anyIP
Made Crystal Clear

A practical, visual guide to proxies and reliable web data with anyIP

CHRISTOPHE PAKA

anyIP

MADE CRYSTAL CLEAR

A practical, visual guide to proxies and reliable web data with anyIP

Christophe Paka

First Edition · 2026-08-08

anyIP Made Crystal Clear · First Edition

A practical, visual guide to proxies and reliable web data with anyIP

Author: Christophe Paka
Published 2026-08-08.

More books in the Made Crystal Clear series: https://madecrystalclear.dev

Sources: https://anyip.io/ and the official anyIP documentation.

> anyIP is a trademark of its respective owner. This is an independent, unofficial guide, not affiliated with, authorized by, or endorsed by anyIP. Product details are drawn from anyIP's public documentation as of the date shown and may change over time, so always verify against the official docs before you rely on them.

You may share this book freely for personal, non-commercial use. Please keep it intact and credit the author.

How to read this book

This is a hands-on, visual guide. Every chapter opens with **The gist**, explains a concept with a diagram before the prose, and closes with **Key takeaways** and a **Go deeper** link list. Recipes are numbered, copy-pasteable, and tell you exactly what a successful result looks like. Watch for these boxes as you read:

⚡ **The gist**
The 3 to 5 things a chapter delivers, up front.

📝 **Recipe**
Numbered steps with commands and a ✅ expected result to match.

📋 **Checklist**
Actionable items to tick off before moving on.

💡 **Tip**
A shortcut or clarification worth knowing.

⚠️ **Watch out**
A common mistake and how to avoid it.

💼 **Business angle**
Why the topic matters to cost, speed, or risk.

📌 **Key takeaways**
The chapter's summary, at the end.

🔗 **Go deeper**
The official docs and sources behind the chapter.

> ⚠️ **Watch out**
>
> This book teaches **reliable, authorized** web data collection and testing. Only access systems and data you are permitted to, respect each site's robots.txt and Terms of Service, rate-limit your requests, and follow the law. Chapter 9 covers ethics and compliance in full.

Contents

01 1. About This Book

02 2. The Ban Problem & the anyIP Answer

03 3. Core Concepts, The Smart Username

04 4. Getting Started, Your First Proxy in Under a Minute

05 5. Targeting, Country, City, Carrier, Coordinates

06 6. Sessions & Rotation, Sticky Without the Surprises

07 7. Real-World Recipes, Reliable Data Collection

08 8. When Things Break, Errors, UDP & Data Hygiene

09 9. Play It Straight, Ethical & Compliant Use

10 10. The Business Case, Pricing, ROI & Alternatives

11 11. Cheat Sheet & Learning Path

12 12. Resources

- Conclusion
- A note on this guide
- Acknowledgments
- About the author
- Thank you for reading

1. About This Book

> ⚡ **The gist**
>
> - A practical field guide to anyIP's residential & mobile proxy network, for developers, growth engineers, and data/QA teams.
> - By chapter 4 you'll have made your first proxied request; by chapter 8 you can diagnose any error code on sight.
> - Every technique is framed around **authorized** data collection, QA, and geo-testing, nothing else.
> - Eight callout boxes carry the signal: learn the legend once, skim the book forever.
>
> *Reading time: ~5 min*

Who this is for, and what you'll be able to do

This book is for people whose work depends on seeing the web the way real users see it: **developers** building scrapers and data pipelines, **growth and marketing engineers** verifying ads and SERPs across countries, and **data or QA teams** testing apps from geographies they don't sit in. If your requests keep dying with 403s and CAPTCHAs, or your "German pricing check" is silently served the US page, you're the reader.

You need three things coming in: comfort with a terminal, working knowledge of HTTP (methods, status codes, headers), and one scripting language. Examples use `curl` and Python, both translate directly to Node, Go, or anything else that can speak to a SOCKS5 or HTTPS proxy.

By the last page you will be able to: build an anyIP connection string from scratch and predict exactly what IP it produces; pin requests to a country, city, carrier, or GPS point; hold one IP across a multi-step workflow or rotate on every request; run three production-grade collection jobs in Python; diagnose every proxy error code; and put a defensible cost-per-row number in front of whoever pays the bill.

The road ahead

The book is a single learning path in four legs. **Foundations** (chapters 2 to 3) builds the mental model: why IPs get banned and how anyIP's "smart username" controls everything from one string. **Hands-on** (chapters 4 to 8) is where you type: first request, targeting, sessions, real recipes, troubleshooting. **Judgment & business** (chapters 9 to 10) covers what to run and what it costs. **Reference** (chapters 11 to 12) is the part you'll photocopy.

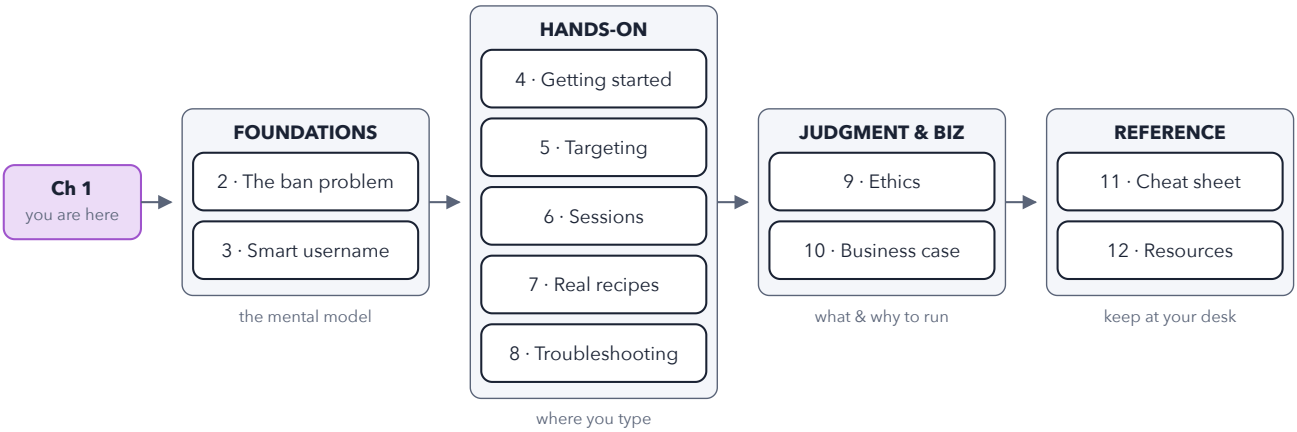


Figure 1.1, Book roadmap: 11 content chapters plus resources, in four legs. Read chapters 2 to 4 in order; after that, jump to whatever hurts.

> 💡 **Tip**
>
> In a hurry? Read chapters 2 to 4 in order (about 30 minutes), then jump straight to the chapter that matches your problem. Chapters 5 to 8 are written to stand alone; chapter 11 is the whole book compressed to one page.

> ⚠️ **Watch out**
>
> Commands in this book contain placeholders like `user_XXXX` and `YOUR_PASSWORD`. Replace them with the credentials from your own anyIP dashboard, pasting them verbatim is the #1 cause of a first-request 407 error (chapter 8 covers the rest).

The promise, and the fine print

Here is the deal, in one paragraph: give this book a few evenings and you will go from "my scraper keeps getting blocked" to running reliable, geo-accurate, cost-controlled data collection, with the judgment to know which jobs to run, the reflexes to fix them when they break, and the numbers to defend the spend. That's the promise. The bans stop being a mystery and start being an engineering parameter.

> 💼 **Business angle**
>
> Reliability compounds: a pipeline that succeeds 99% of the time instead of 70% doesn't just save retries, it makes the downstream data trustworthy enough to base decisions on. That's the real product of this book.

A note on responsible use

Every recipe assumes you have the right to access the target, and that you respect `robots.txt`, terms of service, rate limits, and the law. Chapter 9 turns that into a concrete pre-job checklist. If a job depends on defeating someone's security or anti-fraud controls, or on pretending to be users who don't exist, this is the wrong tool.

📋 What you need before you start

- [ ] An anyIP account with an active plan (sign up at anyip.io; credentials live in the dashboard).
- [ ] A terminal with `curl` installed, your test bench for chapters 4 to 6.
- [ ] Python 3.9+ with `pip` for the recipes in chapter 7 (optional until then).
- [ ] A target you are **authorized** to access, your own site, a public page whose terms permit collection, or a client's property with written permission.
- [ ] Fifteen quiet minutes for chapter 4, that's all the first successful proxied request takes.

> 📌 **Key takeaways**
>
> - A ban is a signal you can engineer around, not bad luck; this book gives you the mental model to do it.
> - The path has four legs: Foundations (2 to 3), Hands-on (4 to 8), Judgment & business (9 to 10), Reference (11 to 12), linear through chapter 4, jump-friendly after.
> - Eight callout boxes structure every chapter; reading only the boxes gives you a two-minute skim of any topic.
> - Everything here is for authorized use: robots.txt, terms of service, and the law are the frame, not an afterthought.

