules:

```
github.com/docker/docker            Docker Engine
github.com/containerd/containerd/v2 containerd (note the /v2 on 2.x)
github.com/opencontainers/runc      runc
github.com/docker/compose/v2        Docker Compose
```

Ask it a question by hand to see the shape of the answer:

```
curl -s -X POST https://api.osv.dev/v1/query \
  -H 'Content-Type: application/json' \
  -d '{"package":{"name":"github.com/containerd/containerd/v2",
       "ecosystem":"Go"},"version":"2.2.4"}' | head -40
```

Three cautions, each learned the hard way. OSV returns both the GitHub advisory and the Go vulnerability-database entry for the same flaw, so a raw count roughly doubles the real number — group by CVE. An empty result is not proof of safety: if the module name is wrong, a clean answer and an unindexed answer look identical, so query the name once without a version to confirm the database knows it at all. And a "fixed" version lower than the one you run means the advisory belongs to a different release lineage, not that you should downgrade.

Appendix C is a script that does all of this on a schedule and renders the result as a page you can bookmark.

## 3. Updating TeslaMate, Grafana & Other Containers

TeslaMate and Grafana use a "latest" image tag — it only changes when you explicitly pull a new version, not on its own. Postgres and Mosquitto are pinned to specific versions and won't move even when you update the others (see Section 6 for upgrading those deliberately).

**Recommended procedure:**

1. Check github.com/teslamate-org/teslamate/releases for what's changed and any noted breaking changes before updating blindly.
2. Run a backup (Section 4) — this is your rollback point if anything goes wrong.
3. Pull the newer images: docker compose pull
4. Recreate the containers: docker compose up -d
5. Watch the startup logs settle cleanly: docker compose logs -f teslamate (Ctrl+C once it settles with no errors)
6. Verify: log into your TeslaMate and Grafana URLs, confirm history and dashboards are intact.

**Rollback nuance:**

TeslaMate runs a one-way database migration on startup when it detects a newer version. Once that runs, you generally can't just revert the container and expect the database to match. If an update goes badly, the real rollback path is restoring the Postgres backup taken in step 2 into a freshly redeployed stack running the previous version — not just reverting the image tag.

Optional: pinning TeslaMate/Grafana to specific version numbers instead of “latest” (e.g. teslamate/teslamate:1.32.1) trades away automatic-latest convenience for predictability — you decide when to move versions, rather than any docker compose pull silently doing it. Not necessary, just a documented trade-off.

## 4. Backing Up Your Installation

Your car's entire recorded history — every drive, charge, and state change — lives in the Postgres database. Everything else (configs, certificates, Grafana's own settings) is worth backing up too, but the database is what you can't afford to lose.

### 4.1 What gets backed up

- Postgres database dump — your car's complete history
- docker-compose.yml, .env, and .htpasswd — your configuration
- Traefik's ACME data — your HTTPS certificates, so you don't have to re-request them
- Grafana's data volume — dashboard customizations and settings
- Mosquitto's config file
- Your SSH authorized_keys — how you access the server itself

### 4.2 The backup script

The script in Appendix A automates all of the above into one timestamped folder, and keeps the last 5 runs so old backups don't accumulate forever.

1. Copy the script from Appendix A into ~/teslamate/backup_teslamate.sh on your server.
2. Make it executable: chmod +x backup_teslamate.sh
3. Run it: ./backup_teslamate.sh

4. Copy the resulting folder off the server to your own computer (e.g. via WinSCP, scp, or rsync) — a backup that only lives on the same server isn't protecting you from that server failing.

**Container names:**

The script assumes container names like teslamate-database-1, matching the Install Guide's convention of deploying from a folder named ~/teslamate. If you used a different folder name, Docker Compose will have generated different container names — run docker ps to see your actual names and adjust the variables at the top of the script.

## 5. Restoring From a Backup

Restoring the database is the core of a recovery, and it's fairly quick.

