8-CHAPTER COURSE

Web Hosting Essentials

DNS, Cloudflare, Apache virtual hosts, tunnels, dynamic DNS, and the self-hosting vs VPS trade-off — for a real home server running on a restrictive ISP.

WEB HOSTING ESSENTIALS — CHAPTER 1

DNS Fundamentals

Chapter 1 — DNS Fundamentals

Almost every problem with web hosting eventually traces back to DNS. A misconfigured record, an expired cache, or a CNAME in the wrong place can make a perfectly healthy server completely unreachable. Understanding how DNS actually works — not just how to copy and paste records — gives you the ability to diagnose these problems confidently rather than guessing.

What this chapter covers: What DNS is and why it exists. The full resolution chain — how a browser turns "osztromok.com" into an IP address. Record types: A, AAAA, CNAME, MX, TXT, NS. TTL and why it controls how long changes take to propagate. Querying DNS with dig and nslookup. The CNAME-at-apex limitation that catches everyone out.

What DNS Is and Why It Exists

The internet routes traffic using IP addresses — numbers like `82.2.236.221`. Humans are bad at remembering numbers and good at remembering names. DNS (Domain Name System) is the phone book that translates between the two: you type `osztromok.com`, DNS looks up the IP address, and your browser connects to that address.

DNS is a distributed, hierarchical database. No single server knows all domain names — instead the responsibility is delegated down a tree of servers, each authoritative for their own portion of the namespace. This design means it scales to handle billions of queries per second across the entire internet.

Why this matters for web hosting: When you set up your website, you're not just configuring the web server — you're also configuring the DNS records that tell the internet how to find it. Get the DNS wrong and nobody can reach your server, even if the server itself is working perfectly. This is what happened with osztromok.com earlier — the server was running fine but DNS was pointing to the wrong IP.

How DNS Resolution Works

When you type `osztromok.com` in a browser, several servers are consulted in sequence before the IP address is found. This happens in milliseconds, but understanding each step helps you diagnose where things go wrong.

Your browser │ │ 1. Check local DNS cache — is the answer already known? │ If yes → use it (no network request needed) │ ▼ Recursive Resolver (your ISP's DNS server, or 8.8.8.8 / 1.1.1.1) │ │ 2. Check resolver cache — has it recently looked this up? │ If yes → return cached answer │ │ 3. If not cached, start from the top: │ ▼ Root Nameservers (13 clusters worldwide, addresses hardcoded) │ │ "I don't know osztromok.com, but .com is handled by Verisign" │ Returns the address of the .com TLD nameservers │ ▼ TLD Nameservers (.com, .net, .org, .co.uk etc.) │ │ "I don't know osztromok.com specifically, but IONOS is │ authoritative for it" — Returns IONOS nameserver addresses │ ▼ Authoritative Nameserver (IONOS, Cloudflare, etc.) │ │ "Yes! osztromok.com has an A record pointing to 82.2.236.221" │ ▼ Recursive Resolver (caches the answer for TTL seconds) │ ▼ Your browser → connects to 82.2.236.221

The recursive resolver does the legwork — it consults the root, TLD, and authoritative servers so your browser doesn't have to. It also caches answers so subsequent lookups for the same domain are instant. The length of time it caches is controlled by the **TTL** value on each DNS record.

Authoritative vs recursive: The *authoritative* nameserver is the one that holds your actual DNS records — for osztromok.com this is IONOS (or Cloudflare, if you transfer DNS there). A *recursive resolver* is the intermediary that queries on your behalf — your ISP runs one, and so do Google (8.8.8.8), Cloudflare (1.1.1.1), and others. When you change a DNS record at IONOS, you're changing the authoritative answer — but resolvers around the world keep serving their cached copy until the TTL expires.

DNS Record Types

Each DNS record has a type that defines what kind of information it holds. You'll encounter these constantly when managing a website.

A Address record
Maps a domain name to an **IPv4 address**. The most fundamental record — this is what tells the internet where your website lives. Every publicly accessible domain needs at least one A record.
osztromok.com. 300 IN A 82.2.236.221
AAAA IPv6 address record

Maps a domain name to an **IPv6 address** (the newer, longer address format). Works exactly like an A record but for IPv6. Most modern websites have both A and AAAA records. If your ISP doesn't give you an IPv6 address, you can skip this.
osztromok.com. 300 IN AAAA 2001:db8::1
CNAME Canonical name (alias)
Points one domain name to **another domain name** (not an IP). The resolver follows the chain until it finds an A record. Used for subdomains and services like Cloudflare Tunnel. **Cannot coexist with other records for the same name, and cannot be used at the root domain (apex) on most providers.**
www 300 IN CNAME osztromok.com.
MX Mail exchanger
Tells email servers where to deliver mail for your domain. Has a **priority number** — lower number = higher priority. Multiple MX records provide redundancy. If you use Google Workspace or Microsoft 365 for email, they give you specific MX records to add.
osztromok.com. 300 IN MX 10 mail.osztromok.com.
TXT Text record
Stores arbitrary text data. Used for **domain verification** (Google, GitHub, etc. ask you to add a TXT record to prove you own the domain), **SPF** (specifies which servers are allowed to send email for your domain), and **DKIM** (email signing keys).
osztromok.com. 300 IN TXT "v=spf1 include:_spf.google.com ~all"
NS Nameserver record
Declares which servers are **authoritative** for your domain. Set at your registrar (where you bought the domain), not in your DNS management panel. Changing NS records is what "moving DNS to Cloudflare" means — you point them at Cloudflare's nameservers instead of IONOS's.
osztromok.com. 86400 IN NS ns1.ionos.co.uk.
The CNAME-at-apex limitation: You cannot use a CNAME record for your root domain (`osztromok.com`) on most DNS providers — only for subdomains (`www.osztromok.com`). This is a limitation of the DNS specification: the root domain must have an SOA and NS record, and CNAME cannot coexist with any other record type. Cloudflare solves this with "CNAME flattening" — it internally resolves the CNAME and serves the resulting IP as an A record, making it transparent. This is one of the main practical reasons to move DNS to Cloudflare.

TTL — Time To Live

Every DNS record has a TTL value measured in seconds. This tells resolvers how long they're allowed to cache the answer before asking again. A record with TTL 300 can be cached for 5 minutes; a record with TTL 86400 can be cached for 24 hours.

TTL directly controls how long a DNS change takes to take effect across the internet. If your A record has a TTL of 86400 and you change the IP address, resolvers that cached it will keep serving the old address for up to 24 hours.

60–300
1–5 minutes
Use when you're about to make a change. Set this 24 hours *before* the change so caches clear quickly when you update the record.
3600
1 hour
A sensible default for most records. Balances propagation speed with resolver load. Good for records that change occasionally.
86400
24 hours
For stable records that rarely change — MX records, NS records. Reduces DNS query load. Don't use for A records on a dynamic IP.
The golden rule for DNS changes: Before making a significant change (moving hosts, changing IP, moving DNS to Cloudflare), lower the TTL on the affected records to 300 (5 minutes) and wait for the current TTL to expire. Then make your change. This means the old cached values drain away quickly. Once the change is stable, you can raise the TTL again.

DNS Propagation — what it actually means

"DNS propagation" is often described as taking 24–48 hours. In practice with a low TTL it can be as fast as a few minutes. What's actually happening is that resolvers around the world are all waiting for their cached copies to expire at different times — some cached it an hour ago, some cached it 30 seconds ago. "Propagation" is just waiting for the last cached copy in the world to expire.

Checking propagation: The tool **dnschecker.org** queries DNS servers in dozens of locations around the world simultaneously and shows which ones have picked up your new record. This is much more useful than just checking on your own machine, where your local resolver might have a long-lived cache.

Querying DNS with dig and nslookup

`dig` is the standard tool for querying DNS from the command line on Linux. It shows you exactly what a nameserver returns, including TTL, record type, and which server answered. `nslookup` is an older alternative available on Windows and Linux.

── Basic lookups ── $ dig osztromok.com ;; ANSWER SECTION: osztromok.com. 300 IN A 82.2.236.221 # Domain TTL Class Type Value # TTL=300 means this answer will be cached for 5 minutes # Look up a specific record type $ dig osztromok.com MX $ dig osztromok.com TXT $ dig osztromok.com NS $ dig osztromok.com CNAME # ── Useful flags ── $ dig osztromok.com +short # just the answer, no extra info 82.2.236.221 $ dig osztromok.com +noall +answer # answer section only, cleaner output # ── Query a specific DNS server (bypass your local resolver) ──────── $ dig osztromok.com @8.8.8.8 # ask Google's public resolver $ dig osztromok.com @1.1.1.1 # ask Cloudflare's public resolver $ dig osztromok.com @ns1.ionos.co.uk # ask IONOS directly (authoritative) # Asking the authoritative server directly shows the "true" value # without any resolver caching. Useful when checking if your change # has taken effect at the source before waiting for propagation. # ── Trace the full resolution chain ──────────────────────────────────── $ dig osztromok.com +trace ;; Received 97 bytes from 198.41.0.4#53(a.root-servers.net) in 12 ms ;; Received 172 bytes from 192.5.6.30#53(a.gtld-servers.net) in 8 ms osztromok.com. 300 IN A 82.2.236.221 ;; Received 62 bytes from 212.227.123.16#53(ns1.ionos.co.uk) in 14 ms # Shows every step: root → .com TLD → IONOS authoritative → answer # ── Find which nameservers are authoritative for a domain ──────────── $ dig osztromok.com NS +short ns1.ionos.co.uk. ns1102.ui-dns.org. # ── Check TTL on a cached record ── # Query the same record twice — the second TTL will be lower if cached $ dig osztromok.com +noall +answer osztromok.com. 298 IN A 82.2.236.221 # TTL=298 (not 300) means your resolver cached this 2 seconds ago # ── Windows: nslookup ── C:\> nslookup osztromok.com C:\> nslookup -type=MX osztromok.com C:\> nslookup osztromok.com 8.8.8.8 # query Google instead of local resolver

Scenario — Diagnosing Why a Domain Isn't Working

Scenario · Chapter 1

Your website was working yesterday but is unreachable today. You haven't changed anything on the server. Use dig to systematically identify whether this is a DNS problem.

1

Find out what IP the domain currently resolves to.

$ dig osztromok.com +short 82.2.236.221 # Note this IP — you'll compare it to the server's actual IP next.

2

Confirm what IP your server actually has right now.

$ curl ifconfig.me 86.11.204.43 # The server's public IP is 86.11.204.43 but DNS says 82.2.236.221. # The IP has changed (common on residential connections without a static IP). # This is a DNS problem — the A record needs updating.

3

Check the TTL to know how long the fix will take to propagate.

$ dig osztromok.com +noall +answer osztromok.com. 82847 IN A 82.2.236.221 # TTL=82847 seconds (~23 hours) — the old IP could be cached for nearly a day. # Update the A record in IONOS immediately, but also lower the TTL to 300 # so anyone who queries fresh will get the new address quickly.

4

After updating the record, confirm the authoritative server has the new value.

Find the authoritative nameserver first $ dig osztromok.com NS +short ns1.ionos.co.uk. # Query it directly — no caching $ dig osztromok.com @ns1.ionos.co.uk +short 86.11.204.43 # Good — the authoritative server already has the new IP. # The rest of the world will catch up as their cached copies expire.

5

Flush your local cache to see the new value immediately on your own machine.

Linux (systemd-resolved) $ sudo systemd-resolve --flush-caches # macOS $ sudo dscacheutil -flushcache && sudo killall -HUP mDNSResponder # Windows (run as Administrator) C:\> ipconfig /flushdns

This is exactly what happened with osztromok.com earlier in this session — the server's IP changed after the router reset, but the DNS record still pointed to the old address. Dynamic IPs are covered in depth in Chapter 7 (Dynamic DNS with DuckDNS and ddclient), which automates this update process.

Quick Reference — Chapter 1

Command / Concept	Purpose	Notes
dig domain.com +short	Quick A record lookup — returns just the IP	Add `MX`, `TXT`, `NS`, `CNAME` to look up specific record types

Command / Concept	Purpose	Notes
dig domain.com @8.8.8.8	Query a specific resolver (bypass local cache)	Use `@ns1.ionos.co.uk` to query your authoritative server directly
dig domain.com +trace	Show the full resolution chain from root → TLD → authoritative	Useful for diagnosing delegation problems
dig domain.com +noall +answer	Show only the answer section including TTL	TTL decreasing between queries confirms the resolver is caching
curl ifconfig.me	Your server's current public IP address	Compare to dig output — a mismatch means the A record needs updating
A record	Domain → IPv4 address. The core record every website needs.	Use AAAA for IPv6. Both can exist simultaneously.
CNAME record	Domain → another domain name (alias)	Cannot be used at the root/apex domain. Cannot coexist with other records.
MX record	Where to deliver email for the domain	Has a priority number — lower = higher priority. Multiple allowed.
NS record	Which servers are authoritative for the domain	Set at your registrar. Changing this = moving DNS to a new provider.
TTL	How long resolvers cache a record (in seconds)	Lower TTL before making changes. 300 = 5 min, 3600 = 1 hr, 86400 = 24 hr.

WEB HOSTING ESSENTIALS — CHAPTER 2

Managing DNS at Your Registrar

Chapter 2 — Managing DNS at Your Registrar

Knowing how DNS works in theory is step one. Step two is being able to make the right changes in the right place quickly and confidently. This chapter covers the practical side: navigating IONOS's DNS panel, adding and editing every record type you'll need, creating subdomains, and verifying that your changes have taken effect.

What this chapter covers: Registrar vs DNS host — understanding where to make changes. The @ symbol and DNS hostname notation. Adding and editing A, CNAME, MX, and TXT records in IONOS step by step. Subdomains — what they are, when to use them, and how to create them. The www vs naked domain debate. Verifying changes with dig. Common mistakes that catch everyone out.

Registrar vs DNS Host — Where Do You Make Changes?

Two different roles are involved in managing a domain, and they're often confused:

- **Registrar** — the company you bought the domain from. They hold the registration (the right to use the domain name) and control the NS records — the records that point to whichever nameserver is authoritative. IONOS is your registrar for osztromok.com.
- **DNS host (authoritative nameserver)** — the service that holds your actual DNS records: A, CNAME, MX, TXT, etc. By default this is also IONOS, because when you bought the domain they set up hosting on their own nameservers. If you later move DNS to Cloudflare, IONOS remains your registrar (you still pay them for the domain) but Cloudflare becomes the DNS host.

┌───┐ │ IONOS (Registrar) │ │ • Holds the domain registration │ │ • Controls NS records (nameserver delegation) │ │ • Currently also the DNS host (both roles in one) │ └───┘ │ │ NS records point to: ▼ ┌───┐ │ IONOS Nameservers (DNS host) │ │ • Holds your A, CNAME, MX, TXT records │ │ • This is where you add/edit/delete records │ │ • Answers queries from resolvers worldwide │ └───┘ When you move DNS to Cloudflare (Chapter 3): • IONOS still = registrar (keeps the registration) • NS records updated to point at Cloudflare • Cloudflare becomes the DNS host (you manage records there)

For osztromok.com right now: All DNS management is done inside IONOS — log in, go to Domains & SSL → DNS → click on osztromok.com. Everything in this chapter takes place in that panel.