🔗 **Go deeper**

- anyIP homepage, product overview and signup.
- Docs: Introduction, the official starting point for everything chapters 3 to 8 expand on.
- Docs: Quick Start, the condensed version of chapter 4.
- Pricing, plan tiers, analyzed in depth in chapter 10.
- Status page, bookmark it before you need it.
- RFC 9309, the robots.txt standard, the baseline for the responsible-use rules in chapter 9.

2. The Ban Problem & the anyIP Answer

> ⚡ **The gist**
>
> - Sites block by **rate**, by **IP reputation**, and by **geography**, one IP hammering a site trips all three.
> - Not all IPs are equal: datacenter IPs are easily detected; residential and mobile IPs look like real people. anyIP routes through **real peer devices only**.
> - The pitch: one gateway (`portal.anyip.io`), everything controlled from the username string, ban rate <0.2% on major platforms, ~98.6% request success.
> - Figure 2.1 is the big schema, every later chapter zooms into a piece of it.
>
> *Reading time: ~9 min*

Why your requests get blocked

You wrote a clean script. It collects public prices from a catalog page you're authorized to monitor, or runs QA checks against your own app from another country. It works for ten minutes, then every response is a CAPTCHA, a 403, or a silently empty page. Nothing in your code changed. What changed is how the target site now scores *your IP address*.

Modern sites defend themselves along three axes, and a single fixed IP loses on all of them:

Defense	What the site checks	How a single IP fails
Rate limits	Requests per IP per time window	100 requests/min from one address is nothing like human browsing, throttled or blocked.
IP reputation	Who owns the IP range and its history	IPs registered to hosting providers, or previously flagged for abuse, are pre-scored as suspicious before the first request lands.
Geo-walls	The IP's geolocation	Content, prices, and search results differ by country, or are simply unavailable, outside the target region.

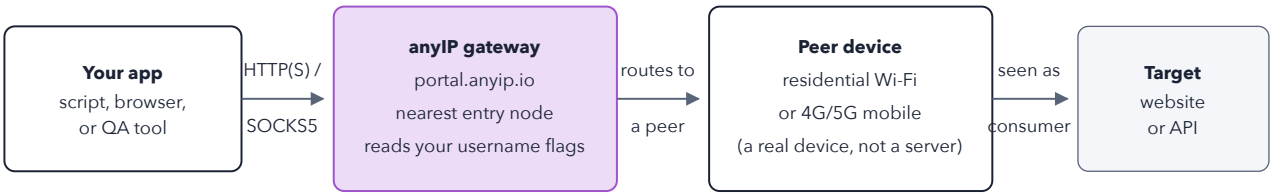
The fix is not a cleverer user-agent header. It's changing what the site sees at the network layer: many IPs instead of one (defeats rate accumulation), IPs that belong to real consumer networks (good reputation), located where you need them (passes geo-walls). That is exactly what a residential and mobile proxy network provides, and throughout this book we use it for **legitimate, authorized data collection and QA**: price monitoring, SEO checks, ad verification, and testing your own apps across geographies.

Where anyIP sits in your stack

anyIP is a layer between your application and the target site. Your code speaks HTTP, HTTPS, or SOCKS5 to a single gateway hostname, `portal.anyip.io` , which auto-routes to the entry node nearest you (regional nodes `portal-na` , `portal-eu` , `portal-as.anyip.io` exist if you want to pin one). The gateway forwards

your request through a **real peer device**, someone's home Wi-Fi connection or 4G/5G phone, and the target website sees that device's ordinary consumer IP, not yours and not a server farm's.

Keep this diagram in mind: the rest of the book zooms into its pieces. Chapter 3 covers how the username string steers the gateway; Chapter 5 covers which peer gets picked; Chapter 6 covers how long you keep it.



Regional entry nodes: portal-na / portal-eu / portal-as.anyip.io, the global hostname auto-picks the nearest.

Figure 2.1, The big schema: your app → anyIP gateway → real peer device → target site. Later chapters zoom into each hop.

The IP trust spectrum: datacenter → residential → mobile

Target sites don't just see *an* IP, they see who owns it. An IP registered to a hosting provider almost certainly isn't a human shopper, so **datacenter proxies** are fast and cheap but easily detected. A **residential proxy** routes through a real home broadband or Wi-Fi device, so the target sees an ordinary consumer IP. A **mobile proxy** routes through a real 4G/5G device and carries the highest trust score of all, carrier-grade NAT means many legitimate users share the same mobile IP, so blocking it means blocking real customers, at the cost of higher latency.

anyIP sits on the right-hand side of this spectrum by design: it uses real peer devices (people's phones and home Wi-Fi), never datacenter servers, and does not sell datacenter IPs at all. The default `mixed` pool auto-assigns residential or mobile for best availability; you can force either type when you need it.

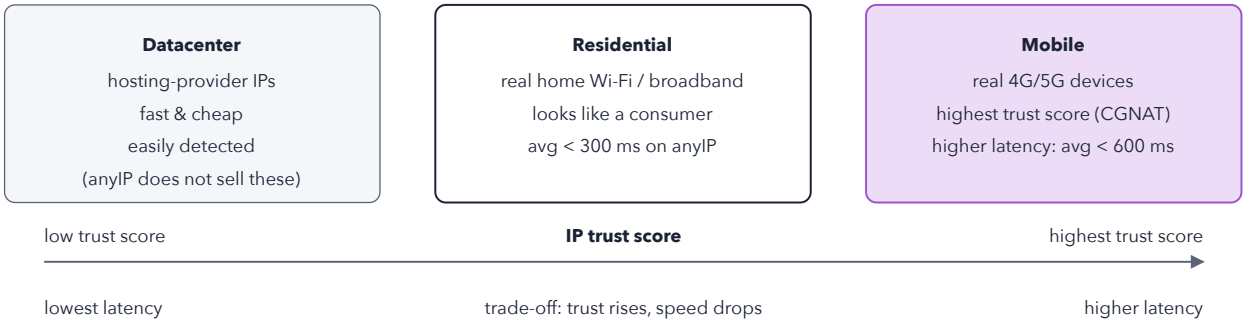


Figure 2.2, The IP trust spectrum. Moving right buys trust (fewer blocks) and costs latency; anyIP operates only in the residential and mobile bands.

The elevator pitch, in numbers

anyIP calls itself "the developer-first residential proxy network," and the developer-first part is concrete: *"You don't need to change ports or server addresses to change your settings. You control everything through your Username."* One hostname, one port, and a comma-separated flag string (Chapter 3) select network type, country, city, carrier, and session behavior. The headline claims, honestly sourced:

Claim	Figure	Source note
Ban rate on major platforms	**< 0.2%**	Docs technical spec
Request success rate	**~98.6%**	Homepage
IP pool	**100M+** residential & mobile IPs (marketed)	Homepage; the docs spec table states **30M+**, treat the pool as "tens of millions or more"
Response time	avg **< 300 ms** residential, **< 600 ms** mobile	Docs spec (homepage rounds this to "~0.6 s average")
Concurrency	Unlimited (bound only by plan traffic)	Docs spec
Protocols	HTTP, HTTPS, SOCKS5, same port	Docs

Pricing is by gigabyte of traffic, not by number of IPs or proxies, in a single plan covering both residential and mobile, Chapter 10 does the cost math. What matters here: the network is built from real consumer devices, which is precisely why its ban rate stays low.

> 💼 **Business angle**
>
> A blocked scrape isn't a technical annoyance, it's a data outage. If your price monitor gets banned on Tuesday, Wednesday's pricing decision is made on stale numbers; if your geo-QA can't reach the German version of your app, you ship a broken checkout to Germany. Reliable IP infrastructure is what keeps the data feeding your decisions complete and current.

Is a proxy network right for your use case?

Before you spend a gigabyte, check that the tool fits the job, and that the job is one you're allowed to run.

- [ ] I am authorized to access the target (my own app, public pages, or data I have permission to collect).
- [ ] My requests are getting rate-limited, reputation-blocked, or geo-walled from a single IP, the problem is at the network layer, not in my code.
- [ ] I need IPs that look like real consumers (residential) or carry maximum trust (mobile), a datacenter IP or simple VPN isn't enough.
- [ ] I need specific geographies (country, city, or carrier) for geo-accurate results.
- [ ] My volume is measured in GB of traffic, so per-GB pricing fits (Chapter 10).
- [ ] My use case is data collection, SEO/ad verification, market research, or QA, not accessing banks, government portals, or sending email, all of which anyIP blocks by design (Chapter 9).

📌 Key takeaways

- Sites block on three axes, request rate, IP reputation, and geography, and a single fixed IP eventually fails all three.
- The trust spectrum runs datacenter (fast, easily detected) → residential (consumer-grade trust) → mobile (highest trust, higher latency). anyIP uses only real peer devices, no datacenter IPs.
- Architecture in one line: your app → `portal.anyip.io` (nearest entry node) → real peer device → target site. Figure 2.1 is the map for the whole book.
- Key numbers: ban rate <0.2% on major platforms, ~98.6% success, avg <300 ms residential / <600 ms mobile, 100M+ marketed IPs (30M+ per docs spec).
- Everything is controlled from the username string, no port or server juggling. That mechanism is Chapter 3.