1. Copy the backup folder you want to restore from back onto the server, e.g. into ~/restore/.
2. Stop the app containers, but leave Postgres running: docker compose stop teslamate grafana
3. Copy the dump into the database container and restore it: docker cp ~/restore/teslamate.dump teslamate-database-1:/tmp/teslamate.dump then docker exec -it teslamate-database-1 pg_restore -U teslamate -d teslamate --clean --if-exists /tmp/teslamate.dump
4. Bring the app containers back up: docker compose start teslamate grafana
5. Verify: log in and confirm the history matches what you expect from that backup's point in time.

**Important:**

A restore is a full replace, not a merge — anything recorded after that backup was taken will be gone once you restore an older one. This is the right move for “something got corrupted, roll back to a known-good point,” not for recovering a single bad entry while keeping everything else current.

If you ever need to rebuild on a brand-new server entirely (not just restore data on the existing one), the process is nearly identical: deploy the same docker-compose.yml, let Postgres initialize empty, then run the same pg_restore step into the fresh database before bringing TeslaMate online. Traefik will automatically re-request fresh HTTPS certificates once DNS points at the new server's IP.

## 6. Upgrading a Pinned Container to a New Major Version

Postgres and Mosquitto are pinned to specific versions deliberately, so routine updates never silently jump you to a new major version with different behavior. If you ever do want to move to a new major version — say, a future Postgres or Mosquitto release — the safe general approach is the same regardless of which container it is:

1. Read the new version's changelog for breaking changes or new required configuration — major version bumps sometimes change default behavior.
2. Back up first (Section 4).
3. If a config change is required, make it while still on the old version and confirm everything still works — this isolates “did the config change break something” from “did the version upgrade break something.”
4. Only then change the image tag to the new version and recreate the container.

5. Verify logs show a clean startup with no errors, and confirm TeslaMate (or whatever depends on it) reconnects properly.
6. Keep the rollback simple: if it doesn't work, change the image tag back — config changes made in step 3 are typically harmless on the old version too.

As a concrete illustration: Mosquitto 2.x requires an explicit listener and allow_anonymous directive in its config, which older 1.x versions didn't need. Following the staged approach above (config change first, verified, then the version bump) turns a two-variable problem into two separate, individually-testable one-variable changes.

## 7. Disk & Docker Housekeeping

Worth running occasionally to see what's actually consuming disk space and reclaim what's unused:

```
docker ps
docker system df
docker image prune -a
```

- docker ps — shows currently running containers
- docker system df — shows disk usage per category (images, containers, volumes, build cache) and how much is reclaimable
- docker image prune -a — removes any image not currently used by a running container; review the list it shows before confirming

If “Build Cache” shows a large reclaimable amount (common if you've built any custom images), this reclaims it:

```
docker builder prune
```

Check overall disk space with:

```
df -h /
```

## 8. Vultr Hosting Notes

- Downgrading a plan isn't supported — choose a size you're comfortable with; upgrading later is straightforward.
- Shrinking the disk isn't possible without risking data loss — the same applies to most cloud providers.
- Snapshots cost roughly $0.05/GB per month, billed continuously, not one-time — factor this into your monthly cost if you keep one.
- A snapshot is a full-disk, point-in-time image — useful for a fast “the OS won't boot” recovery, but it goes stale if not refreshed periodically. It's a complement to the granular backups in Section 4, not a replacement for them.
- Snapshots can only be restored to a disk of equal or larger size.

## 9. Porkbun Domain Notes

- DNS changes typically propagate within minutes, though it can occasionally take up to an hour.

- The “notes” field on a DNS record is just a personal label for your own reference — it doesn't affect how the record resolves.
- Porkbun offers an optional webhooks feature (API Access → Webhooks) that will POST a signed event to an endpoint you run whenever a domain or DNS record changes on your account. It does not notify you — nothing is emailed or pushed. What it gives you is a record: if you run something at that endpoint to receive the events, you get an audit trail you can read later, covering every domain on the account rather than just one. Building the receiver is an optional project beyond the scope of this guide, but Appendix B is a script for reading the log one produces, should you set one up.

## 10. Reconnecting Your Tesla Account

If TeslaMate ever shows a sign-in screen again (most commonly after using the “Sign out” link, which shouldn't normally be needed), you'll need to reconnect it:

1. Run the same tesla_auth tool used during initial setup (Install Guide, Section 10.1) to get a fresh Access Token and Refresh Token.
2. Enter both on the sign-in screen and click Sign in.