DNS Hostname Notation — The @ Symbol

When you add a record in a DNS panel, you need to specify which hostname the record applies to. There are three forms you'll encounter:

What you type	What it means	Example result
@	The root/apex domain itself — the domain with nothing in front of it	osztromok.com
www	The www subdomain — just type the subdomain label, not the full name	www.osztromok.com
blog	Any other subdomain label	blog.osztromok.com
*.osztromok.com	Wildcard — matches any subdomain not explicitly defined	anything.osztromok.com

IONOS uses @ for the root domain in some views and the full domain name in others. If you see a field labelled "Hostname" or "Host", entering @ applies the record to `osztromok.com` itself. Entering `www` applies it to `www.osztromok.com`.

Adding and Editing Records in IONOS

In IONOS, navigate to: **Domains & SSL → DNS → select your domain → Add Record**. The form fields differ slightly by record type. Here's what to enter for each.

A Record — pointing your domain to an IP address

A Record — Root domain pointing to your server
Type

A
Select from the Type dropdown
Hostname
@
@ means the root domain (osztromok.com itself)
Points to
82.2.236.221
Your server's public IP address
TTL
300
5 minutes — keep low if your IP might change
A Record — Subdomain pointing to the same server
Type
A
Hostname
www
Creates www.osztromok.com
Points to
82.2.236.221
Same IP as the root domain
TTL
300

CNAME Record — alias from one name to another

CNAME Record — www as an alias for the root domain
Type
CNAME
Select CNAME from the Type dropdown
Hostname
www
The alias — www.osztromok.com
Points to
osztromok.com.
The target — note the trailing dot (fully qualified). Some panels add it automatically.
TTL
3600
1 hour — CNAME targets rarely change
A record or CNAME for www? Both work. Use an **A record** pointing to your IP if you want simplicity and one less DNS lookup. Use a **CNAME** pointing to your root domain if you want www to automatically follow wherever the root domain points (useful if you change hosts). Never use CNAME for the root domain itself (@) — only subdomains.

MX Record — for email delivery

MX Record — if you host email yourself
Type
MX
Hostname
@
Email for the root domain (user@osztromok.com)
Points to
mail.osztromok.com.
The mail server hostname (must have its own A record)
Priority
10
Lower number = higher priority. 10 is conventional for a single mail server.
TTL

3600
MX records change rarely — 1 hour is fine

TXT Record — domain verification and SPF

TXT Record — domain ownership verification (example: Google Search Console)
Type
TXT
Hostname
@
Verification TXT records almost always go on the root domain
Value
google-site-verification=abc123xyz
Copy exactly from the service asking for verification — don't add quotes
TTL
3600
Editing an existing record in IONOS: Find the record in the list, click the pencil/edit icon on the right. Change the value (e.g. new IP address), save. The record updates immediately on IONOS's servers — propagation to the rest of the world depends on the current TTL. To delete a record, use the bin/trash icon. Always double-check you're deleting the right record — there's usually no undo.

Subdomains — What They Are and When to Use Them

A subdomain is any label placed in front of your domain: `www.osztromok.com`, `blog.osztromok.com`, `shop.osztromok.com`. In DNS terms, creating a subdomain just means adding a record with that label as the hostname. You don't need to "register" a subdomain — you own the whole domain, including all possible subdomains.

www.osztromok.com
A or CNAME
The traditional web subdomain. Many people expect www to work even if you only advertise the root domain. Worth setting up even if you redirect it to the root.
blog.osztromok.com
A or CNAME
A separate section of your site. Could point to the same server (handled by an Apache VirtualHost — Chapter 5) or a completely different server.
mail.osztromok.com
A record
Mail server hostname. Required if you host your own email. MX records point to a hostname, not an IP — so mail.osztromok.com needs its own A record.
*.osztromok.com
A or CNAME wildcard
Catch-all for undefined subdomains. Anything.osztromok.com will resolve to this record. Useful for development environments or multi-tenant apps.
Important: Creating a subdomain DNS record is only *half* the job. The DNS record tells the internet which server to connect to — but your web server also needs to be configured to respond to that subdomain. Apache does this with Virtual Hosts (covered in Chapter 5). A DNS record pointing at your server without a corresponding VirtualHost will show Apache's default page or a 404, not your intended content.

The www vs naked domain debate

Should your site be at `osztromok.com` (naked/apex domain) or `www.osztromok.com`? The practical answer: **set up both and redirect one to the other**. Here's how to decide which is canonical:

- **Root domain canonical** (`osztromok.com` is primary, www redirects to it) — simpler to type, looks cleaner. The setup you already have. Apache redirects www → root.
- **www canonical** (`www.osztromok.com` is primary, root redirects to it) — historically preferred because CNAME records work on www but not on the root, making it easier to point to CDNs and external services. Less relevant now that Cloudflare handles CNAME flattening.

Either choice is fine. What matters is consistency — pick one and redirect the other so visitors and search engines always land on the same URL.

Verifying Your Changes

After making a change in IONOS, verify it immediately at the authoritative server. # This confirms IONOS has saved it — before waiting for propagation elsewhere. # Find your authoritative nameservers $ dig osztromok.com NS +short ns1.ionos.co.uk. ns1102.ui-dns.org. # Check the record directly at the authoritative server (no caching) $ dig osztromok.com @ns1.ionos.co.uk +short 82.2.236.221 # good — authoritative server has the right answer # Check a specific subdomain $ dig www.osztromok.com @ns1.ionos.co.uk +short 82.2.236.221 # Check your local resolver — TTL shows how long until the cached copy expires $ dig osztromok.com +noall +answer osztromok.com. 247 IN A 82.2.236.221 # TTL=247: your resolver cached this 53 seconds ago (300-247=53). Old value still cached. # Check what the world sees — query several resolvers $ dig osztromok.com @8.8.8.8 +short # Google $ dig osztromok.com @1.1.1.1 +short # Cloudflare $ dig osztromok.com @9.9.9.9 +short # Quad9 # Or use the online tool dnschecker.org to see dozens of locations at once

Scenario 1 — Your IP Has Changed, Update the A Record

Scenario · Chapter 2 · Scenario 1
Your router was reset and your ISP assigned a new public IP. osztromok.com is now unreachable. Update the A record in IONOS to restore access.
1
Find your new public IP. Run this on your server (or any device on your home network):

$ curl ifconfig.me 86.11.204.43
2
Log in to IONOS → Domains & SSL → DNS → osztromok.com. Find the existing A record for `@` (the root domain). Note the old IP currently shown.
3
Edit the A record. Click the pencil icon next to the `@` A record. Change the IP from the old value to `86.11.204.43`. Also lower the TTL to `300` while you're there. Save.
4
If you also have a www A record (not a CNAME), update that too — it points to the same IP and also needs changing.
5
Verify the authoritative server has the new value immediately:

$ dig osztromok.com @ns1.ionos.co.uk +short 86.11.204.43 # Good — the change is live on IONOS. The rest of the internet will follow # as their cached copies of the old IP expire (within 5 minutes at TTL=300).
This is the manual version of what Dynamic DNS (Chapter 7) automates — a script on your server detects when the IP changes and updates IONOS automatically via their API, so you never have to do this by hand again.

Scenario 2 — Add a www Subdomain

Scenario · Chapter 2 · Scenario 2
osztromok.com works but www.osztromok.com returns "server not found". Add a DNS record so both work, then verify.
1
Confirm www has no record yet:

$ dig www.osztromok.com +short # no output = no record exists
2
In IONOS, add a new record:
Type: `A` · Hostname: `www` · Points to: `82.2.236.221` · TTL: `300`

Or use a CNAME instead:
Type: `CNAME` · Hostname: `www` · Points to: `osztromok.com.` · TTL: `3600`
3
Verify it immediately at the authoritative server:

$ dig www.osztromok.com @ns1.ionos.co.uk +short 82.2.236.221
4
Test in a browser. DNS now resolves www.osztromok.com to your server's IP — but Apache needs a VirtualHost to respond to it. If you see Apache's default page instead of your site, that's the next step (covered in Chapter 5). If you want www to redirect to the root domain, that's also handled in Apache's VirtualHost config.

DNS and web server configuration are separate concerns. DNS tells the browser which server to connect to. Apache decides what content to serve once the connection arrives. Both need to be set up correctly.

Common Mistakes

Using a CNAME for the root domain (@)
IONOS may let you save it, but most resolvers will ignore it or behave unpredictably. The root domain must use an A record (or AAAA for IPv6). Only subdomains like `www` or `blog` can use CNAME.
Adding a CNAME and an A record for the same hostname
A CNAME record cannot coexist with any other record for the same name. If `www` has a CNAME pointing to `osztromok.com.`, you cannot also have an A record for `www`. Delete one before adding the other.
Forgetting the trailing dot in CNAME targets
In DNS, `osztromok.com.` (with trailing dot) means the fully-qualified root domain. Without the dot, some systems interpret it as relative to the current zone. IONOS usually handles this for you, but if you're editing a zone file directly, always use the trailing dot on target names.
Not waiting for the TTL to expire before testing
If you had a long TTL (e.g. 86400), resolvers can serve the old answer for up to 24 hours. Always query **the authoritative server directly** (`dig domain @ns1.ionos.co.uk`) to confirm the change is saved, then be patient for propagation elsewhere.
Deleting the wrong record
If you have both an A record and a TXT record for `@`, they look similar in the list. Deleting the A record by mistake takes the site down immediately. Always read the Type and Value columns carefully before clicking delete.
Thinking DNS change = site immediately works
DNS resolves which server to connect to — the web server itself still has to be running, port-forwarded (or tunnelled), and configured to respond to that hostname. A correct DNS record pointing at a misconfigured or offline server still gives the visitor an error.

Quick Reference — Chapter 2

Task	Record type	Hostname field	Value field
Root domain → IP	A	@	Your server's public IP
www → same IP	A	www	Your server's public IP
www → root alias	CNAME	www	osztromok.com.
blog subdomain	A or CNAME	blog	IP or target hostname
Email delivery	MX	@	mail.osztromok.com. (priority: 10)
Domain verification	TXT	@	Verification string from the service
Cloudflare Tunnel	CNAME	www (not @)	<tunnel-id>.cfargotunnel.com.

Verification command	Purpose
dig domain.com @ns1.ionos.co.uk +short	Check authoritative answer immediately after saving — no propagation wait
dig www.domain.com +short	Check what your local resolver currently returns (may be cached)
dig domain.com +noall +answer	See the TTL — how many seconds until your resolver re-queries
dig domain.com @8.8.8.8 +short	Check what Google's resolver currently returns (different cache to yours)

WEB HOSTING ESSENTIALS — CHAPTER 3

Moving DNS to Cloudflare

Chapter 3 — Moving DNS to Cloudflare

Moving your DNS to Cloudflare is the single highest-leverage change you can make to a self-hosted website. It solves the CNAME-at-apex limitation that blocked osztromok.com from using Cloudflare Tunnel, makes future changes propagate in seconds rather than hours, and adds a layer of DDoS protection — all for free. The process takes about 15 minutes and your site stays online throughout.

What this chapter covers: Why Cloudflare is worth using even on the free tier. What "moving DNS" actually means — registrar stays IONOS, only the nameservers change. The orange cloud (proxied) vs grey cloud (DNS only) — the most important concept in Cloudflare. The full migration walkthrough: adding the domain, reviewing the auto-imported records, updating nameservers at IONOS, and verifying the transfer. CNAME flattening and why it fixes the apex problem. What to expect during the transition period.

Why Move to Cloudflare?

CNAME Flattening
Cloudflare can serve a CNAME target as an A record at the root domain — something standard DNS cannot do. This is what makes Cloudflare Tunnel work on **osztromok.com** (not just www).
Near-instant propagation
DNS changes on Cloudflare take effect in **seconds**, not hours. Their global Anycast network serves the updated answer from the moment you click Save.
DDoS protection
When the orange cloud is on, your real IP is hidden behind Cloudflare's network. Attacks are absorbed at Cloudflare's edge before they reach your server — included free.
Free SSL certificates
Cloudflare issues and renews TLS certificates for your domain automatically. Visitors get HTTPS even if your origin server only serves HTTP — though properly configuring end-to-end SSL is better (Chapter 1 of the Security course).
Tunnel integration
Cloudflare Tunnel (Chapter 4) creates DNS records in your Cloudflare zone automatically when you configure a public hostname — no manual record creation needed.
Genuinely free
The free tier includes unlimited DNS queries, unlimited Tunnel usage, DDoS protection, and SSL. No credit card required. Cloudflare's business model is selling to enterprises — the free tier is a genuine product, not a trial.
What does NOT change: IONOS remains your registrar — you still pay IONOS to renew the domain each year. Your domain name stays the same. Your server stays the same. The only thing that changes is which nameservers answer DNS queries for osztromok.com.

The Orange Cloud vs Grey Cloud — The Most Important Cloudflare Concept

Every DNS record in Cloudflare has a toggle that controls whether traffic is **proxied through Cloudflare** or goes directly to your server. This is represented by an orange cloud icon (proxied) or a grey cloud icon (DNS only). Getting this right is the single most common source of confusion when setting up Cloudflare.

🟠 Orange Cloud — Proxied
Traffic flows **through Cloudflare's network** before reaching your server. DNS returns Cloudflare's IP addresses, not your real IP.

You get:

- DDoS protection — your real IP stays hidden
- CDN caching — static files served from Cloudflare's edge
- Free HTTPS — Cloudflare terminates SSL for you
- Firewall rules, rate limiting

Use for: HTTP/HTTPS web traffic. A records and CNAME records for your website. Cloudflare Tunnel records (the tunnel itself handles routing).

Cannot be used for: MX records (email bypasses the proxy automatically), SSH, custom non-HTTP ports.
⚪ Grey Cloud — DNS Only
Cloudflare resolves DNS but traffic goes **directly to your server**. DNS returns your real IP address.

You get:

- Fast Cloudflare DNS (still faster than most registrars)
- Near-instant propagation
- No DDoS protection — your IP is visible
- No CDN or SSL termination at Cloudflare

Use for: Records where proxying breaks things — mail servers, FTP, non-standard ports, SSH, anything that doesn't use HTTP/HTTPS.

Always grey: NS and MX records are always grey — Cloudflare doesn't proxy these regardless of your setting.
ORANGE CLOUD (Proxied) GREY CLOUD (DNS Only) Browser → Cloudflare Edge Browser → (your real IP) │ │ │ Encrypted tunnel │ Direct connection ▼ ▼ Your Server Your Server dig osztromok.com → 104.21.x.x dig osztromok.com → 82.2.236.221 (Cloudflare's IP) (your real IP — visible) For Cloudflare Tunnel: orange cloud is correct — Cloudflare receives the request and routes it through the tunnel to your server.
For osztromok.com with Cloudflare Tunnel: the A or CNAME records for your website should be **orange cloud (proxied)**. The tunnel only works when traffic passes through Cloudflare — that's what makes it possible to receive connections despite the Virgin Media Hub blocking inbound ports. If you accidentally set the record to grey cloud, the tunnel won't work.