🔗 Go deeper

- anyip.io homepage, positioning, headline stats (100M+ IPs, ~98.6% success, ~0.6 s avg response).
- anyIP docs, Introduction, the smart-username model and network overview; the docs' technical spec table is the source for the <0.2% ban rate, 30M+ pool, and per-type response times.
- anyIP pricing, the single combined residential + mobile plan, billed by GB.

3. Core Concepts, The Smart Username

> ⚡ **The gist**
>
> - anyIP has one control surface: the **username string**. You never change ports or servers to change behavior, you append flags.
> - Six concepts cover the whole system: protocols, authentication, sessions, network type, location targeting, and configuration flags.
> - Default behavior is **rotating** (new IP every request); add `session_[name]` for a **sticky** IP that lasts up to 7 days.
> - Flags are separated by a comma `,` or a pipe `|`, use the pipe when your software rejects commas.
>
> *Reading time: ~9 min*

The mental model: one string, not settings

Most proxy services make you pick behavior with infrastructure: a different port for mobile, a different hostname per country, a dashboard toggle for rotation. anyIP collapses all of that into a single idea, stated plainly in its docs: *"You don't need to change ports or server addresses to change your settings. You control everything through your Username."*

Your username starts with your credential ID (`user_XXXX`) and grows by appending flags. Want a mobile IP in New York that stays pinned for an authorized multi-step QA workflow? You don't reconfigure anything, you write a longer username. The gateway (`portal.anyip.io`) parses the string on every connection and produces exactly the IP you described.

Once you can read one of these strings, you can predict what IP any anyIP configuration will produce. Here is the anatomy.

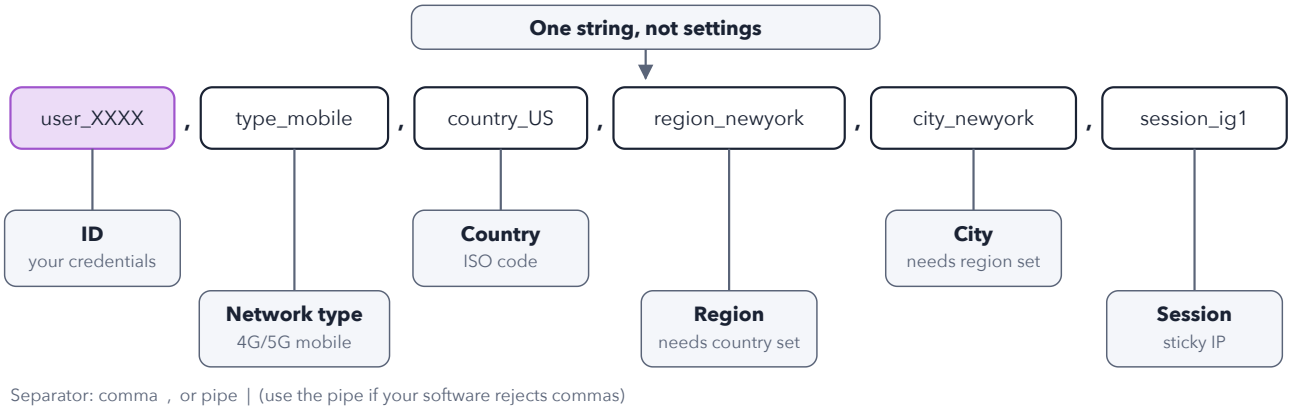


Figure 3.1, Anatomy of a Smart Username: one credential ID plus appended flags controls network type, location, and session, no port or server changes.

The six core concepts

Everything in the rest of this book builds on six ideas. Each one is controlled from the same two places: the connection string and, for auth, the port you pick.

1. Proxy formats

anyIP speaks **HTTP, HTTPS, and SOCKS5, all on the same port**. You don't choose a "SOCKS port"; you point whatever protocol your tool prefers at the same endpoint. (SOCKS5 matters later: it's the only path to UDP/HTTP-3, and it's what anti-detect browsers prefer.)

2. Authentication

Two methods, distinguished by port:

- **Username/password**, ports `1080` or `443`, format `user_ID:password`. Best for browsers, scrapers, and quick testing.
- **IP whitelisting**, ports `2000–2999`; you authorize your device's IP in the dashboard and connect with no credentials at all. Breaks if your home IP changes.

Ports 80 and 8080 are invalid and won't connect. Chapter 4 has the full decision tree.

3. Sessions

Rotating is the default: with no session flag, every request gets a fresh IP. Add `session_[name]` and the same IP sticks across requests for **up to 7 days** (tunable with `sesstime_[minutes]`, 1 to 10,080). Session names are 1 to 32 characters, letters and digits only.

4. Network types

Three values: the default **mixed** pool (anyIP auto-assigns residential or mobile for best availability), `type_residential` (force real home Wi-Fi/broadband IPs), or `type_mobile` (force real 4G/5G cellular IPs, highest trust, higher latency).

5. Location targeting

Pin the exit IP by **country** (`country_US`), **region** (`region_texas`, country must be set first), **city** (`city_dallas`, country and region must be set first), **carrier/ISP** (`asn_21928` = T-Mobile US), **pool** (`pool_europe`, one of 19 curated multi-country regions), or **GPS coordinates** (`lat_48.8584,lon_2.2945`, closest available peer, not guaranteed). Chapter 5 covers all of these in depth.

6. Configuration flags

Every concept above is expressed as a flag appended to `user_XXXX`, separated by commas or pipes. That's the entire configuration language, there is no config file and no per-setting endpoint.

Concept	How you control it	Default
Proxy format	HTTP/HTTPS or SOCKS5, your tool's choice, same port	,
Authentication	Port 1080/443 (user:pass) or 2000 to 2999 (IP whitelist)	user:pass
Session	`session_[name]`, `sesstime_[min]`	Rotating (new IP per request)
Network type	`type_residential` / `type_mobile`	Mixed
Location	`country_`, `region_`, `city_`, `asn_`, `pool_`, `lat_/lon_`	Anywhere
Configuration	Flags in the username, separated by `,` or `|`	`user_XXXX` alone

Rotating vs sticky in one picture

The single most consequential flag is `session_`. Without it, anyIP behaves like a firehose of fresh identities, ideal for bulk collection of public pages you're authorized to gather. With it, you hold one identity steady, what a login flow or multi-step QA scenario needs. A sticky session is *not* a static IP: the peer is a real phone or home router, and if it goes offline anyIP smart-replaces it (Chapter 6 covers the strict flags that control this).

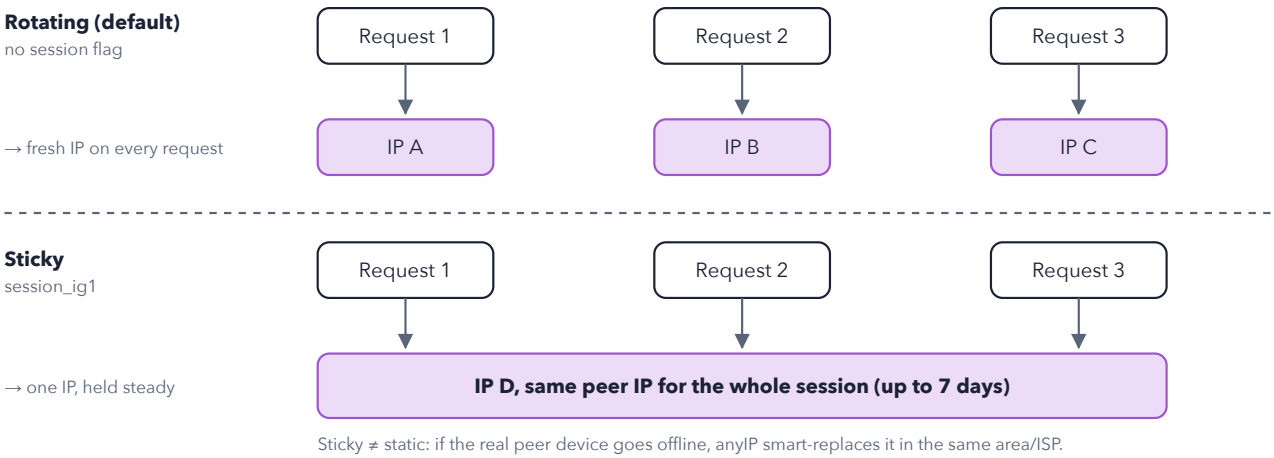


Figure 3.2, Rotating vs Sticky: without a session flag every request exits from a new IP; with `session_[name]` the same IP holds for up to 7 days.