**Worth knowing:**

If you saved the Refresh Token from your original setup, it's reusable — it isn't invalidated by TeslaMate's normal background use. However, Tesla appears to expire refresh tokens on a fixed schedule of roughly 3 months regardless of use, so a saved token more than a few months old will likely have already expired, in which case running tesla_auth again is the fastest path.

## 11. Getting Help

- Official documentation: docs.teslamate.org
- Source code, issues, and discussions: github.com/teslamate-org/teslamate
- Vultr's own documentation and support, for anything hosting-specific: vultr.com
- Porkbun's support, for anything domain/DNS-specific: porkbun.com

## Appendix A: Full Backup Script

```
#!/bin/bash
#
# TeslaMate stack backup script
# Backs up: Postgres DB, docker-compose.yml, .env, .htpasswd, Traefik ACME/cert
# data, Grafana volume, Mosquitto's config file, and ~/.ssh/authorized_keys.
# Run this before any update, and periodically otherwise.

set -u

# --- Configuration: adjust if your setup differs ---
COMPOSE_DIR="${HOME}/teslamate"             # where docker-compose.yml and .env live
DB_CONTAINER="teslamate-database-1"
GRAFANA_CONTAINER="teslamate-grafana-1"
TM_DB_NAME="teslamate"
TM_DB_USER="teslamate"
BACKUP_ROOT="${HOME}/teslamate/Backup"
KEEP_LAST=5

TIMESTAMP=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="${BACKUP_ROOT}/${TIMESTAMP}"
mkdir -p "$BACKUP_DIR"

echo "=== TeslaMate backup started: $TIMESTAMP ==="
echo "Backup folder: $BACKUP_DIR"
echo ""

STATUS_OK=0
STATUS_FAIL=0

# --- 1. Postgres dump ---
echo "[1/6] Dumping Postgres database..."
if docker exec "$DB_CONTAINER" pg_dump -U "$TM_DB_USER" -F c "$TM_DB_NAME" >
"${BACKUP_DIR}/teslamate.dump" 2>"${BACKUP_DIR}/pg_dump.err"; then
    echo "      OK -> teslamate.dump ($(du -h "${BACKUP_DIR}/teslamate.dump" | cut -
f1))"
    rm -f "${BACKUP_DIR}/pg_dump.err"
    STATUS_OK=$((STATUS_OK+1))
else
    echo "      FAILED - see pg_dump.err in backup folder"
    STATUS_FAIL=$((STATUS_FAIL+1))
fi

# --- 2. Config files ---
echo "[2/6] Copying docker-compose.yml, .env, and .htpasswd..."
if cp "${COMPOSE_DIR}/docker-compose.yml" "${BACKUP_DIR}/docker-compose.yml"
2>/dev/null \
   && cp "${COMPOSE_DIR}/.env" "${BACKUP_DIR}/env_file" 2>/dev/null \
   && cp "${COMPOSE_DIR}/.htpasswd" "${BACKUP_DIR}/htpasswd_file" 2>/dev/null; then
```

```
    echo "      OK -> docker-compose.yml, env_file, htpasswd_file"
    STATUS_OK=$((STATUS_OK+1))
else
    echo "      FAILED - check that docker-compose.yml, .env, and .htpasswd exist in
${COMPOSE_DIR}"