The Full Migration — Step by Step

Walkthrough · Chapter 3
Move DNS for osztromok.com from IONOS to Cloudflare. Estimated time: 15–30 minutes hands-on, plus up to 24 hours for NS propagation (usually much faster).
1
Create a free Cloudflare account at cloudflare.com if you don't have one. Log in and click **Add a site** (or **Add a domain** in newer UI). Enter `osztromok.com` and click Continue.
2
Select the Free plan — scroll to the bottom if you don't see it immediately. Cloudflare will show paid plans first; the free tier is fully adequate for self-hosting.
3
Cloudflare automatically scans your existing DNS records. It queries your current IONOS nameservers and imports what it finds. This usually takes 30–60 seconds. You'll then see a list of imported records to review — do not skip this step.
4
Review and correct the imported records carefully. Cloudflare's scan is good but not perfect. Check every record against what you know you have in IONOS:

— Is your A record for @ pointing to the right IP?
— Does www exist and point correctly?
— Are any MX records for email present?
— Are there any old records that no longer apply?
— Is the proxy toggle (orange/grey) set correctly for each? (A records for your website should be orange; MX records are always grey)

Add any missing records now. Delete any that shouldn't exist. You cannot easily go back once the nameservers switch.
5
Cloudflare will give you two nameserver addresses — something like:

elma.ns.cloudflare.com raj.ns.cloudflare.com # These are unique to your account — copy the exact ones Cloudflare gives you.
Copy both of these — you'll need them in the next step.
6
Log in to IONOS and update the nameservers for osztromok.com.

In IONOS: **Domains & SSL** → **click the domain name** → **Nameservers** → **Edit**. Remove the existing IONOS nameservers and replace them with the two Cloudflare addresses from step 5. Save.

This is the critical action — you're telling the global DNS system that Cloudflare is now authoritative for osztromok.com.
7
Wait for nameserver propagation. Cloudflare will email you when it detects the NS change has propagated. This typically takes **a few minutes to a few hours**, occasionally up to 24 hours. Your site remains accessible throughout — the old IONOS nameservers keep answering until the cached NS records expire worldwide.
8
Verify the transfer is complete:

Check which nameservers the world now sees for your domain $ dig osztromok.com NS +short elma.ns.cloudflare.com. raj.ns.cloudflare.com. # When both show Cloudflare nameservers, the transfer is complete. # Verify your A record resolves correctly through Cloudflare $ dig osztromok.com +short 104.21.x.x # Cloudflare's IP (proxied) — correct # Or if you set it to grey cloud (DNS only): 82.2.236.221 # your real IP — also correct if intentional
After Cloudflare confirms the transfer, you manage all DNS records inside the Cloudflare dashboard. The IONOS DNS panel is no longer used — records you change there will be ignored since IONOS's nameservers are no longer authoritative.

What to Check in the Imported Records

Cloudflare's auto-scan misses some record types and occasionally imports the wrong values. Use this as a checklist during step 4 of the migration:

Record	Proxy setting	What to verify
@ A record	Orange (proxied)	IP matches your server. Set orange if you want DDoS protection and CDN, grey if you need your real IP visible (e.g. for non-HTTP services on same IP).
www A or CNAME	Orange (proxied)	Points to your IP or to the root domain. Should match the @ record's proxy setting.
MX records	Always grey	Cloudflare forces MX to grey regardless — confirm the mail server hostname is correct. If you don't have email, you may not have any MX records.
TXT records	Always grey	SPF, DKIM, domain verification strings. Check they were all imported — TXT records are sometimes missed by the scan.
Cloudflare Tunnel CNAMEs	Orange (proxied)	If you already set these up in Cloudflare before the migration, they'll already be there. If not, Chapter 4 covers creating them.
Stale records	Remove	Old IP addresses, test subdomains, services you no longer use. Clean up now — it's easier than later.

CNAME Flattening — How Cloudflare Solves the Apex Problem

As covered in Chapter 1, the DNS specification doesn't allow a CNAME record at the root domain (`@`) because the root must also have NS and SOA records, and CNAME can't coexist with them. This is why adding a CNAME for `osztromok.com` pointing to `<tunnel-id>.cfargotunnel.com` failed in IONOS.

Cloudflare solves this internally with **CNAME flattening**: when you create a CNAME record at the root, Cloudflare doesn't serve it as a CNAME. Instead it follows the CNAME chain internally, resolves it to an A record (an IP address), and returns that A record to resolvers. From the outside it looks like a normal A record — fully spec-compliant. But you can use CNAME targets at the root.

You add in Cloudflare: osztromok.com. CNAME abc123.cfargotunnel.com. ← what you configure What resolvers see: osztromok.com. A 104.21.x.x ← what Cloudflare serves Cloudflare resolves abc123.cfargotunnel.com internally, finds its IP, and returns that as an A record. The CNAME never leaves Cloudflare's system.

This is why Cloudflare Tunnel can be configured to work on the root domain `osztromok.com` once DNS is managed by Cloudflare, even though it couldn't work when DNS was at IONOS.

What to Expect During the Transition

Immediately after saving nameservers in IONOS: Nothing changes yet. IONOS nameservers are still answering. Your site is still accessible normally.
0–2 hours: NS records start propagating. Resolvers around the world see a mix — some return IONOS's answers, some return Cloudflare's. Both sets of records are identical (you checked them in step 4) so visitors won't notice any difference.

2–24 hours: Propagation completes globally. Cloudflare sends you a confirmation email: "Great news! Cloudflare is now protecting your site." From this point, you manage DNS entirely in Cloudflare.
After confirmation: The IONOS DNS panel still shows your old records but they're now irrelevant — changes there have no effect. Manage everything in Cloudflare's dashboard.
SSL during transition: If your site was HTTP-only before, visitors may briefly see browser warnings as Cloudflare issues a certificate. This usually happens within 15 minutes of the transfer completing. If warnings persist, check the SSL/TLS setting in Cloudflare (Dashboard → SSL/TLS → Overview) — set it to **Flexible** initially (HTTPS to visitors, HTTP to your server) while you set up a proper certificate on the server (covered in the Security course, Chapter 1).

Common Mistakes

- **Not reviewing the auto-imported records.** Cloudflare's scan is good but misses records occasionally. Skipping the review and then discovering a missing MX record two days later when email stops working is a painful experience.
- **Setting MX records to orange cloud.** Email doesn't work through Cloudflare's HTTP proxy. Cloudflare automatically overrides this to grey, but it's worth knowing why — you can't proxy non-HTTP traffic through the orange cloud.
- **Continuing to edit records in IONOS after the transfer.** Once Cloudflare's nameservers are active, changes in IONOS's DNS panel are silently ignored. All record management moves to Cloudflare.
- **Forgetting that SSL/TLS mode matters.** If you set a record to orange cloud but leave SSL mode on "Off" in Cloudflare, visitors get HTTP despite HTTPS being possible. If you set it to "Full (Strict)" but don't have a valid certificate on your server, connections will fail. Start with "Flexible" and upgrade to "Full (Strict)" once you have Let's Encrypt set up (Security course, Chapters 1–2).
- **Panicking during propagation.** If your site seems to work for some visitors but not others, that's normal — different resolvers have different cached NS records. It resolves itself as propagation completes. Don't make further changes; you'll introduce inconsistencies.

Quick Reference — Chapter 3

Task / Command	Purpose / Notes
dig osztromok.com NS +short	Check which nameservers are authoritative — Cloudflare names confirm transfer is complete
dig osztromok.com +short	If proxied (orange), returns Cloudflare's IP. If DNS-only (grey), returns your real IP.
dig osztromok.com @elma.ns.cloudflare.com +short	Query Cloudflare's nameserver directly — confirms your records as Cloudflare sees them
Orange cloud (proxied)	Use for: A/CNAME records for HTTP/HTTPS websites, Cloudflare Tunnel CNAMEs
Grey cloud (DNS only)	Use for: MX (always), mail servers, FTP, SSH, any non-HTTP service
CNAME flattening	Cloudflare's automatic feature — lets you use CNAME at the root domain by resolving it internally to an A record
SSL/TLS → Flexible	Safe starting point after migration — HTTPS to visitors, HTTP to origin server. Upgrade to Full (Strict) after setting up Let's Encrypt.
Cloudflare dashboard → DNS	Where all record management happens after migration. IONOS DNS panel is no longer used.

WEB HOSTING ESSENTIALS — CHAPTER 4

Cloudflare Tunnel

Chapter 4 — Cloudflare Tunnel

Cloudflare Tunnel is the solution to the single biggest obstacle for home web hosting: the ISP blocking inbound connections. Instead of waiting for inbound traffic that the router will refuse, the tunnel reverses the connection — your server reaches out to Cloudflare, and Cloudflare forwards visitors' requests back through that outbound connection. No open ports. No port forwarding. Works behind Virgin Media, CGNAT, or any other restrictive ISP.

What this chapter covers: How Cloudflare Tunnel works architecturally — why outbound connections bypass ISP restrictions. Installing cloudflared on your server. The two setup methods: dashboard (GUI) and CLI. The config.yml file and ingress rules. Running the tunnel as a permanent systemd service. Adding multiple services through one tunnel. Troubleshooting common failure modes. This chapter assumes DNS is managed by Cloudflare (Chapter 3) — the tunnel cannot create CNAME records in IONOS.

How Cloudflare Tunnel Works

Traditional web hosting requires your router to accept *inbound* connections on port 80/443 and forward them to your server. ISPs like Virgin Media block these inbound connections — it's why every approach in this session's troubleshooting hit a wall.

Cloudflare Tunnel inverts the model. A small daemon called `cloudflared` runs on your server and establishes an *outbound* encrypted connection to Cloudflare's edge network. When a visitor requests `osztromok.com`, Cloudflare receives it at their edge, then forwards it *back through the existing outbound connection* to your server. Your server's response travels back the same way.

Traditional hosting (blocked by Virgin Media): Visitor ————[port 80]————▶ Router ──▶ Server ❌ Blocked Cloudflare Tunnel (works because it's outbound): Server ──outbound HTTPS──▶ Cloudflare Edge │ Visitor ──HTTPS──▶ Cloudflare Edge │ (routes through existing tunnel) │ ▼ Your Server Key insight: the server initiates the connection TO Cloudflare. ISPs and NAT/firewalls allow outbound connections freely. No port forwarding. No open inbound ports. No router config needed.

The tunnel uses **QUIC** (or falls back to HTTPS) for the connection between cloudflared and Cloudflare's edge. It maintains multiple persistent connections for redundancy. If cloudflared restarts, it reconnects automatically.

Prerequisites before starting this chapter:
1. DNS for osztromok.com is managed by Cloudflare (Chapter 3 complete)
2. You have a Cloudflare account with osztromok.com added
3. Apache is running on your server (already confirmed — port 80, `systemctl status apache2`)

Installing cloudflared

── Debian / Ubuntu (your server's OS) ──────────────────────────────── # Add Cloudflare's package repository $ curl -fsSL https://pkg.cloudflare.com/cloudflare-main.gpg | sudo tee /usr/share/keyrings/cloudflare-main.gpg > /dev/null $ echo "deb [signed-by=/usr/share/keyrings/cloudflare-main.gpg] https://pkg.cloudflare.com/cloudflared $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/cloudflared.list $ sudo apt update && sudo apt install cloudflared -y # Verify installation $ cloudflared --version cloudflared version 2024.x.x (built 2024-xx-xx)

Two Ways to Set Up a Tunnel

Dashboard Method (Recommended for beginners)
Configure everything through Cloudflare's web UI. The dashboard creates the tunnel, generates credentials, and creates DNS records automatically.

Pros:

- Guided — less chance of mistakes
- DNS records created automatically
- Easy to add/edit public hostnames later
- Tunnel visible and manageable in the UI

Cons:

- Requires browser access to Cloudflare
- Config stored in Cloudflare, not on your server

CLI Method (More control)
Create the tunnel via the cloudflared command line, write a local config.yml, run as a systemd service.

Pros:

- Config lives on your server — easy to back up
- More control over ingress rules
- Better for multiple services / complex routing
- Easier to version control

Cons:

- More steps — credentials, config file, DNS
- Need to create DNS records manually (or via CLI)

This chapter covers **both methods**. The dashboard method gets you running fastest; the CLI method gives you a setup that's easier to maintain long-term. Both end up with the same result.

Method 1 — Dashboard Setup

Setup Walkthrough · Dashboard Method
Create a tunnel through the Cloudflare Zero Trust dashboard and connect osztromok.com to Apache on your server.
1
Navigate to Zero Trust. In the Cloudflare dashboard, click **Zero Trust** in the left sidebar (or go to one.dash.cloudflare.com). If prompted to set up a team name, enter anything — it doesn't matter for tunnel use.
2
Create the tunnel. Go to **Networks** → **Tunnels** → **Create a tunnel**. Select **Cloudflared** as the connector type. Give the tunnel a name — something like `home-server`. Click Save tunnel.
3
Install the connector. Cloudflare will show you an install command. Select **Debian** as the OS. It will look like:

$ sudo cloudflared service install eyJhIjoiYWJjMTIzLi4u... # This long token contains your tunnel credentials. # Run this on your server — it installs cloudflared as a systemd service # and writes the credentials automatically. No separate config.yml needed. $ sudo systemctl start cloudflared $ sudo systemctl status cloudflared
After running this, the connector dot in the Cloudflare dashboard should turn green.
4
Add a public hostname. In the tunnel configuration, go to the **Public Hostname** tab and click **Add a public hostname**:

— **Subdomain:** leave blank (for root domain) *or* enter `www`
— **Domain:** select `osztromok.com`
— **Type:** HTTP
— **URL:** `localhost:80`

Click Save. Cloudflare automatically creates the DNS CNAME record for you.
5
Test it. Open a browser (or use curl from another machine) and visit `http://osztromok.com`. If Apache is running and the tunnel is healthy, you should see your site.

$ curl -I http://osztromok.com HTTP/1.1 200 OK Server: Apache/2.4.57 (Debian) CF-Ray: 7f3a... ← CF-Ray header confirms traffic went through Cloudflare
The dashboard method stores the tunnel token inside the systemd service file rather than a separate config.yml. This is fine for a single-service setup. If you later need multiple services (e.g. a separate app on port 8080), switch to the CLI method or add additional public hostnames through the dashboard.

Method 2 — CLI Setup with config.yml

Step 1 — Authenticate cloudflared with your Cloudflare account

$ cloudflared tunnel login Please open the following URL and log in with your Cloudflare account: https://dash.cloudflare.com/argotunnel?callback=... # Open this URL in a browser, log in, and select osztromok.com. # cloudflared saves a certificate to ~/.cloudflared/cert.pem You have successfully logged in. If you wish to copy your credentials to a server, they have been saved to: /home/philip/.cloudflared/cert.pem

Step 2 — Create the tunnel

$ cloudflared tunnel create home-server Tunnel credentials written to /home/philip/.cloudflared/abc12345-... -def6-7890-ghij-klmnopqrstuv.json. Created tunnel home-server with id abc12345-def6-7890-ghij-klmnopqrstuv # Note the tunnel ID — you'll need it in config.yml. # The .json file is the tunnel credential — keep it safe. # Move credentials to /etc/cloudflared/ so the system service can read them $ sudo mkdir -p /etc/cloudflared $ sudo cp ~/.cloudflared/abc12345-*.json /etc/cloudflared/ $ sudo chmod 600 /etc/cloudflared/abc12345-*.json # List tunnels to confirm it was created $ cloudflared tunnel list ID NAME CREATED STATUS abc12345-def6-7890-ghij-klmnopqrstuv home-server 2026-06-14T21:00:00Z inactive

Step 3 — Create the config.yml file

/etc/cloudflared/config.yml tunnel: abc12345-def6-7890-ghij-klmnopqrstuv # your tunnel ID credentials-file: /etc/cloudflared/abc12345-def6-7890-ghij-klmnopqrstuv.json ingress: # Route the root domain to Apache - hostname: osztromok.com service: http://localhost:80 # Route www to the same place - hostname: www.osztromok.com service: http://localhost:80 # Catch-all rule — REQUIRED, must be last # Any request that doesn't match a hostname above gets a 404 - service: http_status:404

The catch-all rule is mandatory. config.yml must end with an ingress rule that has no hostname — it catches requests that don't match any other rule. Without it, cloudflared will refuse to start. The value `http_status:404` returns a 404 for unmatched requests, which is the correct behaviour.