> 💼 **Business angle**
>
> Because configuration lives in one string, switching a job from "rotating German residential" to "sticky US mobile" is a one-line code change, no new endpoints to provision, no dashboard round-trips, no waiting on ops. That collapses experiment time for data teams from hours to seconds, and it means one credential serves every use case your team is authorized to run.

Glossary, the terms this book uses

Term	Meaning
Smart Username / flags	anyIP's control system: comma- or pipe-separated instructions appended to your username set network type, location, session, and more.
Residential proxy	Routes your request through a real home broadband/Wi-Fi device, so the target sees an ordinary consumer IP.
Mobile proxy	Routes through a real 4G/5G cellular device. Highest trust score (carrier-grade NAT means many users share the IP), higher latency.
Mixed pool	The default: anyIP auto-assigns residential or mobile for best availability.
Rotating proxy	A new IP on every request, anyIP's behavior when no `session_` flag is set.
Sticky session	The same IP held across requests for up to 7 days via `session_[name]`. Not static.
Smart replacement	When a sticky peer drops offline, anyIP silently assigns a new IP in the same area/ISP (disable with `sessreplace_false`).
ASN	Autonomous System Number, identifies an ISP/carrier's network; target it with `asn_[number]` (T-Mobile US = 21928).
Pool	A curated multi-country region targeted with one flag (e.g. `pool_europe`); 19 pools total.
IP whitelisting	Authentication by authorizing your device's IP (ports 2000 to 2999) instead of sending a password.
SOCKS5	A versatile proxy protocol supporting TCP and (via UDP Associate) UDP; required for anyIP's UDP/HTTP-3 feature.

📋 Anatomy checklist, every flag you can append

- [ ] `type_residential` / `type_mobile`, force Wi-Fi/broadband or 4G/5G (omit for mixed).
- [ ] `country_[ISO]`, e.g. `country_FR`.
- [ ] `region_[name]`, requires country; e.g. `country_US,region_texas`.
- [ ] `city_[name]`, requires country + region; e.g. `country_US,region_texas,city_dallas`.
- [ ] `asn_[number]`, pin an ISP/carrier; e.g. `country_US,asn_21928`.
- [ ] `pool_[name]`, one of 19 regional pools; e.g. `pool_western`, `pool_europe`.
- [ ] `lat_[x],lon_[y]`, GPS geofencing to the closest available peer.
- [ ] `session_[name]`, sticky IP (1 to 32 chars, letters and digits).
- [ ] `sesstime_[min]`, sticky duration, 1 to 10,080 minutes (default up to 7 days).
- [ ] `sessreplace_false`, fail instead of rotating if the sticky IP drops.
- [ ] `sessasn_strict`, replacement IP must keep the same ASN/ISP.
- [ ] `sessipcollision_strict`, guarantee no two of your sessions share an IP.

📌 Key takeaways

- anyIP's entire configuration surface is the username string: `user_XXXX` plus appended flags, separated by commas or pipes.
- HTTP, HTTPS, and SOCKS5 all work on the same port; auth method is chosen by port (1080/443 for user:pass, 2000 to 2999 for IP whitelist).
- Rotating is the default (new IP per request); `session_[name]` makes it sticky for up to 7 days, sticky, not static.
- Network type is mixed by default; force it with `type_residential` or `type_mobile`.
- Location targeting goes from country down to GPS coordinates, with region requiring country, and city requiring both.

🔗 Go deeper

- anyIP documentation, Introduction, the "control everything through your Username" model.
- anyIP docs, Configuration Reference, the authoritative flag cheat sheet behind Figure 3.1 and the checklist above.
- anyIP docs, Sessions & Rotation, sticky sessions, `sesstime`, and smart replacement (expanded in Chapter 6).

4. Getting Started, Your First Proxy in Under a Minute

> ⚡ **The gist**
>
> - Everything connects through one hostname: `portal.anyip.io`, port `1080` (or `443`), HTTP, HTTPS, and SOCKS5 all on the same port.
> - Your credentials live in **Dashboard > Proxies > Get Proxy Details**: a username like `user_XXXX` plus a password.
> - One curl command proves you're live: it returns a JSON block with an IP that isn't yours.
> - Tool can't send a username/password? Use IP whitelisting on ports 2000 to 2999 instead.
>
> *Reading time: ~7 min*

Prerequisites: two things must be true

Before any request will go through, your account needs an **active plan**, and "active" means two conditions at once: you need **both** remaining subscription time **and** remaining GB. A plan with days left but 0 GB won't connect; neither will leftover GB on an expired plan. If your first connection mysteriously fails, check both meters before debugging anything else.

Second, grab your credentials. In the dashboard, go to **Proxies > Get Proxy Details** and copy two values: your proxy username (it looks like `user_XXXX`) and your proxy password. That pair is all you need for the rest of this book, every setting from Chapter 3 onward is a flag appended to that username.

Build the connection string

The single-line form most tools accept:

```
http://user_YOURID:PASSWORD@portal.anyip.io:1080
```

If your tool asks for separate fields instead, fill them in like this:

Field	Value
Protocol	HTTP/HTTPS or SOCKS5 (all on the same port)
Host	`portal.anyip.io`
Port	`1080` or `443`
Username	`user_XXXX` (+ optional flags)
Password	your proxy password

Which auth method? Decide in one glance

anyIP supports two authentication methods, and the port you connect to selects between them. Standard username/password auth (ports 1080/443) is the right choice for browsers, bots, scrapers, and quick testing. IP whitelisting (ports 2000 to 2999) exists for tools that can't send proxy credentials at all: you register your device's real IP in the dashboard, and then connect with no password.

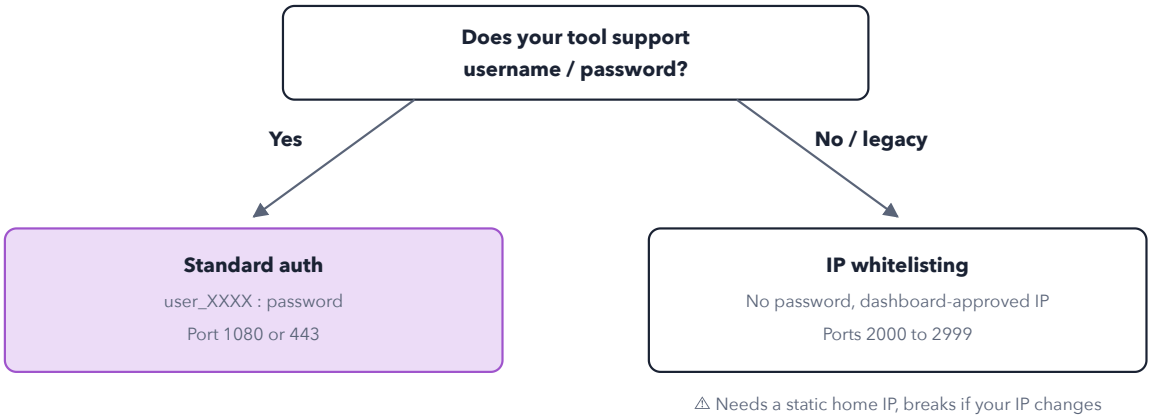


Figure 4.1, Auth decision tree: standard username/password auth on 1080/443, or IP whitelisting on 2000 to 2999 for tools that can't send credentials.

To set up whitelisting: **Proxies → Get Proxy Details → IP Whitelisting tab → Enable**, add your real IP, assign a port (e.g. 2000), and **Save**. Then connect to `http://portal.anyip.io:2000` with no credentials. The caveat baked into Figure 4.1 is real: if your home connection has a dynamic IP, whitelisting silently breaks the day your ISP rotates it, prefer standard auth whenever your tool allows it.

Recipe: zero to first proxied request

📝 Recipe 4.1, Zero to first proxied request

Prerequisites: an active plan (subscription time *and* GB remaining), a terminal with `curl`.

1. Open the dashboard and go to **Proxies > Get Proxy Details**. Copy your username (`user_XXXX`) and password.

   ✅ Expected result: you have a username of the form `user_XXXX` and a password on your clipboard.

2. (Optional) Choose a network type by appending a flag to the username: `type_residential` for Wi-Fi/broadband IPs or `type_mobile` for 4G/5G. No flag = mixed pool, which is fine for a first test.

3. Assemble the connection string, substituting your own credentials:

   ```
   http://user_YOURID:PASSWORD@portal.anyip.io:1080
   ```

4. Fire a test request through the proxy:

   ```
   curl -x "http://user_XXXX:PASSWORD@portal.anyip.io:1080" http://ip-api.com/json
   ```

   ✅ Expected result: a JSON response showing an IP address different from your home internet, plus a country. That IP belongs to a peer on the anyIP network, you're connected.

5. Run the same command again. Because rotating mode is the default, you'll typically see a different IP on each request, that's the rotation working, not a bug.

   ✅ Expected result: a second JSON response, usually with a different IP than step 4.

