

Case Study: Rescuing an End-of-Life WordPress Server (Debian 9 to Debian 12) and Adding HTTPS

Author: Giancarlo Angulo · **Date:** 20 October 2026

Environment: Google Cloud Platform (Compute Engine), WordPress, Apache

Work method: Human-directed, AI-assisted (Claude Code driving the `gcloud` CLI, SSH and a browser, with the owner approving each outward-facing step)

1. Summary

A personal WordPress site had been running for more than five years on a single small cloud VM. Its operating system (Debian 9) was past end of life, nothing served HTTPS, and no backup existed. Instead of patching the old machine in place, we built a new Debian 12 VM of the same size, moved the site across, switched traffic by re-attaching the existing static IP address, and issued a free Let's Encrypt certificate. A local database backup script was added at the end. A second small site (a static website on an Ubuntu 20.04 server) was moved the same way, and after a few days the two old servers were deleted.

	Before	After
Operating system	Debian 9 (end of life)	Debian 12
PHP	7.4.9 (custom image build)	8.2
Database	MySQL 5.7.31	MariaDB 10.11
HTTPS	None (connection refused on port 443)	Valid Let's Encrypt certificate, HTTP redirects to HTTPS, auto-renewal
Backups	None (no disk snapshots)	Disk snapshot taken before work; scripted local database backup
Machine size	e2-micro	e2-micro (unchanged)
Public IP / DNS	Static IP	Same static IP, so no DNS change
Exposed admin tool	phpMyAdmin reachable on the public site	Not migrated (removed from the public surface)

2. Background and starting point

- The site is a personal blog and online resume, hosted on a Compute Engine VM created from a Google "click-to-deploy" WordPress image.
- The VM had been up for over five years without a reboot and was never patched at the OS level.
- Browsers could not open the site over HTTPS (the connection was refused). Plain HTTP worked.
- Before this project, the resume page on the site was rewritten into a high-level summary through the WordPress block editor. That work is not part of the migration, but it showed that admin access and a working HTTPS setup were the next priorities.

3. Discovery (read-only)

Everything in this phase only read information.

- **Which VM serves the domain?** DNS resolved the domain to one of two VMs in the project. Checking the project's reserved addresses showed a static IP attached to that VM.
- **Firewall.** Port 443 was already allowed for the VM through a network-tag rule, so the firewall was not the cause of the HTTPS failure.
- **Web server.** Apache was listening only on port 80. No certificate store existed and certbot was not installed.
- **Stack.** Debian 9, PHP 7.4.9 and MySQL 5.7.31 installed outside the system's package manager (typical of the click-to-deploy image).
- **Size of the job.** About 261 MB of WordPress files and 299 MB of database, a 10 GB disk, a 1 GB memory machine.
- **Safety net.** No snapshots existed.

4. Options and the decision

The first plan was the usual one: install certbot and run it against Apache. It failed at the first step. Debian 9 is archived, and the certbot package returned HTTP 404 from both the main mirror and the archive mirror.

Option	Assessment
Install certbot after repointing apt to the archive	Invasive change to a five-year-old production server; packages only partly available.
Use <code>acme.sh</code> (a standalone shell script) on the old VM	Works without apt, but leaves a Debian 9 server running that gets no security updates.
Upgrade in place, 9 to 10 to 11 to 12	Three hops, each able to break Apache, PHP or MySQL. The PHP and MySQL here were custom builds the Debian upgrade would not carry. Debian 11 itself lost support in August 2026, so stopping early was not an option.
Build a new VM and migrate (chosen)	Small site, easy to copy. The old VM stays untouched as an instant fallback. Fixes the OS, PHP, database and HTTPS in one move.

The owner approved building a new machine of the same size and moving the static IP.

5. How the migration was done

Step 0: Snapshot. A disk snapshot of the old VM was created before anything else, as the only restore point.

Step 1: Build the new VM.

- Same machine type (e2-micro), same zone, 10 GB standard disk, Debian 12.
- Same network tag as the old VM, so the existing firewall rules for ports 80 and 443 applied automatically (no firewall changes).
- Added a 1 GB swap file because the machine has only 1 GB of RAM.
- Installed Apache, PHP 8.2 with the usual WordPress extensions, MariaDB, certbot (with the Apache plugin), and enabled the Apache modules `rewrite`, `headers`, `ssl` and `expires`.

Step 2: Export from the old VM.

- Packaged the WordPress directory into one archive, excluding phpMyAdmin and the cache folders.
- Dumped the database with `mysqldump --single-transaction` so the live site was not locked.
- Database credentials were read from `wp-config.php` by a small script and handed to the dump through the environment. They were never printed, never put on a command line, and the helper files were shredded afterwards.
- Verified the dump (valid gzip, 13 tables).

Step 3: Transfer. Both files were copied from the old VM to the workstation and then to the new VM using `gcloud compute scp`.

Step 4: Import and configure.

- Extracted the files into `/var/www/html`, fixed ownership, created the database and user, imported the dump (13 tables).
- Added an Apache virtual host for the domain and `www` alias, and set directory options so WordPress could use its own `.htaccess` rules.

Step 5: Test before the switch. The new VM was tested while DNS still pointed to the old one, by overriding name resolution on the workstation (`curl --resolve`) so the same hostname hit each server directly.

- The new server returned a page the same length as the old one (87,937 characters) containing the same text.
- Login page, REST endpoint and feed behaved identically on both machines, including the same 404s caused by the site's plain permalink setting.

Step 6: Cutover by moving the IP.