Step 4 — Create DNS records for the tunnel

Tell Cloudflare to create a CNAME DNS record pointing the hostname # to this tunnel. Run once for each hostname in your ingress rules. $ cloudflared tunnel route dns home-server osztromok.com 2026/06/14 21:05:00 Added CNAME osztromok.com which will route to this tunnel tunnelID=abc12345... $ cloudflared tunnel route dns home-server www.osztromok.com 2026/06/14 21:05:01 Added CNAME www.osztromok.com which will route to this tunnel tunnelID=abc12345... # Verify the records were created in Cloudflare DNS $ dig osztromok.com +short abc12345-def6-7890-ghij-klmnopqrstuv.cfargotunnel.com. # CNAME (Cloudflare flattens this)

Step 5 — Test the tunnel before installing as a service

Run in the foreground first — you'll see connection logs and can confirm it works $ cloudflared tunnel --config /etc/cloudflared/config.yml run 2026/06/14 21:06:00 Starting metrics server on 127.0.0.1:... 2026/06/14 21:06:00 Registered tunnel connection connIndex=0 ip=198.41.200.13 2026/06/14 21:06:00 Registered tunnel connection connIndex=1 ip=198.41.192.47 2026/06/14 21:06:00 Registered tunnel connection connIndex=2 ip=198.41.200.13 2026/06/14 21:06:00 Registered tunnel connection connIndex=3 ip=198.41.192.47 # 4 connections registered = healthy tunnel. Open a browser and test now. # Press Ctrl+C to stop once confirmed working.

Step 6 — Install as a systemd service

Install cloudflared as a system service (reads /etc/cloudflared/config.yml) $ sudo cloudflared --config /etc/cloudflared/config.yml service install 2026/06/14 21:10:00 Installing cloudflared client as a systemd service 2026/06/14 21:10:00 cloudflared client was successfully installed as a systemd service # Enable and start the service $ sudo systemctl enable cloudflared $ sudo systemctl start cloudflared $ sudo systemctl status cloudflared ● cloudflared.service - cloudflared Loaded: loaded (/etc/systemd/system/cloudflared.service) Active: active (running) since Sun 2026-06-14 21:10:05 BST; 5s ago # "active (running)" = tunnel is up and will restart automatically on reboot

Routing Multiple Services Through One Tunnel

A single tunnel can serve multiple domains and subdomains to different backend services. You don't need a separate tunnel for each service — just add more ingress rules to config.yml and create the corresponding DNS records.

Example: Multiple services on one tunnel
osztromok.com
→
http://localhost:80
Main Apache site

www.osztromok.com
→
http://localhost:80
Same as root
blog.osztromok.com
→
http://localhost:2368
Ghost blog on port 2368
grafana.osztromok.com
→
http://localhost:3000
Grafana dashboard
(catch-all)
→
http_status:404
Required — must be last
/etc/cloudflared/config.yml — multiple services example tunnel: abc12345-def6-7890-ghij-klmnopqrstuv credentials-file: /etc/cloudflared/abc12345-def6-7890-ghij-klmnopqrstuv.json ingress: - hostname: osztromok.com service: http://localhost:80 - hostname: www.osztromok.com service: http://localhost:80 - hostname: blog.osztromok.com service: http://localhost:2368 - hostname: grafana.osztromok.com service: http://localhost:3000 - service: http_status:404
After editing config.yml, reload the service $ sudo systemctl restart cloudflared # Create DNS records for any new hostnames you added $ cloudflared tunnel route dns home-server blog.osztromok.com $ cloudflared tunnel route dns home-server grafana.osztromok.com

Troubleshooting

Check the service status $ sudo systemctl status cloudflared # Watch live logs $ sudo journalctl -u cloudflared -f # Check the tunnel is registered in Cloudflare $ cloudflared tunnel info home-server NAME: home-server ID: abc12345-... CONNECTIONS: 4 active ← healthy: 4 connections to Cloudflare edge # Validate config.yml syntax before restarting $ cloudflared tunnel --config /etc/cloudflared/config.yml ingress validate Validating rules from /etc/cloudflared/config.yml OK
Service starts but website shows "ERR_CONNECTION_REFUSED" or "522" error
Apache isn't running or isn't listening on port 80. Check: `systemctl status apache2` and `ss -tlnp | grep :80`. The tunnel forwards traffic to `localhost:80` — if nothing is there to receive it, you get a connection error.
Tunnel is "active" in systemctl but website doesn't load — just times out
DNS record not pointing to the tunnel. Check: `dig osztromok.com +short` — should return a Cloudflare IP or the tunnel CNAME. If it still returns your home IP (82.2.236.221), the CNAME record wasn't created. Run `cloudflared tunnel route dns home-server osztromok.com` and check the Cloudflare DNS dashboard.
Cloudflare shows "526 Invalid SSL certificate" or "525 SSL handshake failed"
SSL/TLS mode mismatch. In Cloudflare dashboard → SSL/TLS → Overview, check the mode. **Flexible** = Cloudflare speaks HTTPS to visitors, HTTP to your server (no server cert needed). **Full** = requires HTTPS on your server. **Full (Strict)** = requires a valid signed certificate on your server. If you haven't set up Let's Encrypt yet, use Flexible for now.
cloudflared service won't start — "no such file or directory" for credentials
The credentials JSON file path in config.yml doesn't match the actual file. Check: `ls /etc/cloudflared/` and verify the filename matches what's in the `credentials-file:` line exactly, including the full UUID.
"config.yml: ingress: rule #X has no service" — tunnel refuses to start
The catch-all rule at the end of ingress is missing or malformed. It must be the last rule and must have no `hostname:` key — just `- service: http_status:404`. Check indentation too — YAML is whitespace-sensitive; use spaces, never tabs.
Tunnel was working, stopped after router reset / reboot
The service may not be enabled to start on boot. Check: `systemctl is-enabled cloudflared`. If it says `disabled`, run `sudo systemctl enable cloudflared`. The tunnel itself doesn't depend on port forwarding — it should survive router changes as long as outbound internet access is available.

Quick Reference — Chapter 4

Command	Purpose
cloudflared tunnel login	Authenticate with your Cloudflare account — required for CLI method
cloudflared tunnel create NAME	Create a new named tunnel and generate credentials JSON

Command	Purpose
cloudflared tunnel list	List all tunnels on your account and their status
cloudflared tunnel route dns NAME host	Create a CNAME DNS record in Cloudflare pointing a hostname to this tunnel
cloudflared tunnel --config /etc/cloudflared/config.yml run	Run the tunnel in the foreground for testing — Ctrl+C to stop
cloudflared tunnel ingress validate	Check config.yml for syntax errors before restarting the service
cloudflared tunnel info NAME	Show tunnel details and number of active connections to Cloudflare edge
sudo cloudflared service install	Install cloudflared as a systemd service (reads /etc/cloudflared/config.yml)
sudo systemctl enable cloudflared	Ensure tunnel starts automatically on every boot
sudo journalctl -u cloudflared -f	Watch cloudflared logs live — first place to look when troubleshooting

File / Location	Purpose
/etc/cloudflared/config.yml	Main tunnel config — tunnel ID, credentials path, ingress rules
/etc/cloudflared/<uuid>.json	Tunnel credentials — treat like a password, don't share or commit to git
~/.cloudflared/cert.pem	Account-level certificate from cloudflared tunnel login

WEB HOSTING ESSENTIALS — CHAPTER 5

Apache Virtual Hosts

Chapter 5 — Apache Virtual Hosts

A single Apache server can host dozens of websites simultaneously. Virtual hosts let Apache serve different content depending on which domain name the visitor requested — all on the same IP address, the same port 80, the same machine. This chapter covers everything from the configuration file anatomy to enabling/disabling sites, setting up directory permissions, and understanding how virtual hosts interact with a Cloudflare Tunnel.

What this chapter covers: Name-based vs IP-based virtual hosts. Apache's sites-available/sites-enabled architecture. Every directive in a vhost config block. Creating document root directories with correct permissions. Enabling and disabling sites with a2ensite/a2dissite. The default vhost catch-all. How Cloudflare Tunnel interacts with vhosts. Troubleshooting: 403 Forbidden, 404 Not Found, and wrong site served.

Name-Based vs IP-Based Virtual Hosts

Name-Based (what you'll use)
Apache reads the **Host header** in the HTTP request to decide which vhost to serve. Multiple domains share the same IP address.

- One IP serves unlimited domains
- Works on port 80 and 443
- The standard for virtually all hosting
- Config: **VirtualHost *:80**

Requirement: client must send a Host header — all modern browsers and HTTP/1.1+ clients do.
IP-Based (rare, legacy)
Each domain gets a dedicated IP address. Apache matches on the IP the request arrived on, not the hostname.

- Requires multiple IPs or network interfaces
- Needed for TLS before SNI existed (pre-2012)
- Almost never used today
- Config: **VirtualHost 192.168.1.10:80**

Not covered here — name-based vhosts are the correct approach for your setup.

When a request arrives, Apache goes through the enabled vhosts in order and picks the first one whose `ServerName` or `ServerAlias` matches the Host header. If no vhost matches, Apache serves the first enabled vhost alphabetically — this is the "default" behaviour covered later.

HTTP request arrives at Apache (port 80): GET /index.html HTTP/1.1 Host: blog.osztromok.com ← Apache reads this header Apache checks enabled vhosts in order: ┌──┐ │ VirtualHost 1: ServerName osztromok.com │ ✗ no match │ VirtualHost 2: ServerName blog.osztromok.com │ ✓ match → serve from /var/www/blog │ VirtualHost 3: ServerName shop.osztromok.com │ (not checked — match already found) └──┘

Apache's Site Configuration Structure

Apache on Debian/Ubuntu uses a clean two-directory pattern for managing vhosts. You write configs in `sites-available` and use `a2ensite` to activate them — it simply creates a symlink into `sites-enabled`. Apache only reads `sites-enabled` at startup.

/etc/apache2/ ├── sites-available/ ← write your vhost configs here │ ├── 000-default.conf ← the catch-all default (ships with Apache) │ ├── osztromok.com.conf ← your main site │ └── blog.osztromok.com.conf ← second site │ ├── sites-enabled/ ← Apache reads ONLY these at startup (symlinks) │ ├── 000-default.conf → ../sites-available/000-default.conf │ ├── osztromok.com.conf → ../sites-available/osztromok.com.conf │ └── blog.osztromok.com.conf → ../sites-available/blog.osztromok.com.conf │ ├── mods-available/ mods-enabled/ ← same pattern for modules └── conf-available/ conf-enabled/ ← same pattern for global config snippets
Why this pattern? You can write a config and leave it inactive (in sites-available but not enabled). To disable a site without deleting its config, just run `sudo a2dissite sitename.conf`. The config stays in sites-available for later use.

Key commands

$ sudo a2ensite osztromok.com.conf # enable (creates symlink in sites-enabled) $ sudo a2dissite osztromok.com.conf # disable (removes symlink) $ sudo apache2ctl configtest # validate all config before reloading Syntax OK $ sudo systemctl reload apache2 # graceful reload — no dropped connections # Use reload (not restart) when only changing config. # restart kills and recreates the process; reload just re-reads config.

Anatomy of a Virtual Host Config

A complete, production-ready vhost config for osztromok.com:

/etc/apache2/sites-available/osztromok.com.conf <VirtualHost *:80> # ── Identity ── ServerName osztromok.com ServerAlias www.osztromok.com ServerAdmin emubantam@gmail.com # ── Where your files live ─── DocumentRoot /var/www/osztromok.com/public_html # ── Permissions for the document root ─────────────────────────────── <Directory /var/www/osztromok.com/public_html> Options Indexes FollowSymLinks AllowOverride All Require all granted </Directory> # ── Per-site logs ─── ErrorLog ${APACHE_LOG_DIR}/osztromok_error.log CustomLog ${APACHE_LOG_DIR}/osztromok_access.log combined </VirtualHost>

VirtualHost *:80

Listen on **all interfaces** (*) on port 80. If you have multiple IPs and want to restrict to one, replace * with the IP address. Almost always use *.

ServerName

The **primary hostname** this vhost responds to. Apache matches this against the Host header in the request. Required — without it Apache guesses based on the server hostname.

ServerAlias

Additional hostnames that map to this same vhost. Space-separated. Use for www, other subdomains, or alternative domains. Wildcards allowed: `*.osztromok.com`.

ServerAdmin

Email address shown in Apache error pages. Optional but useful — helps identify which site an error belongs to.

DocumentRoot

The **filesystem path** where Apache looks for files to serve. A request for `/index.html` maps to `DocumentRoot/index.html`. The directory must exist and be readable by the Apache user (`www-data`).

Directory block

Sets **permissions and behaviour** for files in that path. Required — without it, Apache refuses to serve the files even if they exist.

Options Indexes FollowSymLinks

`Indexes`: show a file listing if no index.html exists (useful during dev, disable in prod). `FollowSymLinks`: required for mod_rewrite to work. `None` disables all options.

AllowOverride All

Permits **.htaccess files** in this directory to override config (URL rewriting, auth, redirects). `None` ignores .htaccess entirely — faster but less flexible. Use `All` if the site uses WordPress, Laravel, or any framework with .htaccess rewrites.

Require all granted

Apache 2.4 access control — **allow all requests** to this directory. Without this, Apache returns 403 Forbidden regardless of file permissions.

ErrorLog / CustomLog

Per-site log files. `${APACHE_LOG_DIR}` expands to `/var/log/apache2/`. Separate logs per vhost make debugging vastly easier — you can tail just the log for the broken site.

Creating the Document Root and Permissions

The DocumentRoot directory must exist and be readable by the Apache process user (`www-data`). The simplest approach: put your files under `/var/www/` and set the owner to your own user with `www-data` as the group.

Create the directory structure $ sudo mkdir -p /var/www/osztromok.com/public_html # Set ownership: your user owns the files, www-data is the group $ sudo chown -R philip:www-data /var/www/osztromok.com # Permissions: # 755 = directories: owner rwx, group+others r-x (readable, traversable) # 644 = files: owner rw-, group+others r-- (readable, not executable) $ sudo find /var/www/osztromok.com -type d -exec chmod 755 {} \; $ sudo find /var/www/osztromok.com -type f -exec chmod 644 {} \; # Create a test index page $ echo '<h1>osztromok.com — virtual host working</h1>' | sudo tee /var/www/osztromok.com/public_html/index.html

Avoid chmod 777. Making directories world-writable lets any process on the server write to your web files — a significant security risk. Use 755 for directories and 644 for files. Apache (running as `www-data`) can read 644 files because it has execute permission on 755 directories.

If you're uploading files via SSH (scp / rsync)

After uploading as your user (philip), fix group ownership so Apache can read $ sudo chown -R philip:www-data /var/www/osztromok.com/public_html $ sudo find /var/www/osztromok.com/public_html -type f -exec chmod 644 {} \;

Scenario — Adding a Second Website to the Server

Setup Walkthrough · Complete
Add blog.osztromok.com as a separate website on the same server — different document root, different logs.
1
Create the document root.