Set it up in your tools

The same host/port/credentials plug into whatever you already use. Quick settings for the common cases:

Tool	Settings	Notes
FoxyProxy (browser extension)	Host `portal.anyip.io`, port `1080`, SOCKS5 or HTTP, username + password	Add flags directly in the username field to change targeting.
Anti-detect browser (AdsPower, Dolphin)	Same host/port/credentials in the profile's proxy settings	One proxy profile per browser profile; use a distinct `session_` flag per profile (Chapter 6).
iOS (Shadowrocket, Potatso)	Same host/port/credentials	**Turn OFF iCloud Private Relay**, it conflicts with the proxy.

> ⚠️ **Watch out**
>
> **The classic first error: 407 Proxy Authentication Required.** A 407 means one of two things: your credentials are wrong (re-copy them from **Proxies > Get Proxy Details**, no stray spaces, exact `user_XXXX` spelling), or you're using the wrong port for your auth method. Username/password only works on **1080/443**; whitelisted-IP connections only work on your assigned port in **2000 to 2999**. Mixing them, credentials on a whitelist port, or a whitelist connection on 1080, earns you a 407 every time. Also note that ports 80 and 8080 are invalid and simply won't connect. anyIP tells you exactly what went wrong in the `X-Anyip-Error` response header; Chapter 8 maps every code.

> 💼 **Business angle**
>
> Setup cost matters when you're evaluating vendors: here the integration is one connection string, so an engineer can go from signup to a verified geo-distributed request in minutes, no SDK, no server list to sync, no per-tool custom work. That keeps proof-of-concept cost near zero before you commit budget.

📋 First-connection checklist

- [ ] My plan shows both active subscription time and remaining GB.
- [ ] I copied `user_XXXX` and the password from Dashboard > Proxies > Get Proxy Details.
- [ ] I know which auth method my tool needs (user/pass → 1080/443; whitelist → 2000 to 2999).
- [ ] The curl test returned a JSON IP + country different from my home connection.
- [ ] Running the test twice showed rotation (different IPs) as expected.
- [ ] On iOS, iCloud Private Relay is switched off.

> 📌 **Key takeaways**
>
> - Connecting requires both an unexpired plan *and* remaining GB, check both before debugging.
> - One string does it all: `http://user_YOURID:PASSWORD@portal.anyip.io:1080` , with HTTP/HTTPS and SOCKS5 on the same port.
> - Port selects auth method: 1080/443 for username/password, 2000 to 2999 for IP whitelisting (static home IP required).
> - Verify with `curl -x … http://ip-api.com/json` , a foreign IP + country in the JSON means you're live.
> - A 407 is almost always wrong credentials or a port/auth-method mismatch.

🔗 **Go deeper**

- anyIP documentation, Quick Start, the official 5-step version of Recipe 4.1.
- anyIP documentation, Authentication Methods, standard auth vs IP whitelisting, with dashboard setup steps.
- anyIP Dashboard, where your credentials, whitelist, and plan meters live.
- ip-api.com/json, the free IP-echo endpoint used for verification throughout this book.

5. Targeting, Country, City, Carrier, Coordinates

⚡ The gist

- Pin an IP to a **country, region, or city** with three flags, but each level requires its parent (`city_` needs `country_` + `region_`).
- Target a specific **ISP or mobile carrier** with `asn_[number]` (e.g. T-Mobile US = `asn_21928`).
- Cover a whole region at once with one of **19 named pools** (`pool_europe` = 50 countries).
- Go hyper-local with GPS: `lat_[x],lon_[y]` picks the closest available peer.
- If your software chokes on commas, use the pipe separator: `user_123|country_US` .

Reading time: ~9 min

The hierarchy: country → region → city

Every location flag rides in the same smart username you met in Chapter 3. The three geographic levels nest, and anyIP enforces a strict **must-specify-parent rule**: you cannot ask for a region without naming its country, and you cannot ask for a city without naming both country and region. Skip a parent and the gateway rejects the request with a 407 (`region_country_is_missing` or `city_region_is_missing`) instead of guessing.

- **Country**, `country_[ISO]` , e.g. `user_XXXX,country_FR`
- **Region**, `region_[name]` , e.g. `country_US,region_texas` (country required first)
- **City**, `city_[name]` , e.g. `country_US,region_texas,city_dallas` (country + region required)

A country-only request is the workhorse. Here it is end to end:

```
curl -x "http://user_XXXX,country_FR:pass@portal.anyip.io:1080" http://ip-api.com/json
```

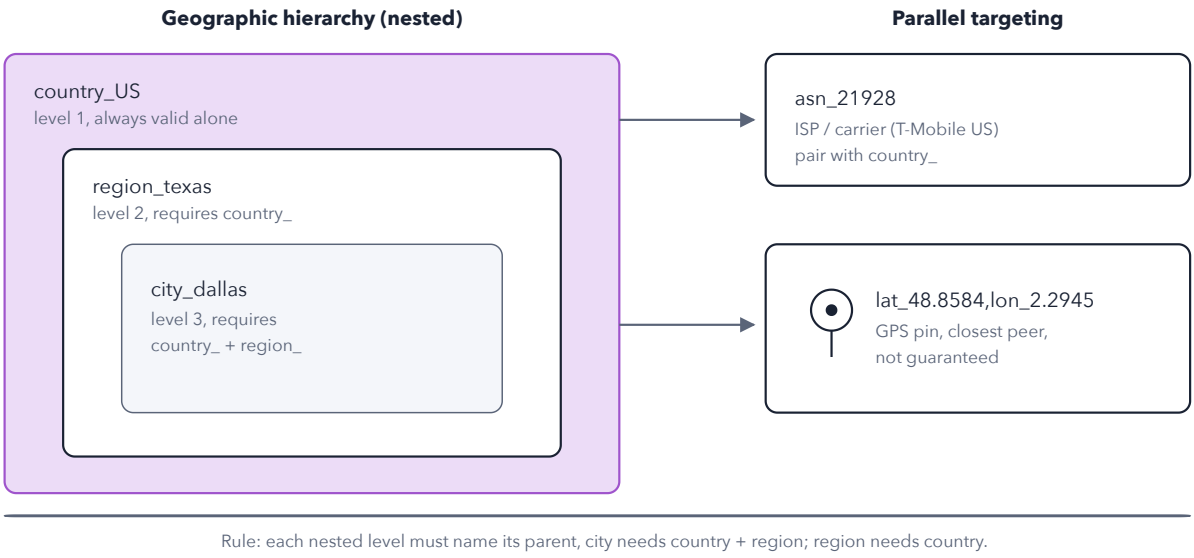


Figure 5.1, Targeting hierarchy: geography nests (country ⊃ region ⊃ city); ASN and GPS run in parallel.

📝 Recipe 5.1, Get a German residential IP for authorized price monitoring

Prerequisites: active anyIP plan, your `user_XXXX` credentials, authorization to monitor the target catalog.

1. Build the username with a country flag and a strict-residential flag:

```
user_XXXX,country_DE,type_residential
```

2. Send a test request through the gateway:

```
curl -x "http://user_XXXX,country_DE,type_residential:PASSWORD@portal.anyip.io:1080"
http://ip-api.com/json
```

3. Check the JSON response: the `countryCode` field should read `DE`. Re-run it, the default rotating mode gives you a fresh German residential IP each time.

✅ Expected result: `"countryCode": "DE"` in the ip-api response, on a residential IP that changes per request.

ASN targeting: pin an ISP or carrier

Geography answers *where*; ASN answers *whose network*. Add `asn_[number]` to land on a specific ISP or mobile carrier, the go-to move when you need to see exactly what a given carrier's subscribers see, for example when QA-testing your own app on T-Mobile versus Verizon. Pair it with a country flag, as in `country_US,asn_21928`.

Carrier / ISP	Country	ASN flag
T-Mobile	US	`asn_21928`
Verizon	US	`asn_701`
AT&T	US	`asn_7018`
Vodafone	UK	`asn_13285`

📝 Recipe 5.2, Pin to a US carrier (T-Mobile, ASN 21928)

Prerequisites: active anyIP plan, your `user_XXXX` credentials.

1. Build the username with country + ASN flags:

```
user_XXXX,country_US,asn_21928
```

2. Send a test request:

```
curl -x "http://user_XXXX,country_US,asn_21928:PASSWORD@portal.anyip.io:1080" http://ip-api.com/json
```

3. Inspect the response's organization/AS fields: they should identify T-Mobile / AS21928.

✅ Expected result: ip-api's `org` / `as` fields name T-Mobile (AS21928), with `countryCode: US`.

Pools: one flag, a whole region

When "any IP in Europe" is precise enough, don't juggle country lists, use a **regional pool**. A single `pool_[name]` flag draws from every country in that group; `pool_europe` alone spans 50 countries. anyIP ships 19 pools, grouped by continent below. Broader targeting also means more available peers, so pools are the pragmatic fix when a tight city or ASN filter starts returning `no_peers_available`.