    STATUS_FAIL=$((STATUS_FAIL+1))
fi

# --- 3. Traefik ACME (Let's Encrypt) data ---
echo "[3/6] Archiving Traefik ACME certificate data..."
if tar czf "${BACKUP_DIR}/acme.tar.gz" -C "$COMPOSE_DIR" acme
2>"${BACKUP_DIR}/acme.err"; then
    echo "      OK -> acme.tar.gz ($(du -h "${BACKUP_DIR}/acme.tar.gz" | cut -f1))"
    rm -f "${BACKUP_DIR}/acme.err"
    STATUS_OK=$((STATUS_OK+1))
else
    echo "      FAILED - see acme.err in backup folder"
    STATUS_FAIL=$((STATUS_FAIL+1))
fi

# --- 4. Grafana data volume ---
echo "[4/6] Archiving Grafana data volume..."
if docker run --rm --volumes-from "$GRAFANA_CONTAINER" -v "${BACKUP_DIR}:/backup"
alpine \
    tar czf /backup/grafana_data.tar.gz /var/lib/grafana
2>"${BACKUP_DIR}/grafana.err"; then
    echo "      OK -> grafana_data.tar.gz ($(du -h "${BACKUP_DIR}/grafana_data.tar.gz"
| cut -f1))"
    rm -f "${BACKUP_DIR}/grafana.err"
    STATUS_OK=$((STATUS_OK+1))
else
    echo "      FAILED - see grafana.err in backup folder"
    STATUS_FAIL=$((STATUS_FAIL+1))
fi

# --- 5. Mosquitto config ---
echo "[5/6] Copying Mosquitto config..."
if cp -r "${COMPOSE_DIR}/mosquitto-config" "${BACKUP_DIR}/mosquitto-config"
2>"${BACKUP_DIR}/mosquitto.err"; then
    echo "      OK -> mosquitto-config/"
    rm -f "${BACKUP_DIR}/mosquitto.err"
    STATUS_OK=$((STATUS_OK+1))
else
    echo "      FAILED - see mosquitto.err in backup folder"
    STATUS_FAIL=$((STATUS_FAIL+1))
fi

# --- 6. SSH authorized_keys ---
echo "[6/6] Copying ~/.ssh/authorized_keys..."
if cp "${HOME}/.ssh/authorized_keys" "${BACKUP_DIR}/authorized_keys"
2>"${BACKUP_DIR}/ssh.err"; then
    echo "      OK -> authorized_keys"
    rm -f "${BACKUP_DIR}/ssh.err"
```

```
    STATUS_OK=$((STATUS_OK+1))
else
    echo "      FAILED - see ssh.err in backup folder"
    STATUS_FAIL=$((STATUS_FAIL+1))
fi

# --- Retention ---
echo ""
echo "Enforcing retention (keeping last ${KEEP_LAST} backups)..."
ls -1dt "${BACKUP_ROOT}"/*/ 2>/dev/null | tail -n +$((KEEP_LAST+1)) | while read -r
old_dir; do
    echo "  Removing old backup: $old_dir"
    rm -rf "$old_dir"
done

echo ""
echo "=== Backup finished: ${STATUS_OK} succeeded, ${STATUS_FAIL} failed ==="
echo "Location: $BACKUP_DIR"
du -sh "$BACKUP_DIR"

if [ "$STATUS_FAIL" -gt 0 ]; then
    echo ""
    echo "WARNING: one or more steps failed. Check the .err files (if any remain) in:"
    echo "  $BACKUP_DIR"
    exit 1
fi

echo ""
echo ">>> REMINDER: this backup only lives on the server right now. <<<"
echo ">>> Copy this folder to your own computer before moving on:   <<<"
echo ">>>   $BACKUP_DIR"
echo ""

exit 0
```