$ sudo mkdir -p /var/www/blog.osztromok.com/public_html $ sudo chown -R philip:www-data /var/www/blog.osztromok.com $ sudo find /var/www/blog.osztromok.com -type d -exec chmod 755 {} \; $ echo '<h1>blog.osztromok.com</h1>' | sudo tee /var/www/blog.osztromok.com/public_html/index.html
2
Write the vhost config file.

$ sudo nano /etc/apache2/sites-available/blog.osztromok.com.conf
Enter this content:

<VirtualHost *:80> ServerName blog.osztromok.com DocumentRoot /var/www/blog.osztromok.com/public_html <Directory /var/www/blog.osztromok.com/public_html> Options Indexes FollowSymLinks AllowOverride All Require all granted </Directory> ErrorLog ${APACHE_LOG_DIR}/blog_error.log CustomLog ${APACHE_LOG_DIR}/blog_access.log combined </VirtualHost>
3
Enable the site, test config, reload.

$ sudo a2ensite blog.osztromok.com.conf Enabling site blog.osztromok.com. To activate the new configuration, you need to run: service apache2 reload $ sudo apache2ctl configtest Syntax OK $ sudo systemctl reload apache2
4
Create the DNS record for the subdomain. In Cloudflare's DNS dashboard (since DNS moved to Cloudflare in Chapter 3), add:

— **Type:** CNAME | **Name:** blog | **Target:** `osztromok.com` | **Proxy:** orange cloud (proxied)

Or if you're using Cloudflare Tunnel: add `blog.osztromok.com` as an additional public hostname in the tunnel config (Chapter 4). The tunnel will forward requests to `localhost:80` with the original Host header intact, and Apache's ServerName matching will route them to the blog vhost.
5
Test from outside.

$ curl -H "Host: blog.osztromok.com" http://localhost <h1>blog.osztromok.com</h1> ← correct vhost served $ curl -H "Host: osztromok.com" http://localhost <h1>osztromok.com — virtual host working</h1> ← different vhost
Passing the `-H "Host:"` header lets you test vhost routing locally without needing public DNS.
Each site has its own document root (/var/www/osztromok.com and /var/www/blog.osztromok.com), its own config file, and its own log files. They're completely independent — changing one doesn't affect the other.

The Default Virtual Host

When no enabled vhost matches the Host header, Apache falls back to the **first vhost alphabetically in sites-enabled**. On a fresh Debian/Ubuntu install, that's `000-default.conf` — the `000` prefix puts it first.

/etc/apache2/sites-available/000-default.conf (shipped with Apache) <VirtualHost *:80> ServerAdmin webmaster@localhost DocumentRoot /var/www/html # serves the Apache "It works!" page ErrorLog ${APACHE_LOG_DIR}/error.log CustomLog ${APACHE_LOG_DIR}/access.log combined </VirtualHost>

This vhost has *no ServerName* — it never matches any specific request. Its role is purely to be the catch-all fallback. Common approaches:

- **Leave it enabled** — requests with unknown hostnames (e.g. someone hitting your IP directly) see the Apache default page. Fine for most cases.
- **Return 444 / 403 for unmatched requests** — add a catch-all vhost that returns an error instead of leaking your default site.
- **Disable it** — run `sudo a2dissite 000-default.conf`. The first enabled named vhost becomes the fallback. Acceptable if you're fine with any unmatched request seeing your main site.

The "ServerName" warning on Apache startup — if you see `AH00558: apache2: Could not reliably determine the server's fully qualified domain name`, add `ServerName localhost` to `/etc/apache2/apache2.conf`. This is a cosmetic warning, not an error — your vhosts work fine without it.
Fix the AH00558 ServerName warning $ echo "ServerName localhost" | sudo tee -a /etc/apache2/apache2.conf $ sudo apache2ctl configtest && sudo systemctl reload apache2

Cloudflare Tunnel + Virtual Hosts

When traffic arrives at your server via Cloudflare Tunnel, Apache's vhost routing works exactly as if the client connected directly — because Cloudflare preserves the original `Host` header when forwarding the request through the tunnel to `localhost:80`.

Request flow with Cloudflare Tunnel + Apache vhosts: Visitor requests blog.osztromok.com │ ▼ Cloudflare edge receives it (Host: blog.osztromok.com) │ ▼ forwards through tunnel, preserving Host header cloudflared daemon on your server │ ▼ HTTP to localhost:80 with Host: blog.osztromok.com Apache │ ▼ checks ServerName of each enabled vhost blog.osztromok.com.conf → DocumentRoot /var/www/blog/public_html │ ▼ Response flows back through the same path to the visitor

The key point: when you add a new vhost for `newsite.osztromok.com`, you need to:

1. Create the vhost config in Apache (this chapter)
2. Create the DNS record in Cloudflare *or* add a public hostname to the tunnel config (Chapter 4)

Apache handles *which* content to serve. The tunnel handles *how traffic gets to the server*. They're independent layers.

Testing vhosts locally (bypassing the tunnel): Use `curl -H "Host: blog.osztromok.com" http://localhost` on the server itself. This hits Apache directly without going through the tunnel and is the fastest way to verify a new vhost is working before touching DNS.

Troubleshooting

Print all enabled vhosts and which config file each came from $ sudo apache2ctl -S VirtualHost configuration: *:80 osztromok.com (/etc/apache2/sites-enabled/osztromok.com.conf:1) *:80 blog.osztromok.com (/etc/apache2/sites-enabled/blog.osztromok.com.conf:1) # If a vhost isn't listed, it's not enabled — check a2ensite or symlink # Watch live error log for the affected site $ sudo tail -f /var/log/apache2/blog_error.log # Watch access log to see what Apache is actually receiving $ sudo tail -f /var/log/apache2/blog_access.log
403 Forbidden — Apache returns 403 when trying to load the site
Usually a **permissions problem**. Apache (running as www-data) can't read the files or traverse the directory.
Check 1 — directory permissions: `ls -la /var/www/yourdomain.com/` — parent directories need execute (x) bit for www-data.
Check 2 — `Require all granted` missing from the Directory block in the vhost config. Without it, Apache denies all access regardless of file permissions.
Check 3 — `Options Indexes` is off and there's no index.html. Apache refuses to list the directory. Add an index.html or add `Options Indexes`.
Fix: `sudo chown -R philip:www-data /var/www/yourdomain.com && sudo chmod -R 755 /var/www/yourdomain.com`
404 Not Found — site loads but pages return 404
File not found in the DocumentRoot — either the path in DocumentRoot is wrong or the file doesn't exist there. Check: `ls /var/www/yourdomain.com/public_html/`. Also check the error log: it shows the exact path Apache was looking for. If using URL rewriting (.htaccess), ensure `AllowOverride All` is set and mod_rewrite is enabled (`sudo a2enmod rewrite`).
Wrong site is served — Apache serves the wrong vhost's content
Run `sudo apache2ctl -S` to see the vhost list order. The wrong site being served usually means: (1) the correct vhost's `ServerName` doesn't match the incoming Host header exactly, (2) the correct vhost isn't enabled, or (3) there's a typo in ServerName/ServerAlias. Test with `curl -H "Host: exact.domain.name" http://localhost` to confirm what Apache receives.
Config change has no effect after reload
Always run `sudo apache2ctl configtest` before reload — if the config has a syntax error, `systemctl reload apache2` fails silently and the old config stays active. The configtest output will show the exact line and error. Fix the syntax, re-run configtest, then reload.
New vhost works locally (curl localhost) but not from outside
Apache is configured correctly but the DNS record or tunnel routing isn't set up. Check: (1) DNS — `dig blog.osztromok.com` should resolve to a Cloudflare IP. (2) If using Cloudflare Tunnel, confirm the new hostname is in the tunnel's ingress rules (config.yml or dashboard public hostname tab). (3) Run `sudo systemctl status cloudflared` to check the tunnel is running.

Quick Reference — Chapter 5

Command	Purpose
sudo a2ensite site.conf	Enable a vhost — creates symlink from sites-available to sites-enabled
sudo a2dissite site.conf	Disable a vhost — removes the symlink (config file stays in sites-available)
sudo apache2ctl configtest	Validate all Apache config files — always run before reloading
sudo apache2ctl -S	List all enabled vhosts with their config file paths and port/hostname
sudo systemctl reload apache2	Gracefully reload config — no dropped connections; use this over restart
sudo a2enmod rewrite	Enable mod_rewrite — required for .htaccess URL rewriting (WordPress etc.)
curl -H "Host: x.com" http://localhost	Test a specific vhost locally by spoofing the Host header
sudo tail -f /var/log/apache2/NAME_error.log	Watch live errors for a specific vhost

Directive	Required?	Purpose
ServerName	Yes	Primary hostname for this vhost — must match Host header
ServerAlias	Optional	Additional hostnames (www, subdomains) — space-separated
DocumentRoot	Yes	Filesystem path to serve files from
Directory block	Yes	Permissions for the document root — without it you get 403
Require all granted	Yes	Apache 2.4 access control — allows public access to the directory
AllowOverride All	For .htaccess	Enables .htaccess overrides — needed for WordPress, Laravel etc.
ErrorLog / CustomLog	Optional	Per-site logging — highly recommended for any multi-vhost setup

Path	Purpose
/etc/apache2/sites-available/	Write vhost configs here — Apache doesn't read this directly
/etc/apache2/sites-enabled/	Symlinks to active vhosts — Apache reads only these at startup
/var/www/yourdomain.com/public_html/	Conventional document root location for each site
/var/log/apache2/	Apache log directory — ${APACHE_LOG_DIR} in config expands to this

WEB HOSTING ESSENTIALS — CHAPTER 6

Subdomains End-to-End

Chapter 6 — Subdomains End-to-End

Previous chapters covered DNS, Cloudflare, tunnels, and virtual hosts as separate subjects. This chapter pulls them together: a subdomain only works when both the DNS layer and the server layer are configured correctly. Miss either one and the subdomain either doesn't resolve or resolves to the wrong place. This chapter is a practical integration guide — complete walkthroughs, common patterns, and the exact sequence to follow every time you add a new subdomain.

What this chapter covers: The two-part requirement and why both parts must align. Complete end-to-end walkthrough for a static subdomain. Redirect www to the root domain (two methods). Wildcard subdomains in DNS and Apache. Reverse proxying to a backend application running on a different port. Subdomain deployment checklist. Troubleshooting: subdomain not resolving, SSL errors, serving wrong content.

The Two-Part Requirement

A subdomain like `blog.osztromok.com` requires two completely separate systems to agree with each other. If either is missing, the subdomain fails — and the error messages are different depending on which part is absent.

1
DNS — tells the internet where to send traffic
A DNS record in Cloudflare pointing `blog.osztromok.com` to Cloudflare's edge (via the tunnel CNAME or a direct A record).

What goes wrong without it: Browser shows "This site can't be reached / DNS_PROBE_FINISHED_NXDOMAIN" — the domain doesn't resolve at all. `dig blog.osztromok.com` returns NXDOMAIN (no record found).
2
Server — tells Apache what to serve when traffic arrives
An Apache virtual host with `ServerName blog.osztromok.com` pointing to the correct document root (or ProxyPass to a backend).

What goes wrong without it: DNS resolves correctly, browser reaches Apache, but Apache serves the *default vhost* (or returns 403/404). The right domain, the wrong content.
Complete path — everything must be in place:
┌──┐ │ Part 1: DNS layer │ │ blog.osztromok.com ─CNAME→ tunnel.cfargotunnel.com │ │ (Cloudflare DNS panel, or auto-created by cloudflared) │ └──────────────────────────┬───┘ │ resolves → Cloudflare edge ▼ ┌──┐ │ Cloudflare → tunnel → cloudflared → localhost:80 │ │ Host header: blog.osztromok.com (preserved through tunnel) │ └──────────────────────────┬───┘ │ arrives at Apache ▼ ┌──┐ │ Part 2: Server layer │ │ VirtualHost *:80 { ServerName blog.osztromok.com } │ │ DocumentRoot /var/www/blog.osztromok.com/public_html │ │ (/etc/apache2/sites-enabled/blog.osztromok.com.conf) │ └──┘

Common Subdomain Patterns

www.osztromok.com
www alias
Traditional alias for the root domain. Usually configured to redirect to the apex — most sites pick one canonical form and redirect the other.
blog.osztromok.com
content site
Separate document root, can run a completely different CMS (WordPress, Ghost) or just static HTML. Independent Apache vhost.
api.osztromok.com
backend app
ProxyPass to a backend (Flask, FastAPI, Node) running on a different port. Apache acts as a reverse proxy — the app never knows about subdomain routing.
dev.osztromok.com
staging / dev

Separate document root with a work-in-progress version of the site. Optional: restrict access by IP with a Require ip directive.
*.osztromok.com
wildcard
One DNS record matches all subdomains. Combined with a wildcard Apache ServerAlias, useful for multi-tenant apps or catching unknown subdomains.
status.osztromok.com
monitoring
Uptime / status page (Uptime Kuma, Cachet). Runs on its own port, served via ProxyPass. Can be hosted separately so it stays up when the main site is down.

Scenario 1 — Static Subdomain End-to-End

Full Walkthrough · blog.osztromok.com
Create a working blog subdomain serving static files — from empty directory to publicly accessible URL, with Cloudflare Tunnel.
1
Create the document root and a test page.

$ sudo mkdir -p /var/www/blog.osztromok.com/public_html $ sudo chown -R philip:www-data /var/www/blog.osztromok.com $ sudo find /var/www/blog.osztromok.com -type d -exec chmod 755 {} \; $ cat <<'EOF' | sudo tee /var/www/blog.osztromok.com/public_html/index.html <!DOCTYPE html> <html><body><h1>blog.osztromok.com</h1> Subdomain working. </body></html> EOF
2
Write the Apache vhost config.

$ sudo nano /etc/apache2/sites-available/blog.osztromok.com.conf

<VirtualHost *:80> ServerName blog.osztromok.com DocumentRoot /var/www/blog.osztromok.com/public_html <Directory /var/www/blog.osztromok.com/public_html> Options FollowSymLinks AllowOverride All Require all granted </Directory> ErrorLog ${APACHE_LOG_DIR}/blog_error.log CustomLog ${APACHE_LOG_DIR}/blog_access.log combined </VirtualHost>
3
Enable, test config, reload Apache.

$ sudo a2ensite blog.osztromok.com.conf $ sudo apache2ctl configtest Syntax OK $ sudo systemctl reload apache2
4
Test locally — verify Apache routing before touching DNS. This is the most valuable debugging step: it confirms Apache is working independently of DNS and the tunnel.

$ curl -H "Host: blog.osztromok.com" http://localhost <!DOCTYPE html> <html><body><h1>blog.osztromok.com</h1> Subdomain working. </body></html> # Correct content = Apache vhost is working. Proceed to DNS. # Also confirm the main site still works (no regression) $ curl -H "Host: osztromok.com" http://localhost | head -3
5
Add the DNS record in Cloudflare. Go to your Cloudflare DNS dashboard for osztromok.com and add:

Type: CNAME | **Name:** `blog` | **Target:** `osztromok.com` | **Proxy:** orange cloud (proxied)

If using Cloudflare Tunnel: instead of a DNS record, go to Zero Trust → Networks → Tunnels → your tunnel → Public Hostnames → Add a public hostname:
Subdomain: `blog` · Domain: `osztromok.com` · Service Type: HTTP · URL: `localhost:80`
The tunnel creates the CNAME automatically and routes traffic through. No separate DNS step needed.
6
Verify DNS propagation and test from outside.