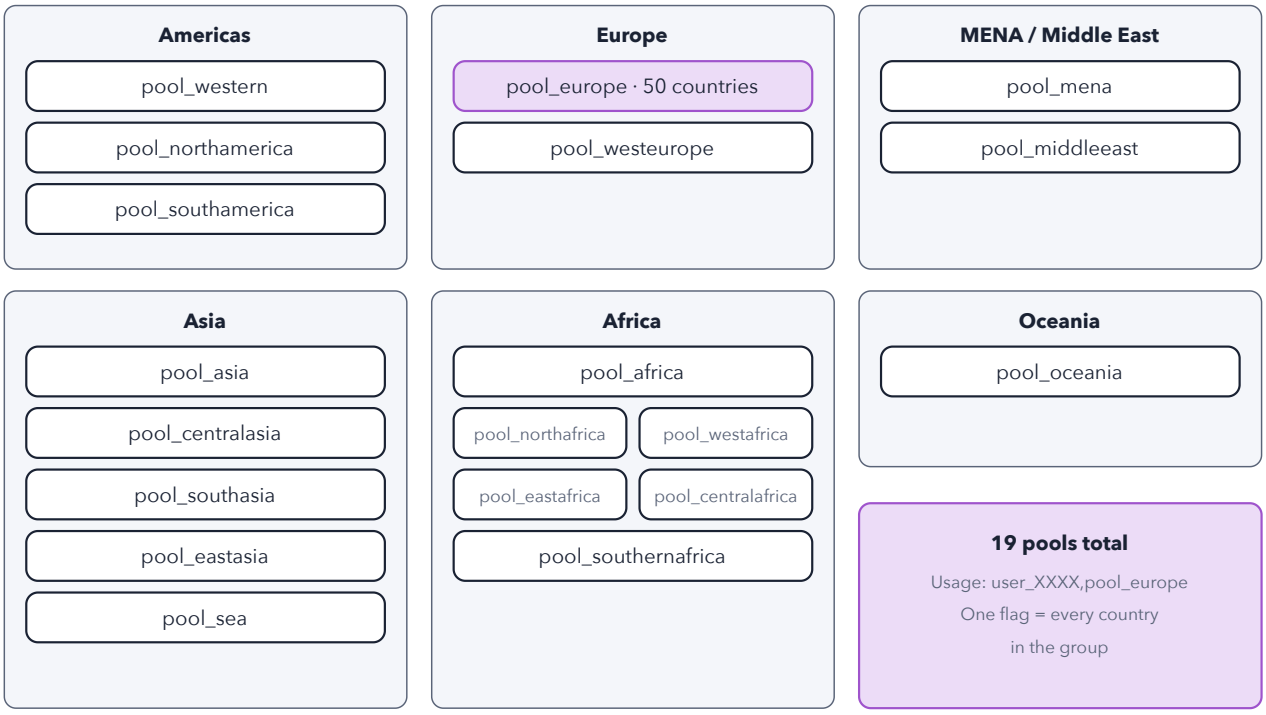


Figure 5.2, The 19 regional pools grouped by continent. One `pool_` flag covers the whole group; `pool_europe` alone spans 50 countries.

GPS coordinates: hyper-local targeting

For point-precision work, drop latitude and longitude flags into the username: `lat_[x],lon_[y]`. For example, `user_XXXX,lat_48.8584,lon_2.2945` requests a peer near the Eiffel Tower. anyIP connects you to the **closest available peer** to those coordinates, proximity is best-effort, not guaranteed, since it depends on which real devices are online nearby. Use it when city-level targeting is too coarse, such as verifying location-based ads or store-locator results for a specific neighborhood.

> ⚠️ **Watch out**
>
> **The comma trap.** Flags are joined with commas by default (`user_XXXX,country_FR`), but some proxy-config fields and third-party tools reject commas in usernames. anyIP accepts the pipe `|` as an equivalent separator, switch to `user_123|country_US` and every flag in this chapter works unchanged. If a targeting flag "mysteriously" does nothing in a GUI tool, a swallowed comma is the first thing to check.

> 💼 **Business angle**
>
> Geo-targeting turns one subscription into a worldwide field team: check localized prices in Germany, verify your ad placements on T-Mobile US, and audit SERPs from 50 European countries, all authorized checks on your own campaigns and markets, without opening offices or shipping test phones anywhere.

📋 Targeting flags quick-reference

- [ ] `country_[ISO]`, pin a country (e.g. `country_FR`); valid on its own.
- [ ] `region_[name]`, pin a state/region; must set `country_` first.
- [ ] `city_[name]`, pin a city; must set `country_` + `region_` first.
- [ ] `asn_[number]`, pin an ISP/carrier (T-Mobile US 21928, Verizon 701, AT&T 7018, Vodafone UK 13285).
- [ ] `pool_[name]`, draw from one of 19 regional pools (e.g. `pool_europe`, `pool_northamerica`).
- [ ] `lat_[x],lon_[y]`, GPS geofencing; closest available peer, not guaranteed.
- [ ] Separator: comma `,` by default; pipe `|` if your software rejects commas.

📌 Key takeaways

- Geography nests: `country_` stands alone, `region_` needs country, `city_` needs country + region, skip a parent and the gateway returns a 407 instead of guessing.
- `asn_[number]` pins a specific ISP or carrier (T-Mobile US = 21928), the tool for carrier-specific QA and ad verification.
- 19 named `pool_` flags cover whole regions in one shot; `pool_europe` spans 50 countries and pools ease peer availability.
- `lat_/lon_` requests the closest available peer to a coordinate, best-effort, not guaranteed.
- Commas are the default flag separator, but the pipe `|` is a drop-in replacement for comma-hostile software.

🔗 Go deeper

- anyIP documentation, Location Targeting, Regional Pools, Supported Countries, and the Configuration Reference behind every flag in this chapter.
- anyIP Configuration Reference, the full flag cheat sheet, including the comma/pipe separator rule.
- ip-api.com, the free geo-IP endpoint used to verify country, city, and ASN in Recipes 5.1 and 5.2.

6. Sessions & Rotation, Sticky Without the Surprises

> ⚡ **The gist**
>
> - **Rotating is the default**, no session flag means a fresh IP on every request. Best for bulk scraping.
> - Add `session_[name]` to go **sticky**: the same IP stays with you for up to **7 days**.
> - Sticky is **not static**, peers are real phones and home Wi-Fi that go offline. Smart replacement quietly swaps in a new IP in the same area/ISP.
> - Strict flags (`sessreplace_false` , `sessasn_strict` , `sessipcollision_strict`) let you trade convenience for guarantees.
> - Two ways to rotate on demand: rename the session, or call the IP-change URL.
>
> *Reading time: ~9 min*

Rotating mode, the default

If you send no session flag, anyIP is in **rotating** mode: every request goes out on a new IP. This is exactly what you want for bulk data collection over public pages, each hit looks like a different, unrelated visitor, and you never have to manage state.

```
# Rotating (default), no session flag, new IP per request
curl -x "http://user_XXXX:pass@portal.anyip.io:1080" http://ip-api.com/json
```

Run that three times and you will usually see three different IPs. Nothing to configure, nothing to expire. Rotating is the mode you reach for when you are fanning out across many URLs and no single request depends on the one before it.

Sticky sessions, keep one IP for a task

Some authorized jobs need *the same* IP across several steps: a multi-page QA walkthrough, a login-and-verify flow you are permitted to run, a checkout you are testing end to end. For that, add `session_[name]` . anyIP pins your traffic to one peer IP and keeps it for **up to 7 days**.

```
# Sticky, same IP across requests
curl -x "http://user_XXXX,session_myprofile:pass@portal.anyip.io:1080" http://ip-api.com/json
```

Session names are your own labels: pick anything and reuse it to return to the same IP. Want a shorter window? Set a custom duration with `sesstime_[min]` , valid from **1 to 10080** minutes (1 minute to 1 week). For example `sesstime_10` expires the session after 10 minutes.

```
# Sticky with a custom 10-minute lifetime
curl -x "http://user_XXXX,session_myprofile,sesstime_10:pass@portal.anyip.io:1080" http://ip-api.com/json
```

> ⚠ **Watch out**
>
> **Sticky is NOT static.** anyIP's peers are real devices, people's phones and home Wi-Fi, never datacenter servers. A peer can drop off the network at any moment. Sticky means "prefer the same IP and keep it as long as it's reachable," not "guaranteed this exact IP for 7 days straight." Design your workflow to tolerate a mid-session IP change unless you explicitly opt into `sessreplace_false` (below).

Smart replacement & the strict flags

When your sticky peer goes offline, the default behaviour is **smart replacement**: anyIP assigns a new IP in the *same area and ISP* so your session continues with as little disruption as possible. For most authorized multi-step work that is exactly what you want, the workflow keeps running instead of erroring out.