- Removed the external address from the old VM, removed the temporary one from the new VM, and attached the existing reserved static IP to the new VM. This is a handful of API calls and the site was serving from the new machine immediately.
- Because the IP did not change, DNS did not change either. There was no propagation delay and no domain-registrar work. (The interruption was a few seconds of API calls; it was not measured.)

Step 7: HTTPS.

- Ran `certbot --apache` for the domain and its `www` alias, with automatic HTTP-to-HTTPS redirect. Issuance worked first time because the domain now pointed at the new server.
- Checked all four URL variants (HTTP, HTTPS, `www` and non-`www`, login page): every one ended on the HTTPS non-`www` address with a verified certificate.
- Set the WordPress Address and Site Address to the HTTPS URL, then confirmed WordPress.com's Jetpack service reported the site as active under the new HTTPS address.

Step 8: Clean-up. Deleted the temporary archives from the workstation and the new VM (shredded on the server). The old VM was left untouched as a fallback.

Step 9: Backups. Wrote a PowerShell script that dumps the database on the VM, downloads it, deletes the server-side copy, verifies the file, and prunes copies older than 30 days. It saves outside any cloud-synced folder because the dump holds user data.

6. Safety practices used

- **Read before write.** Discovery commands were read-only; every change followed a recorded decision.
- **Snapshot first**, and the old VM was kept running as a fallback instead of being modified.

- **Test the new server before the switch** using hostname override.
- **Credentials handled by machine, not by chat.** The AI never saw or printed the database password; the human logged in to Google Cloud and WordPress themselves.
- **Human approval for each outward-facing step:** creating the VM, moving the IP, publishing the page, switching the site to HTTPS.
- **Temporary files shredded**, scripts kept in a scratch area.
- **Minimal public surface.** phpMyAdmin was not carried over.

7. Problems met and how they were solved

Problem	Cause	Fix
certbot would not install	Debian 9 packages no longer downloadable	Rebuilt on Debian 12 instead of altering the old system
New VM refused the first SSH connection	VM was still booting	Retried after it finished starting
PowerShell quoting broke multi-line remote commands	Nested quotes between PowerShell and bash	Wrote small shell scripts, copied them to the VM and ran them
Backup script failed with "positional parameter cannot be found"	A function named Gc collided with PowerShell's built-in alias for Get-Content	Renamed the function
Backup verification rejected a good dump	MariaDB 10.11 starts dumps with a sandbox comment, not "dump"	Check the first few lines instead of the first line
Browser automation hit the login page	Several Chrome profiles were connected, and only one was signed in	Listed the connected browsers and used the signed-in one
Could not edit the site through the WordPress tools	Required a paid Jetpack AI plan	Used the browser and the block editor directly instead
Re-attaching a reserved IP failed (second site)	The IP was on the Standard network tier, the new VM on Premium	Re-attached with --network-tier STANDARD

8. Results

- HTTPS works on the apex domain and www, with a valid Let's Encrypt certificate (90-day validity, renewing automatically).
- The operating system, PHP and database are all currently supported versions.
- The site content is identical to before, verified by comparison.
- Same VM size and cost class, same static IP, no DNS change.
- A snapshot and a scripted local backup now exist where there were none.
- The old server was never modified during the move, so rollback was one IP re-attach. After a soak period of several days, both old servers were deleted, saving about \$15 a month. Two disk snapshots were kept as the rollback.

9. A second site moved the same way

A static company site (39 MB of HTML, no database) ran on an Ubuntu 20.04 server with its own reserved IP. A Google Cloud VM can hold only one external IP, so the site could not simply share the WordPress server and keep its address. Instead, a new Debian 12 VM of the smallest size (e2-micro, 10 GB) was built, the site was copied across, tested with a hostname override, and the reserved IP was moved over. HTTPS was issued with certbot for `www`, the name that points at the server. A few unneeded files (old `php.ini` copies and a debug log) were left behind, and directory listings were turned off. One mistake: the reserved IP was on Google's Standard network tier, and the new VM defaulted to Premium, so the first attach failed and the site was down for a few minutes until it was re-attached with the Standard tier.

10. Lessons learned

- **When the platform is end of life, rebuild instead of repairing.** A small site makes migration cheaper and safer than a multi-hop in-place upgrade.
- **A reserved static IP makes cutover easy.** Moving the address between machines avoids DNS delays.
- **Test with the real hostname before switching.** A hostname override lets you compare old and new servers directly.
- **Take a snapshot first, and keep the old system as the fallback.**
- **Wrap fragile remote commands in script files** rather than nesting quotes.
- **Keep secrets out of logs, command lines and chat.** Read them in place, pass them through the environment, and delete the helper files.
- **The first plan can fail, and that is fine.** The discovery work that made the failure visible also pointed to the better plan.
- **Check the network tier before moving a static IP,** and keep the old server in place until the new one answers on that IP.

11. Follow-ups

- Delete the two kept snapshots after a few weeks, once the old servers are certainly not needed.
- Remove the SSH key added to the project's metadata, if it is not wanted.
- Register an email address with the certificates for expiry warnings (both were issued without one).
- Schedule the backup script, for example daily with Windows Task Scheduler.
- Confirm Jetpack's connection on its screen in the WordPress dashboard.

12. Tools used

Google Cloud Compute Engine and the `gcloud` CLI · OpenSSH/PuTTY · Debian 12, Apache, PHP 8.2, MariaDB · Let's Encrypt with certbot · WordPress block editor · PowerShell · Claude Code (AI assistant that ran the commands, with human approval).