$ dig blog.osztromok.com +short 104.21.x.x ← Cloudflare IP = DNS record is live and proxied # From another machine / your phone on mobile data: $ curl https://blog.osztromok.com <h1>blog.osztromok.com</h1> ← subdomain working end-to-end
If DNS isn't resolving yet, wait 1–2 minutes (Cloudflare propagates quickly) and try again.
Always test locally with curl before touching DNS — it separates Apache problems from DNS problems. If curl localhost works but the public URL doesn't, the problem is in DNS or the tunnel, not Apache.

Scenario 2 — Redirecting www to the Root Domain

Most sites pick one canonical form — either `www.osztromok.com` or `osztromok.com` — and redirect the other. This matters for SEO (duplicate content) and for user expectations. The standard modern approach is to use the apex (`osztromok.com`) as canonical and redirect `www` to it.

Method A — Apache redirect vhost (server-side)

/etc/apache2/sites-available/www-redirect.conf # A dedicated vhost whose only job is to redirect www → apex <VirtualHost *:80> ServerName www.osztromok.com # Permanent redirect (301) — browsers and search engines remember this Redirect permanent / https://osztromok.com/ </VirtualHost>
$ sudo a2ensite www-redirect.conf $ sudo apache2ctl configtest && sudo systemctl reload apache2 # Test the redirect (follow the redirect with -L, show headers with -I) $ curl -I -H "Host: www.osztromok.com" http://localhost HTTP/1.1 301 Moved Permanently Location: https://osztromok.com/ ← correct destination
Don't forget the DNS record for www. You need a CNAME for `www` pointing to `osztromok.com` in Cloudflare, otherwise the redirect vhost is never reached. Apache can only redirect a request that actually arrives.

Method B — Cloudflare Redirect Rules (no Apache config needed)

If DNS is on Cloudflare, you can handle the www redirect entirely at the Cloudflare edge — the request never reaches your server. Go to: **Rules → Redirect Rules → Create rule**

- **When:** Hostname equals `www.osztromok.com`
- **Then:** Static redirect → `https://osztromok.com/` (301 Permanent)

This is faster (redirect happens at Cloudflare's edge) and requires no Apache changes. Either method works — use the Cloudflare approach if you're already comfortable there, or the Apache approach if you want all routing logic on the server.

Wildcard Subdomains

A wildcard matches any subdomain that doesn't have a more specific record. It's one DNS record that covers `anything.osztromok.com`.

Wildcard DNS record in Cloudflare

Type	Name	Target	Proxy
CNAME	*	osztromok.com	Orange cloud (proxied)

This makes `anything.osztromok.com` resolve to Cloudflare's edge. Traffic still needs somewhere to land on the server.

Wildcards only match one level. `*.osztromok.com` matches `foo.osztromok.com` but **not** `foo.bar.osztromok.com`. If you need deeper subdomains, create explicit records for each.

Wildcard Apache vhost (catch-all for unknown subdomains)

/etc/apache2/sites-available/wildcard.osztromok.com.conf # Serves a generic page for any subdomain without its own vhost <VirtualHost *:80> ServerName wildcard.osztromok.com ServerAlias *.osztromok.com # Use the server name in the document root so each subdomain # can have different content if you create matching directories DocumentRoot /var/www/html # or a generic "subdomain not configured" page <Directory /var/www/html> Options FollowSymLinks AllowOverride None Require all granted </Directory> ErrorLog ${APACHE_LOG_DIR}/wildcard_error.log CustomLog ${APACHE_LOG_DIR}/wildcard_access.log combined </VirtualHost>
Order matters. The wildcard vhost should load *after* all specific vhosts alphabetically (or use a `z-` prefix in the filename). Specific vhosts — `blog.osztromok.com.conf`, `shop.osztromok.com.conf` — match first. The wildcard only catches subdomains with no explicit vhost.

Scenario 3 — Reverse Proxy to a Backend App

When you have an application (Python/FastAPI, Node.js, anything) running on a local port (e.g. 8080), Apache can act as a reverse proxy: it accepts the subdomain request and forwards it to the app, then passes the response back to the visitor. The app doesn't need to know about Cloudflare or subdomains at all.

Reverse proxy flow: Visitor → https://api.osztromok.com │ ▼ Cloudflare Tunnel → Apache *:80, Host: api.osztromok.com │ ▼ Apache matches api.osztromok.com vhost → ProxyPass │ ▼ Forwards internally to http://localhost:8080 │ ▼ FastAPI / Node / Flask running on port 8080 │ Response flows back: app → Apache → Cloudflare → Visitor
Enable the required Apache modules (one-time setup) $ sudo a2enmod proxy proxy_http proxy_balancer lbmethod_byrequests $ sudo systemctl restart apache2 # restart needed for new modules, not just reload

/etc/apache2/sites-available/api.osztromok.com.conf <VirtualHost *:80> ServerName api.osztromok.com # Forward all requests to the backend app on port 8080 ProxyPreserveHost On ProxyPass / http://localhost:8080/ ProxyPassReverse / http://localhost:8080/ ErrorLog ${APACHE_LOG_DIR}/api_error.log CustomLog ${APACHE_LOG_DIR}/api_access.log combined </VirtualHost>

`ProxyPreserveHost On`	Passes the original `Host: api.osztromok.com` header to the backend app. Without this, the app sees `Host: localhost:8080`, which can break apps that construct absolute URLs.
`ProxyPass /`	Forward all requests (`/`) to `localhost:8080` . A request for `/users/42` becomes `localhost:8080/users/42` .
`ProxyPassReverse /`	Rewrites `Location:` headers in redirect responses from the backend — so the browser sees `https://api.osztromok.com/...` rather than `http://localhost:8080/...` .

$ sudo a2ensite api.osztromok.com.conf $ sudo apache2ctl configtest && sudo systemctl reload apache2 # Test — the curl response should come from your backend app $ curl -H "Host: api.osztromok.com" http://localhost/health {"status": "ok"} ← response from FastAPI/Node running on :8080

Subdomain Deployment Checklist

Use this every time you add a new subdomain. Work top-to-bottom — each step depends on the previous one.

New Subdomain Checklist — print or tick as you go
Document root created — `sudo mkdir -p /var/www/subdomain.osztromok.com/public_html` with correct ownership (`philip:www-data`) and permissions (755 dirs, 644 files).
Content placed — at minimum an `index.html` in the document root, so a successful hit returns visible content rather than a 403/directory listing.
Apache vhost config written — `/etc/apache2/sites-available/subdomain.osztromok.com.conf` with correct ServerName, DocumentRoot, Directory block (AllowOverride + Require all granted), and separate log files.
Modules enabled if needed — `sudo a2enmod rewrite` for .htaccess rewrites; `sudo a2enmod proxy proxy_http` for reverse proxy; restart Apache after enabling modules.
Site enabled — `sudo a2ensite subdomain.osztromok.com.conf` to create the symlink in sites-enabled.
Config validated — `sudo apache2ctl configtest` returns "Syntax OK" before reloading. Never skip this step.
Apache reloaded — `sudo systemctl reload apache2` — not restart.
Local test passes — `curl -H "Host: subdomain.osztromok.com" http://localhost` returns the correct content. This confirms Apache is working before DNS is involved.
DNS record created — CNAME in Cloudflare (proxied/orange cloud) pointing `subdomain` → `osztromok.com`, OR new public hostname added to the Cloudflare Tunnel config.
DNS verified — `dig subdomain.osztromok.com +short` returns a Cloudflare IP (not NXDOMAIN, not your home IP).
External test passes — `curl https://subdomain.osztromok.com` from outside the local network (or via mobile data) returns the correct content with 200 OK.
Main site unaffected — `curl -H "Host: osztromok.com" http://localhost` still returns the correct main site content. Adding a new vhost should never disturb existing ones.

Troubleshooting

DNS_PROBE_FINISHED_NXDOMAIN — "This site can't be reached"
The DNS record doesn't exist or hasn't propagated yet. Check: `dig subdomain.osztromok.com +short` — if it returns nothing, the record is missing from Cloudflare DNS (or you only added the public hostname to the tunnel but not the DNS record, and the tunnel didn't create one automatically). Also check: if using Cloudflare Tunnel via the dashboard GUI, the public hostname *does* auto-create the DNS record. If using the CLI, run `cloudflared tunnel route dns tunnel-name subdomain.osztromok.com` manually.
DNS resolves but serves the wrong site (main site content instead of subdomain)
DNS is working but the Apache vhost isn't. Run `sudo apache2ctl -S` and check whether the subdomain vhost appears in the list. If not, it's not enabled — run `sudo a2ensite subdomain.conf` and reload. If it is listed but still wrong, check ServerName in the config matches exactly what's in the Host header (including capitalisation, which Apache treats case-insensitively but typos matter).
SSL error — "NET::ERR_CERT_COMMON_NAME_INVALID" or similar
The subdomain's DNS isn't proxied through Cloudflare (grey cloud instead of orange). Cloudflare's shared SSL certificate only covers proxied subdomains. Fix: in Cloudflare DNS, click the cloud icon next to the subdomain record and switch from grey (DNS only) to orange (proxied). Wait

a moment and refresh.

Reverse proxy returns 502 Bad Gateway

Apache can reach the ProxyPass destination but the backend isn't responding. Check: (1) is the app actually running? `ss -tlnp | grep 8080` — if nothing's listening on that port, start the app. (2) Is it listening on `127.0.0.1` or `0.0.0.0`? The app must accept connections from localhost. (3) Check the backend app's own logs for errors.

Wildcard subdomain works but specific subdomains with their own vhost serve the wildcard content instead

The wildcard vhost is loading before the specific vhost. This happens when the wildcard filename sorts alphabetically before the specific vhost (e.g. `a-wildcard.conf` before `blog.conf`). Fix: rename the wildcard config to something that sorts last, like `z-wildcard.conf`, then run `a2dissite old-name.conf && a2ensite z-wildcard.conf` and reload.

Quick Reference — Chapter 6

Task	Action
New static subdomain	Create dir → write vhost → a2ensite → configtest → reload → test locally → add DNS → test externally
Redirect www → apex	Vhost with `Redirect permanent / https://apex.com/` OR Cloudflare Redirect Rule
Wildcard DNS	CNAME record with Name: * → Target: `osztromok.com` in Cloudflare, proxied
Wildcard Apache	`ServerAlias *.osztromok.com` in vhost, filename should sort last alphabetically
Reverse proxy	`a2enmod proxy proxy_http` → restart → vhost with ProxyPreserveHost + ProxyPass + ProxyPassReverse
Test vhost locally	`curl -H "Host: sub.osztromok.com" http://localhost`
Verify DNS live	`dig sub.osztromok.com +short` → should return Cloudflare IP
Check vhost order	`sudo apache2ctl -S` → lists all vhosts in match priority order

Symptom	Likely cause
NXDOMAIN / "can't be reached"	DNS record missing in Cloudflare, or not propagated yet
Wrong site content served	Apache vhost not enabled, or ServerName mismatch
SSL certificate error	DNS not proxied (grey cloud) — switch to orange cloud in Cloudflare
502 Bad Gateway	Backend app not running on the port specified in ProxyPass
Wildcard catches specific subdomain	Wildcard vhost sorting before specific one — rename to z- prefix

WEB HOSTING ESSENTIALS — CHAPTER 7

Dynamic DNS

Chapter 7 — Dynamic DNS

Consumer ISPs like Virgin Media assign dynamic IP addresses — the same address isn't guaranteed to stick. When your IP changes, any DNS A record pointing directly at your home IP becomes stale and your server becomes unreachable. Dynamic DNS solves this automatically: a daemon on your server detects the change and updates the DNS record within minutes. This chapter covers how DDNS works, when Cloudflare Tunnel removes the requirement entirely, and two practical implementations: DuckDNS (simplest) and Cloudflare API (best for the existing setup).

What this chapter covers: Why home IPs change and how DDNS responds. When Cloudflare Tunnel makes DDNS irrelevant for web traffic (but not for SSH). DDNS provider options. DuckDNS setup with a cron job. ddclient configured against the Cloudflare API — updates your own A record automatically. systemd timer as a modern cron replacement. Alerting when your IP changes. Troubleshooting stale records and failed updates.

The Dynamic IP Problem

Static IP addresses cost extra on consumer broadband and Virgin Media doesn't offer them on residential plans. Your IP is leased by DHCP from the ISP and renewed periodically — or reassigned whenever the router reboots, after a long power cut, or just at the ISP's discretion.

Without DDNS — IP changes break DNS: Day 1: osztromok.com A → 82.2.236.221 ← your IP today Visitors → 82.2.236.221 → your server ✓ works Router reboots / ISP renews lease: Day 3: Your new IP = 82.3.45.67 osztromok.com A still → 82.2.236.221 (stale!) Visitors → 82.2.236.221 → nobody home ✗ broken With DDNS — daemon detects change and updates the record: Day 3: DDNS daemon sees 82.3.45.67 ≠ 82.2.236.221 Calls Cloudflare API → updates A record → 82.3.45.67 (within ~5 minutes of IP change) Visitors → 82.3.45.67 → your server ✓ works again

The key parameters that determine downtime are the **detection interval** (how often the daemon checks your IP) and the DNS **TTL** (how long resolvers cache the old IP). If you check every 5 minutes and TTL is 60 seconds, maximum downtime is about 6 minutes.

When Cloudflare Tunnel Removes the DDNS Requirement

If your web traffic uses Cloudflare Tunnel (Chapter 4), your home IP is *not in the DNS records for your website at all*. The DNS records point to Cloudflare's edge (the tunnel CNAME), not your home IP. The tunnel itself is an outbound connection — it reconnects automatically with whatever IP your router currently has.

Tunnel setup — your IP is completely invisible: osztromok.com CNAME → abc123.cfargotunnel.com (Cloudflare's servers) Visitors never see or connect to 82.2.236.221 When your IP changes: - cloudflared daemon reconnects to Cloudflare outbound (with new IP) - No DNS record needs updating - Visitors are unaffected ✓ Website DDNS: NOT needed (tunnel handles it) ✗ SSH access DDNS: NEEDED if you SSH in by IP or home hostname
DDNS is still worth setting up for SSH access. Even with the tunnel handling web traffic, if you ever want to SSH into your server from outside your home network, you need a way to find its current IP. The common approach: keep a separate A record for `home.osztromok.com` or `ssh.osztromok.com` updated by DDNS — not proxied through Cloudflare (so it exposes the real IP directly for SSH). This record doesn't route web traffic and doesn't need the tunnel.

DDNS Provider Options

Cloudflare API (Recommended)
Update your own A records directly through Cloudflare's API. No third-party service. Works because your DNS is already on Cloudflare.

- Use your own domain (osztromok.com)
- Update any record you control
- Works with ddclient or a curl script
- Free — uses your existing Cloudflare account

Best for: keeping ssh.osztromok.com pointing at your real IP.
DuckDNS (Simplest)

Free service with a simple curl-based update. Gives you a `yourname.duckdns.org` subdomain.

- Completely free, no credit card
- Login with GitHub/Google/etc.
- One-liner curl update command
- Subdomain is .duckdns.org (not your own domain)

Best for: quick setup to test DDNS, or SSH access when you don't need your own domain name.
No-IP
Popular DDNS provider with free tier. Custom client available.