When you need firmer guarantees, three strict flags change the trade-off:

Flag	What it guarantees	Use it when
`sessreplace_false`	No silent swap. On disconnect the request **fails with** `peer_not_found` instead of rotating to a new IP.	You would rather know the IP dropped than continue on a different one.
`sessasn_strict`	Any replacement IP stays on the **same ASN/ISP**.	The carrier or ISP itself matters to what you're testing.
`sessipcollision_strict`	Each session is guaranteed a **unique IP**, no two of your sessions ever share one.	You run many parallel sessions (e.g. multi-account managers) and need them cleanly separated.

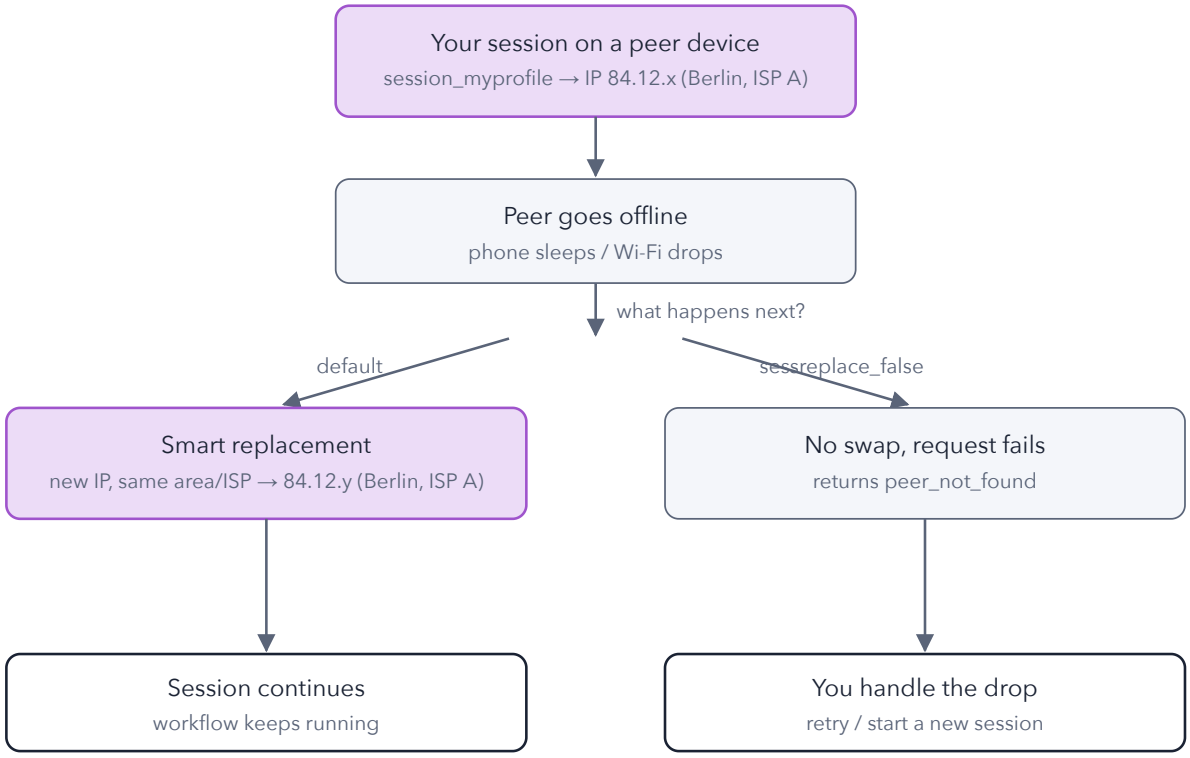


Figure 6.1, Sticky ≠ Static: when a peer drops, smart replacement keeps the same area/ISP by default, while `sessreplace_false` fails instead.

> 💼 **Business angle**
>
> Smart replacement is a reliability feature, not a loophole. For authorized QA and multi-step data jobs it means a dropped peer doesn't kill your run halfway through, fewer wasted GB on failed sessions, fewer manual restarts, and predictable throughput your team can plan around.

Rotating on demand

Sometimes you want a fresh IP *right now*, without waiting for the session to expire. There are two ways to force it.

Path A, rename the session. Session names *are* the identity of the IP, so changing the name gives you a brand-new IP instantly. Go from `session_profile1` to `session_profile1_v2` and you're on a new peer.

Path B, call the IP-change URL. anyIP exposes a rotation link that invalidates the current IP for a named session:

```
https://dashboard.anyip.io/api/proxy_accounts/[proxy_id]/invalidate/[hash]/session/[session_name]
```

Call it in a browser or with curl. It **requires** that your request use a `session_[name]` flag, and the session name in the URL must **match** the one you're connecting with. Rename is the simplest lever; the URL is the one to wire into a script when you want to trigger rotation programmatically without changing your connection string.

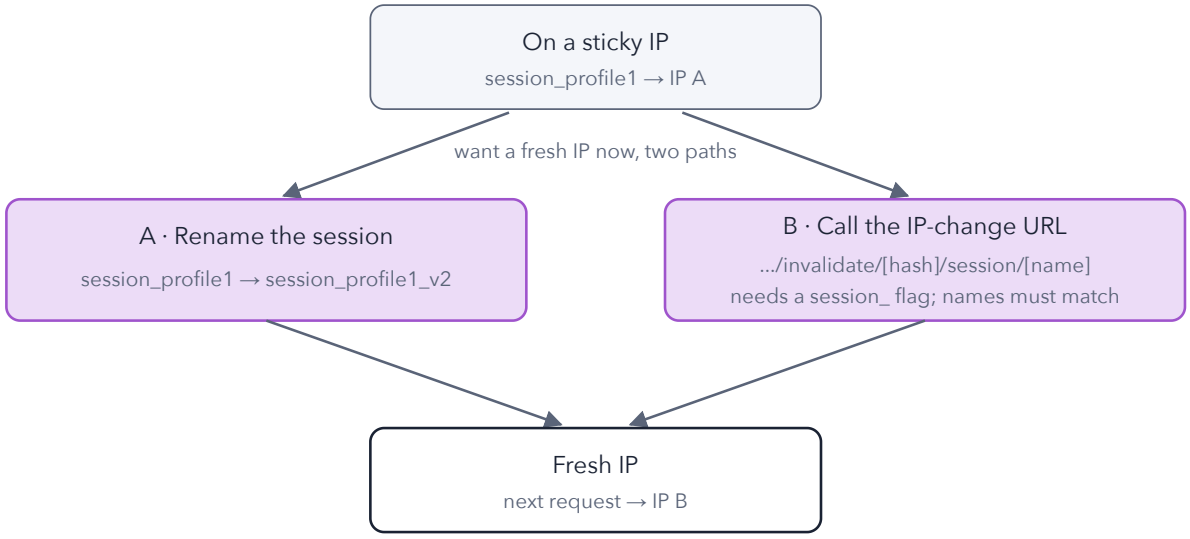


Figure 6.2, Rotate on demand: rename the session for an instant new IP, or call the change-IP URL to invalidate the current one programmatically.

Recipe, hold one IP across a workflow

📝 Recipe 6.1, Keep one IP across a multi-step authorized workflow

Prerequisites: an active plan, your `user_XXXX` credentials, and a target you are authorized to access across several steps.

1. Pick a session name and add it to your username with `session_[name]` :

```
PX="http://user_XXXX,session_myprofile:pass@portal.anyip.io:1080"
```

2. Run your first request and note the IP:

```
curl -x "$PX" http://ip-api.com/json
```

3. Run the next two steps of your workflow on the *same* session string:

```
curl -x "$PX" http://ip-api.com/json
curl -x "$PX" http://ip-api.com/json
```

4. Confirm all three responses report the same IP.

> ✅ Expected result: the same IP address appears in all three responses, the session is sticky. (If a peer drops mid-run, smart replacement keeps you in the same area/ISP; add `sessreplace_false` if you'd rather the request fail than switch.)

Recipe, force a fresh IP

📝 Recipe 6.2, Force a fresh IP via the rotation URL

Prerequisites: a live sticky session from Recipe 6.1, your `proxy_id` and rotation `hash` from the dashboard, and the same session name you're connecting with.

1. Call the IP-change URL for your session (the name must match your connection string):

```
curl
"https://dashboard.anyip.io/api/proxy_accounts/[proxy_id]/invalidate/[hash]/session/myprofile"
```

2. Re-run a request on the same session string:

```
curl -x "http://user_XXXX,session_myprofile:pass@portal.anyip.io:1080" http://ip-api.com/json
```

3. Compare the IP to the one from before the call.

✅ Expected result: a different IP than the pre-invalidate value, the session kept its name but was assigned a new peer. (No `session_` flag on the request = nothing to invalidate.)

📋 Choosing rotating vs sticky

- [ ] My job is bulk collection of independent pages → **rotating** (no session flag).
- [ ] My job is a multi-step flow that must stay on one identity → **sticky** (`session_[name]`).
- [ ] I need the IP for a fixed window → set `sesstime_[min]` (1 to 10080).
- [ ] A mid-run IP swap would corrupt my results → add `sessreplace_false` and handle `peer_not_found`.
- [ ] The exact ISP/carrier matters → add `sessasn_strict`.
- [ ] I run many parallel sessions and they must never share an IP → add `sessipcollision_strict`.
- [ ] I need a fresh IP mid-session → rename the session or call the IP-change URL.