## Appendix B: DNS Event Viewer Script

Porkbun's webhooks (Section 9) deliver events to an endpoint you run; they do not notify you. If you set up a receiver that appends each event to a log file as one JSON object per line, this script renders that log one line per event — timestamp, event type, what changed, and the detail that matters.

Save it as show_dns_events.sh on the server, point the LOG variable at your receiver's log file, and chmod +x it. Run it with no argument for the last ten events, a number for that many, or all for the whole log.

```
#!/bin/bash
#
# show_dns_events.sh - readable view of a Porkbun webhook event log
#
# Porkbun's outbound webhooks (Porkbun account -> API Access -> Webhooks) POST a
# signed event to an endpoint you run whenever a domain or DNS record changes on
# your account. Those events are only recorded if something receives them; this
# script assumes you have a receiver writing one JSON object per line to the log
# file below. Timestamps in that log are UTC; this converts them to the server's
# local timezone and labels them.
#
# The subscription is account-wide, so events for every domain on the account
# appear here, not only the one running TeslaMate.
#
# Usage:   ./show_dns_events.sh        last 10 events (default)
#          ./show_dns_events.sh 30     last 30 events
#          ./show_dns_events.sh all    the whole log

set -u

# --- Configuration: point this at wherever your receiver writes its log ---
LOG="${HOME}/teslamate/porkbun-webhook-data/porkbun_events.log"

COUNT="${1:-10}"

# --- Checks ---
if [ ! -r "$LOG" ]; then
    echo "Cannot read $LOG"
    echo "Edit the LOG variable at the top of this script, and check the"
    echo "receiver is running:  docker ps --filter name=porkbun"
    exit 1
fi

case "$COUNT" in
    all|[0-9]*) ;;
    *) echo "Usage: $0 [<number of events> | all]"; exit 1 ;;
esac

TOTAL=$(wc -l < "$LOG")
if [ "$COUNT" = "all" ]; then
    SHOWING="$TOTAL"
else
```

```
    SHOWING=$COUNT
    [ "$SHOWING" -gt "$TOTAL" ] && SHOWING=$TOTAL
fi

# --- Header ---
echo "Porkbun webhook events - showing $SHOWING of $TOTAL"
echo ""
printf '%-20s  %-20s  %-26s  %s\n' "WHEN (local)" "EVENT" "NAME / DOMAIN" "DETAIL"
printf '%-20s  %-20s  %-26s  %s\n' "--------------------" "--------------------" \
                                   "--------------------------" "--------------------
"

# --- Body ---
if [ "$COUNT" = "all" ]; then cat "$LOG"; else tail -n "$COUNT" "$LOG"; fi | python3 -
c '
import sys, json, datetime
for line in sys.stdin:
    line = line.strip()
    if not line:
        continue
    try:
        e = json.loads(line)
    except ValueError:
        print("(unparseable line skipped)")
        continue
    d = e.get("payload", {}).get("data", {}) or {}
    try:
        when =
datetime.datetime.fromisoformat(e["received_at"]).astimezone().strftime("%Y-%m-%d
%H:%M %Z")
    except Exception:
        when = str(e.get("received_at", ""))[:19] + " UTC"
    what = d.get("name") or d.get("domain") or ""
    if "content" in d:
        detail = str(d["content"])
    elif "daysToExpire" in d:
        detail = "%s days left (expires %s)" % (d["daysToExpire"],
str(d.get("expireDate", ""))[:10])
    else:
        detail = str(d.get("message", ""))[:40]
    print("%-20s  %-20s  %-26s  %s" % (when, e.get("event_type", ""), what, detail))
'

echo ""
echo "Log records UTC; times above are this server's local timezone."
```

One thing worth knowing: the timestamps Porkbun sends are UTC. The script converts them to the server's local timezone and prints the zone name alongside, so its output cannot be misread as local time when it is not. Read raw, a late-evening change can appear to have happened the following day.

## Appendix C: Version & Security Check

A companion script, check_versions.py, answers both questions from Section 2.2 on a schedule and writes the result as JSON. A single HTML page renders that JSON. Both files ship alongside this document; the script is stdlib-only Python 3 — no pip install, nothing to keep updated.

**What it reports**

For every container in the stack and the host components beneath it, installed version against the newest release upstream, and what a docker compose pull would actually do given the tag you pinned. Separately, a Docker security review: which published advisories are not yet fixed in the version installed, split into those an upgrade would clear today and those with no published fix. For the first group it builds the apt command naming the packages actually affected.

**Installing it**

```
# copy check_versions.py to the server, then:
python3 /root/check_versions.py --dry-run    # prints, writes nothing
python3 /root/check_versions.py              # writes versions.json


# then schedule it
crontab -e
0 0 * * 2 /usr/bin/python3 /root/check_versions.py >> /root/versions-check.log 2>&1
```

That is midnight every Tuesday. It needs outbound HTTPS to reach GitHub, Docker Hub and OSV.dev. Serve versions.html from the same web container as the rest of your site, with versions.json beside it — the page fetches the JSON at load time, so nothing is regenerated when someone visits.