- Free tier: 3 hostnames
- Must confirm free hostnames monthly
- Supports custom domains on paid plan
- Has its own official Linux client

Best for: users who want a branded DDNS service with a GUI.
Dynu
Free DDNS with custom domain support and no nag emails.

- Free custom domain DDNS
- No monthly confirmation needed
- Supports ddclient
- Less well-known but reliable

Best for: free custom domain DDNS without No-IP's renewal hassle.

Checking Your Public IP

DDNS clients work by comparing your current public IP against what's in the DNS record. Checking your IP from the server itself requires querying an external service — `ip route` only shows your local network IP, not the public one the internet sees.

Several ways to get your current public IP from the server $ curl -s https://api.ipify.org 82.2.236.221 $ curl -s https://ifconfig.me 82.2.236.221 $ curl -s https://checkip.amazonaws.com 82.2.236.221 # Store it in a variable for scripting $ CURRENT_IP=$(curl -s https://api.ipify.org) $ echo "Public IP: $CURRENT_IP" Public IP: 82.2.236.221 # Compare against what's in DNS $ DNS_IP=$(dig +short ssh.osztromok.com) $ echo "DNS IP: $DNS_IP" DNS IP: 82.2.236.221 ← match = no update needed

Method 1 — DuckDNS (Simplest)

Setup Walkthrough · DuckDNS
Get a free duckdns.org subdomain and keep it updated automatically with a cron job.
1
Register a subdomain. Go to `duckdns.org`, log in with GitHub/Google, and create a subdomain — e.g. `osztromok.duckdns.org`. DuckDNS shows you your token on the dashboard. Copy it — you'll need it in the update URL.
2
Create the update script.

$ mkdir -p ~/duckdns $ nano ~/duckdns/duck.sh
Content of the script:

#!/bin/bash TOKEN="your-duckdns-token-here" DOMAIN="osztromok" # just the subdomain part, not .duckdns.org echo url="https://www.duckdns.org/update?domains=${DOMAIN}&token=${TOKEN}&ip=" \| curl -s -K - -o ~/duckdns/duck.log

$ chmod +x ~/duckdns/duck.sh $ ~/duckdns/duck.sh # run once to test $ cat ~/duckdns/duck.log OK ← "OK" = update succeeded, "KO" = failed (check token)
3
Add a cron job to run every 5 minutes.

$ crontab -e

Add this line at the bottom:

*/5 * * * * ~/duckdns/duck.sh >/dev/null 2>&1

This runs the update script every 5 minutes. DuckDNS is smart enough to do nothing if the IP hasn't changed — it's a cheap curl call.

4

Verify the IP is current.

$ dig +short osztromok.duckdns.org 82.2.236.221 ← should match curl https://api.ipify.org

The DuckDNS subdomain (osztromok.duckdns.org) can be used as an SSH target: `ssh philip@osztromok.duckdns.org`. It always resolves to your current home IP. TTL is 60 seconds, so after an IP change it's current within a minute of the next cron run.

Method 2 — Cloudflare API with ddclient

ddclient is a mature DDNS daemon that supports dozens of providers including Cloudflare. It runs as a system service, detects IP changes, and updates the DNS record. This approach lets you keep a record like `ssh.osztromok.com` on your own domain pointing at your real home IP.

Step 1 — Create a Cloudflare API token

Go to **Cloudflare dashboard → My Profile → API Tokens → Create Token**. Use the **Edit zone DNS** template:

- **Permissions:** Zone → DNS → Edit
- **Zone Resources:** Include → Specific zone → osztromok.com
- Click Continue and Create Token. Copy the token — it's only shown once.

Use an API Token, not the Global API Key. The Global API Key has full account access. The scoped token above only permits editing DNS for osztromok.com — if it ever leaks, the damage is limited.

Step 2 — Create the DNS record to be updated

In Cloudflare DNS, add an A record for the SSH hostname:

Type	Name	Content	Proxy	TTL
A	ssh	82.2.236.221 (current IP)	Grey cloud (DNS only)	60 seconds

Grey cloud (DNS only) is *essential* here — this record needs to expose the real IP so SSH clients can connect directly. The orange cloud would hide the IP behind Cloudflare, which breaks SSH. TTL 60 seconds means the record updates are visible within a minute.

Step 3 — Install ddclient

$ sudo apt update && sudo apt install ddclient -y # The installer may show a configuration wizard — skip it (press Cancel or Esc) # We'll write the config file manually

Step 4 — Write the ddclient config

/etc/ddclient.conf — Cloudflare provider ## How often to check for IP change (seconds) daemon=300 # check every 5 minutes syslog=yes # log to syslog (journalctl -u ddclient) pid=/var/run/ddclient.pid ## How to detect the current public IP use=web web=api.ipify.org # external service that returns plain IP text ## Cloudflare provider settings protocol=cloudflare zone=osztromok.com # your Cloudflare zone ttl=1 # TTL 1 = Cloudflare's minimum (auto = 60s) ## Cloudflare authentication — use the scoped API token login=token password=your-cloudflare-api-token-here ## DNS record(s) to update — one per line ssh.osztromok.com # updates this A record with current IP

Lock down the config file — it contains your API token $ sudo chmod 600 /etc/ddclient.conf $ sudo chown root:root /etc/ddclient.conf # Test the config before starting the service $ sudo ddclient -daemon=0 -debug -verbose -noquiet DEBUG: CONNECT: api.ipify.org DEBUG: SENDING: GET / HTTP/1.0 DEBUG: RECEIVE: 82.2.236.221 SUCCESS: ssh.osztromok.com: updated successfully to 82.2.236.221 # Enable and start the service $ sudo systemctl enable ddclient $ sudo systemctl start ddclient $ sudo systemctl status ddclient ● ddclient.service Active: active (running) # Watch the logs live $ sudo journalctl -u ddclient -f

Update multiple records. If you want to keep more than one record updated, list them one per line at the bottom of ddclient.conf — all using the same zone and token settings above. For example, to also update `home.osztromok.com`, add it on a new line after `ssh.osztromok.com`.

Alternative — Pure Bash Script via Cloudflare API

If you'd rather not install ddclient, the same result is achievable with a short bash script and cron. This is useful if you want full visibility into what's happening or want to add custom logic (notifications, logging).

#!/bin/bash # /usr/local/bin/cf-ddns.sh — update Cloudflare A record with current IP API_TOKEN="your-cloudflare-api-token" ZONE_ID="your-zone-id" # Cloudflare dashboard → Overview → Zone ID (right sidebar) RECORD_NAME="ssh.osztromok.com" STATE_FILE="/var/cache/cf-ddns-last-ip" # Get current public IP CURRENT_IP=$(curl -s https://api.ipify.org) # Read last known IP LAST_IP=$(cat "$STATE_FILE" 2>/dev/null) # Exit early if IP hasn't changed — avoids unnecessary API calls if ["$CURRENT_IP" = "$LAST_IP"]; then exit 0 fi # Get the record ID for the A record we want to update RECORD_ID=$(curl -s -X GET \ "https://api.cloudflare.com/client/v4/zones/${ZONE_ID}/dns_records?type=A&name=${RECORD_NAME}" \ -H "Authorization: Bearer ${API_TOKEN}" \ -H "Content-Type: application/json" \ | python3 -c "import sys,json; print(json.load(sys.stdin)['result'][0]['id'])") # Update the record RESULT=$(curl -s -X PATCH \ "https://api.cloudflare.com/client/v4/zones/${ZONE_ID}/dns_records/${RECORD_ID}" \ -H "Authorization: Bearer ${API_TOKEN}" \ -H "Content-Type: application/json" \ --data "{\"content\":\"${CURRENT_IP}\"}") # Save the new IP so we don't update again unnecessarily echo "$CURRENT_IP" > "$STATE_FILE" echo "$(date): Updated ${RECORD_NAME} → ${CURRENT_IP}"
$ sudo chmod +x /usr/local/bin/cf-ddns.sh $ sudo cf-ddns.sh # test run Sun Jun 14 21:30:00 BST 2026: Updated ssh.osztromok.com → 82.2.236.221 # Schedule with cron (every 5 minutes) $ sudo crontab -e # Add: */5 * * * * /usr/local/bin/cf-ddns.sh >> /var/log/cf-ddns.log 2>&1

Systemd Timer — Modern Alternative to Cron

Systemd timers are the modern replacement for cron on Debian/Ubuntu systems. They have better logging (integrated with journald), dependency support, and can be randomised to avoid thundering-herd problems on servers that all update at exactly the same second.

/etc/systemd/system/cf-ddns.service [Unit] Description=Cloudflare DDNS update After=network-online.target # don't run before network is ready [Service] Type=oneshot ExecStart=/usr/local/bin/cf-ddns.sh
/etc/systemd/system/cf-ddns.timer [Unit] Description=Run Cloudflare DDNS update every 5 minutes [Timer] OnBootSec=2min # run 2 minutes after boot OnUnitActiveSec=5min # then every 5 minutes RandomizedDelaySec=30s # spread load: run anywhere in a 30s window [Install] WantedBy=timers.target
$ sudo systemctl daemon-reload $ sudo systemctl enable cf-ddns.timer $ sudo systemctl start cf-ddns.timer # Check the timer status and when it will next fire $ sudo systemctl list-timers cf-ddns.timer NEXT LEFT LAST PASSED UNIT ACTIVATES Sun 2026-06-14 21:35:22 BST 4min Sun 2026-06-14 21:30:00 BST 22s cf-ddns.timer cf-ddns.service # View logs for past runs $ sudo journalctl -u cf-ddns.service --since today

Alerting When Your IP Changes

Worth adding a notification so you know when an IP change happens — useful for auditing that DDNS actually worked and for spotting unexpected IP changes (which might indicate a router issue).

Add to cf-ddns.sh after the state file write, to send an email alert # Send email via msmtp (or any installed mail sender) echo "IP changed from ${LAST_IP:-unknown} to ${CURRENT_IP} — dns updated" \ | mail -s "DDNS update: ${RECORD_NAME}" emubantam@gmail.com # Or log to a file with timestamp for manual review echo "$(date '+%Y-%m-%d %H:%M:%S') IP changed: ${LAST_IP:-unknown} → ${CURRENT_IP}" \ >> /var/log/ip-changes.log
View the IP change history $ cat /var/log/ip-changes.log 2026-04-10 03:22:14 IP changed: 82.2.236.221 → 82.3.45.67 2026-05-20 07:41:03 IP changed: 82.3.45.67 → 82.2.236.221 # Virgin Media sometimes reassigns the same IP you had before

Troubleshooting

ddclient runs but DNS record hasn't updated — still shows old IP
Run ddclient in debug mode to see exactly what's happening: `sudo ddclient -daemon=0 -debug -verbose -noquiet`. Common causes:
— **Auth failure:** "login" is wrong — in ddclient.conf, `login=token` is literal text (not your email), `password=` is the API token value.
— **Wrong zone:** the zone= value must exactly match your Cloudflare zone name.
— **Record not found:** the hostname at the bottom of the config must match an existing DNS record in Cloudflare. Create the A record manually first, then ddclient will update it.
DuckDNS script returns "KO" instead of "OK"
The token or domain name is wrong. Double-check on the DuckDNS dashboard — the token is the long string under your account name. The domain should be just the subdomain part (e.g. `osztromok`, not `osztromok.duckdns.org`). Test manually: `curl "https://www.duckdns.org/update?domains=YOURDOMAIN&token=YOURTOKEN&ip="`
IP updated in Cloudflare but DNS still resolves to old IP
TTL caching. Even though Cloudflare updated the record, resolvers that cached the old answer continue to serve it until the TTL expires. Check: `dig ssh.osztromok.com +short` — if the record is grey cloud (DNS only), Cloudflare should apply the TTL you set (60 seconds is the minimum). If it's proxied/orange cloud, Cloudflare caches differently. Wait the TTL interval and dig again.

Cron job not running — script works manually but not on schedule

Cron's environment is minimal — it may not have the same PATH as your interactive session. Use absolute paths in cron scripts (`/usr/bin/curl` not `curl`). Check cron's own log: `grep CRON /var/log/syslog | tail -20`. Also verify the script is executable (`chmod +x`) and that the crontab line has no typos in the timing fields.

Cloudflare tunnel reconnects fine after IP change but SSH access breaks

This is the expected split: the tunnel (web traffic) uses an outbound connection that reconnects automatically. SSH access via `ssh.osztromok.com` uses a DNS A record that points to your real IP — this record needs DDNS to stay current. Verify DDNS is running: `sudo systemctl status ddclient` or check the state file: `cat /var/cache/cf-ddns-last-ip` and compare against `curl -s https://api.ipify.org`.

Quick Reference — Chapter 7

Command	Purpose
curl -s https://api.ipify.org	Get your current public IP from the server
dig +short ssh.osztromok.com	Check what IP is currently in the DNS record
sudo ddclient -daemon=0 -debug -verbose -noquiet	Run ddclient once in foreground with full debug output — best troubleshooting tool
sudo journalctl -u ddclient -f	Watch ddclient daemon logs live
sudo journalctl -u cf-ddns.service --since today	View today's systemd timer runs for the custom script
sudo systemctl list-timers	Show all timers and when they last/next ran
crontab -l	List current user's cron jobs
grep CRON /var/log/syslog | tail -20	Check whether cron ran recently (and if it logged errors)

Scenario	DDNS needed?	Best approach
Website via Cloudflare Tunnel	No	Tunnel handles it — no home IP in DNS
SSH into home server from outside	Yes	A record (grey cloud, TTL 60s) + ddclient updating it
Direct connection (no tunnel)	Yes	A record + ddclient or DuckDNS
Just want to know your current IP	No	`curl -s https://api.ipify.org`

File / Location	Purpose
/etc/ddclient.conf	ddclient configuration — provider, token, records to update. Chmod 600.
/var/cache/ddclient.cache	ddclient's record of the last IP it sent — compared on each check
/var/cache/cf-ddns-last-ip	State file for the custom bash script approach
~/duckdns/duck.log	DuckDNS update response log (OK or KO)
/var/log/ip-changes.log	Custom IP change history log

WEB HOSTING ESSENTIALS — CHAPTER 8

Self-Hosting vs VPS

Chapter 8 — Self-Hosting vs VPS

You've built a working home server with public access via Cloudflare Tunnel. Now the question is: when does it make sense to move to a VPS, and when is self-hosting the right call? This chapter is honest about the trade-offs — cost, reliability, control, and the ISP restrictions that make home hosting harder than it should be. It also covers the hybrid approach: keeping your home server for most things while using a cheap VPS for what it's actually better at.

What this chapter covers: What self-hosting and VPS hosting really mean day-to-day. Complete pros and cons for both. Real cost comparison including electricity, hardware, and time. Best-value VPS providers in 2026. The hybrid approach (VPS as jump server or tunnel endpoint). Decision framework — when to switch and when to stay. First five steps when you spin up a VPS for the first time. Course summary.

What Each Option Actually Means

Self-hosting (what you have now): Your home │ Your server │ Your Apache │ Your files broadband ──┤ (Linux box) ├──────────▶│ /var/www/ (dynamic IP) │ running 24/7 │ port 80/443 │ osztromok.com │ │ │ └── Cloudflare Tunnel ──────┘ bypasses ISP block VPS (renting a slice of a data centre): Data centre │ Your VPS │ Your Apache │ Your files (1 Gbps │ (Debian VM) ├──────────▶│ /var/www/ fibre, UPS, │ static IP │ port 80/443 │ osztromok.com generators) │ no ISP block │ │ │ 99.9% uptime │ │

The core difference is **who provides the infrastructure**. Self-hosting means you own the hardware, pay the electricity, and your ISP's policies are your ceiling. A VPS means renting compute in a data centre — you get a static IP, no ISP restrictions, and someone else worries about power, cooling, and network redundancy.

Self-Hosting — The Full Picture

Advantages

- **Effectively free** — hardware you already own + ~£5–10/month electricity for a Pi or mini PC. No ongoing subscription.
- **Unlimited storage** — add a hard drive, not a pricing tier. A 4TB drive is a one-time cost.
- **No bandwidth caps** — Virgin Media gives you a home broadband allowance. Serve as much as you want within it.
- **Full hardware control** — GPUs, USB devices, cameras, sensors — things a VPS can't touch.
- **Privacy** — your data never leaves your home. No cloud provider can access it.
- **Learning value** — every problem you solve on home hardware teaches you something a managed service would hide.
- **Local network speed** — internal services (NAS, Plex, Home Assistant) run at LAN speeds, not through the internet.

Disadvantages

- **ISP restrictions** — Virgin Media blocks inbound port 80/443. Cloudflare Tunnel works around it, but it's a workaround.
- **Dynamic IP** — DDNS required for direct access (Chapter 7). Tunnel makes this irrelevant for web traffic.
- **Single points of failure** — power cut, router reboot, ISP outage, hardware failure → site goes down. No SLA.
- **Upload speed** — Virgin Media home broadband upload is typically 10–20 Mbps. Fine for a personal site, limiting for high traffic.
- **Security exposure** — an internet-connected device at home means a compromised server is on the same network as everything else.
- **No IPv6 support** on some Virgin Media plans, complicating modern deployment.
- **Time cost** — hardware breaks, you fix it. Nobody else's problem.

VPS — The Full Picture

Advantages

- **Static public IP** — no DDNS, no Cloudflare Tunnel needed. Direct port 80/443 with no ISP interference.
- **Data centre reliability** — UPS, generator backup, redundant network. 99.9%+ uptime guarantees typical.
- **Fast symmetric bandwidth** — 1 Gbps is standard. Upload as fast as download.
- **Isolated from home network** — a compromised VPS doesn't put your home devices at risk.
- **Easy to snapshot and restore** — most providers let you take a full VM snapshot in one click.
- **Spin up / down on demand** — resize, clone, destroy with no physical hardware involved.
- **IPv6 included** — all major VPS providers assign a public IPv6 address.

Disadvantages

- **Ongoing monthly cost** — £3–10/month indefinitely. Small, but it adds up over years: £36–120/year.
- **Storage costs money** — base VPS typically 20–40GB SSD. Extra storage billed separately.
- **Your data on someone else's hardware** — provider can theoretically access it. Encryption helps.
- **Still your responsibility** — the OS, security patches, Apache config, backups — all still you. A VPS is not a managed service.
- **Bandwidth limits** — usually 1–3 TB/month on cheap plans. Fine for most sites; heavy traffic incurs overage charges.
- **No physical access** — can't plug in a USB drive, GPU, or USB key. Everything remote.

Real Cost Comparison

Cost Item	Self-Hosting	VPS (Hetzner CX22)
Monthly fee	£0	~£4.50/month
Electricity (idle server)	~£5–12/month (5–15W × 24hr)	Included
Hardware	Already owned (Raspberry Pi / mini PC). Replace every 4–7 years.	None
Bandwidth	Home broadband (already paid)	20 TB/month included — far more than needed
Storage	Cheap — add HDDs as needed	40GB SSD. Extra costs £0.04/GB/month.
Static IP	Not available (Virgin Media). Use Tunnel.	Included
Uptime SLA	None — depends on your hardware + ISP	99.9% SLA (network)
Time to fix outages	You, immediately, whatever time it is	Hardware managed by provider
Realistic monthly total	**£5–12** (electricity only)	**£4.50** (fixed)

The costs are surprisingly close. If your server draws 10W and electricity is 25p/kWh, that's £1.80/month — cheaper than any VPS. But a desktop draws 50–150W, adding £9–27/month. **Power consumption is the make-or-break factor for home hosting economics.** A Raspberry Pi or a modern mini PC (N100/N305) running at 5–8W is genuinely cheaper than a VPS. A desktop PC left on 24/7 is not.

Check your server's actual power draw: plug it into a smart plug with energy monitoring (Tapo P110, Shelly Plug S) and read the real wattage. Multiply by 24 × 30 × £0.25 to get the monthly electricity cost. That number tells you whether self-hosting is cheaper than a VPS.

Best-Value VPS Providers (2026)

Hetzner Recommended
£3.60–5/mo
German provider, excellent reputation. CX22 (2 vCPU, 4GB RAM, 40GB SSD, 20TB traffic) is the sweet spot. Data centres in Nuremberg, Falkenstein, Helsinki, Ashburn. No UK DC, but latency from Germany is fine (~20ms). Best value in EU by a large margin.
Oracle Cloud Free forever
£0
Genuinely free — Always Free tier includes 4 ARM CPUs + 24GB RAM (split across up to 4 instances) + 200GB storage. Catch: sign-up requires a credit card, instances can be reclaimed if "idle." Hard to provision due to capacity. Worth trying — remarkable specs at zero cost.
Vultr UK DC
~£4.50/mo
Has a London data centre — useful if UK latency matters or if GDPR requires UK data residence. $6/month starter is 1 vCPU / 1GB RAM / 25GB SSD. Good control panel, hourly billing. Slightly pricier than Hetzner for equivalent specs.
DigitalOcean UK DC
~£5.50/mo
Excellent developer UX, great documentation. $6/month Droplet (1 vCPU / 1GB / 25GB). London data centre. Better-known than Hetzner, slightly more expensive. Great if you want a polished dashboard and one-click apps (WordPress, LAMP stack etc.).
Contabo
~£4/mo
Very cheap specs — 4 vCPU / 8GB RAM for ~£5. European data centres. Trade-off: shared resources, slower support, less polished experience. Good for hobbyist workloads where raw specs matter more than reliability guarantees.
Linode (Akamai)
~£4.50/mo
Reliable, long-established. Now owned by Akamai. Nanode plan: 1 vCPU / 1GB / 25GB / 1TB traffic for $5/month. Good documentation, predictable pricing. London DC available. Similar to DigitalOcean in feel.

Pricing changes. These figures are approximate as of mid-2026. Always check the current pricing page before signing up — providers adjust prices regularly. Hetzner in particular sometimes changes its Euro pricing, which affects the GBP equivalent.

The Hybrid Approach

You don't have to choose one or the other. Many home hosting setups use a small cheap VPS for the specific things a VPS is better at, while keeping the home server for storage-heavy or hardware-dependent workloads. Three practical hybrid configurations:

Option A — VPS as SSH jump server
Cheapest hybrid. Keep the home server for everything. Use a tiny £3/month VPS as a fixed jump point for SSH access.

How it works: create an SSH tunnel from the home server to the VPS (`ssh -R 2222:localhost:22 user@vps-ip` as a systemd service). SSH into the VPS first, then hop to the home server via `ssh -p 2222 localhost`. VPS does no serving — it's just a door.

Best for: you want a reliable way to SSH into your home server even when your home IP changes. £3/month for this is reasonable.
Option B — VPS replaces Cloudflare Tunnel
Instead of relying on Cloudflare Tunnel, run your own reverse proxy on a VPS. The VPS accepts traffic on ports 80/443, forwards it to the home server via a persistent SSH or WireGuard tunnel.

How it works: WireGuard VPN between VPS and home server → Nginx on VPS proxies `osztromok.com` to the home server's WireGuard IP → home server serves the content.

Pros vs Cloudflare Tunnel: not dependent on Cloudflare, full control of the reverse proxy, can use standard Let's Encrypt certs. **Cons:** more setup, costs £3–5/month, you maintain the VPS.

Best for: those who want to reduce dependency on Cloudflare or need features the tunnel doesn't support (non-HTTP protocols, custom TCP ports).
Option C — VPS for public, home server for private
Split by visibility. Public-facing website and APIs live on the VPS (static IP, fast, reliable). Heavy-storage services — Nextcloud, Plex, backups, home automation — stay at home where storage is cheap and data stays local.

How it works: two separate environments. The VPS runs Apache/Nginx for public content. The home server is only accessible on the local network (or via VPN/tunnel for authorised access).

Best for: growing setups where the public site has matured beyond a learning project and needs real uptime, but you still want cheap local storage.

Decision Framework — When to Move, When to Stay

Is your home server drawing more than 20W continuously?
YES → Electricity cost is £8+/month. A VPS is probably cheaper. **Consider VPS.**
NO → Home power cost is under £4/month. Self-hosting wins on cost.
Does your site need 99.9%+ uptime (customers, revenue, commitments)?
YES → Home hosting can't reliably deliver this. ISP outages, router reboots, power cuts.
Move to VPS — uptime is non-negotiable for production services.
NO → Personal/hobby sites tolerate occasional downtime. Stay home.
Do you need more than 20 Mbps sustained upload?
YES → Virgin Media home upload typically maxes out around 10–20 Mbps. High-traffic video, large downloads, API-heavy services will hit this ceiling.
VPS — data centre bandwidth solves this.
NO → A personal site with a few thousand monthly visitors uses well under this. Stay home.
Is Cloudflare Tunnel working reliably for your use case?
YES → You've bypassed the Virgin Media restriction elegantly. No reason to move.
NO (non-HTTP needs, performance issues, reliability concerns) →
Consider VPS or hybrid (Option B — self-hosted reverse proxy).
Is this primarily for learning / homelab?
YES → Stay home. Home hosting teaches you more. The constraints are the lesson.

Keep the home server. The problems you hit are features, not bugs.

For your specific situation — learning project, Virgin Media, Cloudflare Tunnel working — **the home server is the right answer right now.** The tunnel solved the ISP restriction problem. A £3 Hetzner VPS is worth having alongside it purely as an SSH jump box or for a second experimental environment.

First Steps When You Spin Up a VPS

If you do go the VPS route — whether for a jump box, a hybrid setup, or full migration — these are the first things to do on a fresh Debian/Ubuntu VPS before anything else.

1
Add your SSH public key and disable password login. Most providers let you add an SSH key at provisioning. If not: `ssh-copy-id root@vps-ip`, then edit `/etc/ssh/sshd_config` and set `PasswordAuthentication no`. Reload sshd. Password-based SSH is attacked constantly — key auth is not optional.
2
Create a non-root user. `adduser philip` → `usermod -aG sudo philip` → copy your SSH key to the new user → test login → then disable root SSH login (`PermitRootLogin no` in sshd_config). Never work as root day-to-day.
3
Update everything. `sudo apt update && sudo apt upgrade -y && sudo apt autoremove -y`. Then configure unattended security upgrades: `sudo apt install unattended-upgrades -y && sudo dpkg-reconfigure unattended-upgrades`. A VPS that installs security patches automatically is dramatically safer than one that doesn't.
4
Set up the firewall. `sudo apt install ufw -y` → `sudo ufw allow ssh` → `sudo ufw allow 80/tcp` → `sudo ufw allow 443/tcp` → `sudo ufw enable`. UFW defaults to deny all inbound. Open only what you need. A VPS without a firewall is exposed to the entire internet on every port.
5
Install fail2ban. `sudo apt install fail2ban -y` → `sudo systemctl enable fail2ban && sudo systemctl start fail2ban`. Default config bans IPs with 5 failed SSH attempts for 10 minutes. On a public-IP VPS, you'll have hundreds of brute-force attempts per day — fail2ban handles them silently.
6
Install Apache and point DNS at the VPS IP. `sudo apt install apache2 -y`. The VPS has a static public IP — go to Cloudflare DNS, add an A record pointing your domain at that IP (orange cloud for CDN/SSL, or grey cloud for direct). No tunnel needed. Apache on port 80 will be publicly reachable immediately.
Quick "is the VPS ready?" checklist — run after initial setup $ ssh philip@vps-ip "whoami && sudo ufw status && sudo systemctl status fail2ban --no-pager | head -3" philip Status: active To Action From -- ------ ---- 22/tcp ALLOW Anywhere 80/tcp ALLOW Anywhere 443/tcp ALLOW Anywhere ● fail2ban.service - Fail2Ban Service Active: active (running)

Course Summary — What You've Built

You've worked through everything required to understand, configure, and maintain a complete home web hosting setup — and you know exactly when and why you'd move beyond it. Here's what this course covered:

Chapter 1
DNS Fundamentals
Resolution chain: recursive → root → TLD → authoritative.
A, CNAME, MX, TXT, NS records. TTL strategy.
Chapter 2
Managing DNS at IONOS
Registrar vs DNS host. @ notation. Adding A, CNAME, MX, TXT records. Pitfalls and propagation.
Chapter 3
Moving to Cloudflare
Orange vs grey cloud. NS transfer, CNAME flattening at apex. Migration without downtime.
Chapter 4
Cloudflare Tunnel
Outbound architecture bypasses ISP block. cloudflared install, config.yml, ingress rules, systemd service.
Chapter 5
Apache Virtual Hosts

sites-available/enabled, a2ensite. ServerName, DocumentRoot, Directory block, AllowOverride, per-site logs.
Chapter 6
Subdomains End-to-End
Two-part requirement: DNS + server. Deployment checklist, www redirect, wildcard, reverse proxy (ProxyPass).
Chapter 7
Dynamic DNS
DuckDNS cron script. ddclient + Cloudflare API. Systemd timer. When tunnel makes DDNS unnecessary.
Chapter 8
Self-Hosting vs VPS
Real cost comparison. Provider options. Hybrid approaches. Decision framework. VPS first-steps checklist.
What's covered in the companion course — Securing Your Web Server: HTTPS with Let's Encrypt, enforcing HTTPS redirects, UFW firewall in depth, fail2ban tuning, SSH hardening (key auth, port change, AllowUsers), Apache security headers (HSTS, CSP, X-Frame-Options), ClamAV malware scanning, and ongoing monitoring and maintenance. Security is the natural next step from everything in this course.

Quick Reference — Chapter 8

Situation	Recommendation
Personal / learning project	Self-host. The constraints are the learning. Cloudflare Tunnel handles Virgin Media.
Server draws >20W continuously	Electricity > VPS cost. Switch to VPS or get a low-power mini PC (N100/Pi).
Need 99.9% uptime	VPS. Home hosting cannot guarantee this.
Need large cheap storage	Home server. HDDs are far cheaper than VPS storage pricing.
Need reliable SSH from outside	Cheap VPS as SSH jump box (£3/month) or DDNS + grey-cloud A record.
Want to reduce Cloudflare dependency	Hybrid Option B: self-hosted WireGuard + VPS reverse proxy.
Best all-round cheap VPS	Hetzner CX22 (~£4.50/month). UK latency fine, excellent value.
Want free VPS	Oracle Cloud Always Free (ARM, 4 CPU/24GB RAM). Hard to provision, worth trying.

VPS first-steps command	Purpose
ssh-copy-id philip@vps	Copy your SSH public key to the VPS — enables key-based login
PasswordAuthentication no	In /etc/ssh/sshd_config — disable password SSH after key auth works
sudo ufw allow ssh/80/443	Open only the ports you actually need before enabling the firewall
sudo ufw enable	Activate the firewall — default deny all inbound
sudo apt install fail2ban -y	Automatic IP banning for brute-force SSH attempts
sudo apt install unattended-upgrades -y	Automatic security patches — essential on a public server