**What you must change**

Lines that were specific to the author's own server are present but commented out, each marked with a CHANGE THIS note naming what to substitute. Work through them before the first run:

```
check_versions.py
  UA               your own domain in the User-Agent string
  OUT_DEFAULT      where your web root actually is
  COMPOSE_DIR      the folder holding your docker-compose.yml
  the webhook component block — delete it unless you run the same receiver


versions.html
  the header subtitle    your own hostnames
  the Reminders nav link  remove it, or point it at a page you have
  the footer paths        wherever you put the script
```

A caution about what you publish. This page states, in one place, exactly which versions you run and which advisories are outstanding against them. That is a useful summary for you and a useful summary for anyone else who finds the URL. Put it behind the same authentication as your TeslaMate instance, or on a private path — not on the public side of your site.

**The DNS events section**

If you run a receiver for your registrar's webhooks, the script also reads its log and the page renders the recent end of it — the same four columns the shell script in Appendix B prints, newest first rather than oldest. Point DNS_EVENT_LOG at your log and the section fills in; leave it alone and the section simply reports itself unavailable, which costs nothing. Nothing in the rest of the page depends on it.

Two things to know before publishing that section. A webhook subscription is usually account-wide, so the table names every domain on the account rather than only the one running TeslaMate. And a record-change event carries the record's new value, so an A record reads as a server IP. Set DNS_EVENT_REDACT = True to replace those values with (hidden) on the page while the shell script still shows them.

**Keeping the two copies in step**

The pair shipped here is generated from the author's own working copies by versions-g.py, in the _Scripts-and-Backup-files folder beside this document. It comments out each instance-specific line, marks it CHANGE THIS, and reports whether anything slipped through. That folder also holds the generic form of the two shell scripts printed in Appendices A and B, so you can take the file rather than retype it from the page.

## Appendix D: Change Log

Tracks revisions to this document. The Reference # of the most recent entry appears in the footer of every page.

| Ref # | Date | What changed |
|---|---|---|
| REV-001 | August 25, 2026 | Initial version. |
| REV-002 | September 5, 2026 | Section 9: the Porkbun webhooks bullet said the feature "can notify an endpoint you control". Corrected — webhooks deliver a signed event to an endpoint you run, but nothing is emailed or pushed to you; what they give you is a readable record, and only if something is there to receive them. New Appendix B (DNS Event Viewer Script) added for reading such a log, with a note that the timestamps are UTC. The change log moved from Appendix B to Appendix C. |
| REV-003 | September 9, 2026 | New Section 2.1 (what automatic updates do not cover — the release pocket is frozen so only security patches install automatically, and nothing from download.docker.com is covered at all, including containerd and runc) and 2.2 (checking versions and advisories yourself, with the OSV.dev query, the Go module names for Docker's components, and the three ways a naive reading of the results misleads). New Appendix C (Version & Security Check) covering the companion check_versions.py and versions.html that ship with this document, what to change before running them, and why the page should not be published openly. The change log moved from Appendix C to Appendix D. |
| REV-004 | September 9, 2026 | **Appendix C** extended: the companion script now also reads a registrar webhook event log and the page renders the recent end of it as a fourth section, so the appendix covers what that section shows, that it is optional and silent when unconfigured, and the two cautions that matter before publishing it — webhook subscriptions are account-wide, and a record-change event carries the record's |

| Ref # | Date | What changed |
|---|---|---|
| | | new value. Also records that the shipped pair is generated by versions-g.py rather than edited by hand, and that the generic forms of the Appendix A and B scripts now ship as files in _Scripts-and-Backup-files beside this document. |
| REV-005 | September 11, 2026 | Appendix A, step 1 — correctness fix, and it invalidates earlier dumps. The line read 'docker exec -t "$DB_CONTAINER" pg_dump ... -F c ... > teslamate.dump'. The -t allocates a pseudo-terminal, whose ONLCR translation turns every newline byte into carriage-return-plus-newline — harmless for text, fatal for the binary -F c format. Measured on a live instance: 145,223 bytes injected into a 38 MB dump, and pg_restore --list refused the file with "could not read from input file: end of file". If you have been running this script, your existing teslamate.dump files will not restore; take a fresh backup after applying this change and verify it with pg_restore --list. The rest of each backup folder is unaffected. The -t is simply removed, here and in the copy of the script shipped in _Scripts-and-Backup-files. |