📌 Key takeaways

- **Rotating** (default, no flag) = new IP per request, ideal for bulk scraping; **sticky** (`session_[name]`) holds one IP up to 7 days.
- Sticky ≠ static: peers are real devices, so **smart replacement** swaps in a same-area/ISP IP when one drops, unless `sessreplace_false` makes it fail with `peer_not_found` instead.
- `sessasn_strict` keeps the same ASN on replacement; `sessipcollision_strict` guarantees each session a unique IP.
- Tune the window with `sesstime_[min]` (1 to 10080); rotate on demand by renaming the session or calling the invalidate URL (needs a matching `session_` flag).

🔗 **Go deeper**

- anyIP docs, Sessions & Rotation, sticky sessions, smart replacement, and the strict flags.
- anyIP docs, Configuration Reference, full flag cheat sheet including `session_`, `sesstime_`, and the `sess*_strict` options.
- ip-api.com, the verification endpoint used to confirm the IP across requests in both recipes.

7. Real-World Recipes, Reliable Data Collection

> ⚡ **The gist**
>
> - Three end-to-end Python recipes for **authorized** work: a rotating residential scrape, multi-country SEO/ad checks, and a sticky mobile QA run.
> - The proxy is just a `proxies` dict on your existing HTTP client, one string controls the exit IP.
> - Retry with **backoff**, never a `while True` loop: a bad retry can fire 10,000 req/s and burn 10 GB on error pages.
> - Disable image/video loading to cut data use by up to 80%.
>
> *Reading time: ~11 min*

Before you run anything

Every recipe in this chapter assumes you are collecting **public data you're authorized to collect**, running QA against **your own systems**, or doing market research within the target's terms. Only scrape what you're authorized to; respect `robots.txt` and published rate limits. We cover the ethics and compliance checklist in depth in **Chapter 9**, treat it as a prerequisite, not an appendix.

The mechanics are identical across all three jobs: you attach anyIP as a standard HTTP proxy, and the *smart username* decides which network and which exit IP each request uses. In Python that's a two-line `proxies` dict passed to `requests`:

```
proxies = {
  "http": "http://user_XXXX,country_US:PASSWORD@portal.anyip.io:1080",
  "https": "http://user_XXXX,country_US:PASSWORD@portal.anyip.io:1080",
}
r = requests.get("http://ip-api.com/json", proxies=proxies, timeout=10)
```

No session flag means **rotating** mode: every request gets a fresh IP. That's exactly what you want for bulk scraping. Add `type_residential` or `type_mobile` to pin the network, `country_XX` to pin geography, and `session_[name]` only when you need one IP to stick across a flow.

How a reliable scraper is wired

A production scraper is a small pool of workers pulling from a URL queue, each request going out through the rotating proxy, with a retry/backoff loop that *slows down* on failure instead of hammering. Figure 7.1 shows the shape.

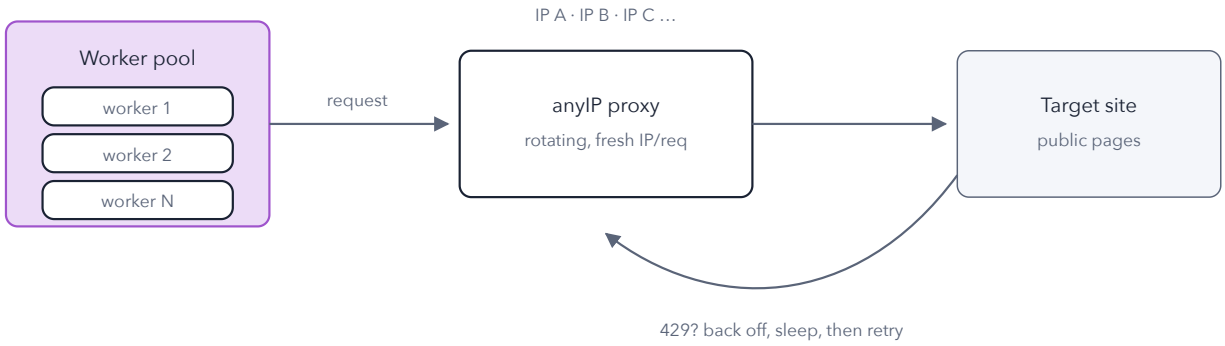


Figure 7.1, Scraper architecture: a worker pool sends each request through the rotating anyIP proxy to the target; on a 429 the worker backs off and retries instead of hammering.

> ⚠ **Watch out**
>
> **Never use a `while True: retry()` loop with no delay.** As the anyIP docs put it: "Your script fires 10,000 requests per second for an hour, burning 10 GB of data on error pages alone." Every failed request that reached anyIP's server still counts against your GB. Always sleep between attempts and cap your retries (see the backoff pattern in R7.1).

Recipe A, Rotating residential scrape of a public catalog

📝 Recipe 7.1, Rotating residential scrape of a public price/catalog page

Prerequisites: active anyIP plan with remaining GB, Python 3 with `requests`, and authorization to collect the target's public data (check its `robots.txt` and rate limits first).

1. Build a **rotating residential** proxies dict, use `type_residential` and *no* `session_` flag so every request draws a fresh IP.
2. Assemble the list of public catalog page URLs you're authorized to fetch.
3. Loop over the pages, sending each through the proxy with a sane `timeout`.
4. Wrap each fetch in a retry-with-backoff helper, sleep and grow the delay on failure; give up after a few attempts. **No `while True`.**
5. Add a small `time.sleep()` between pages to respect the target's rate limits.
6. Parse and collect rows; log the exit IP occasionally to confirm rotation is working.

```
import requests, time

PROXY = "http://user_XXXX,type_residential,country_US:PASSWORD@portal.anyip.io:1080"
proxies = {"http": PROXY, "https": PROXY}

def fetch(url, attempts=4):
    delay = 2
    for i in range(attempts):                # bounded, NOT while True
        try:
            r = requests.get(url, proxies=proxies, timeout=10)
            if r.status_code == 429:         # rate limited: back off
                time.sleep(delay); delay *= 2
                continue
            r.raise_for_status()
            return r.text
        except requests.RequestException:
            time.sleep(delay); delay *= 2    # exponential backoff
    return None                              # give up cleanly

rows = []
for page in range(1, 21):
    html = fetch(f"https://example.com/catalog?page={page}")
    if html:
        rows.extend(parse_rows(html))        # your parser
    time.sleep(1)                            # respect rate limits

print(f"collected {len(rows)} rows")
```

✅ Expected result: rows collected from every reachable page, and a spot-check of `http://ip-api.com/json` through the same proxy shows **different exit IPs across requests**, confirming rotation.

Recipe B, Checking the same page from many countries

SEO rankings, ad placements, and localized pricing change by geography. To verify them you fetch the *same* URL through a series of country flags and compare the responses. One script fans out across N exit countries, as in Figure 7.2.

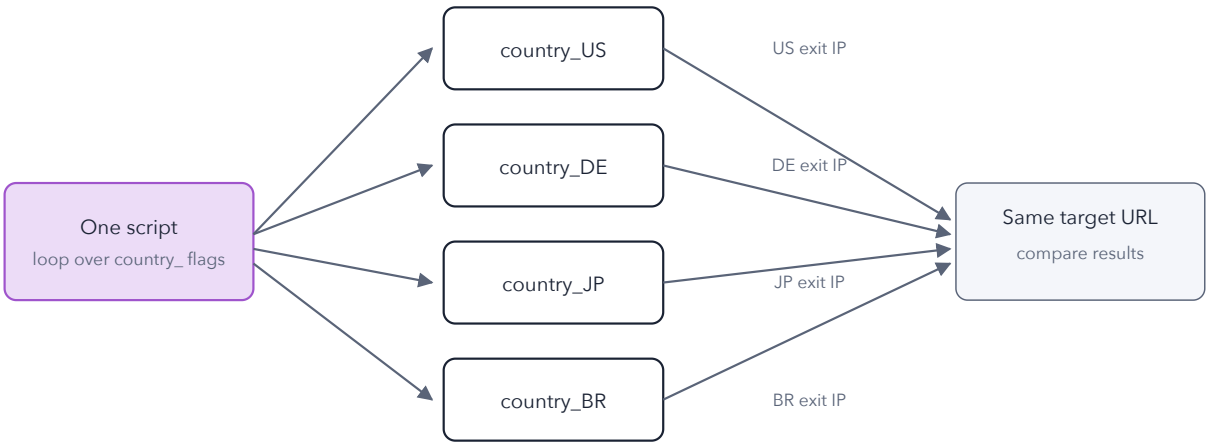


Figure 7.2, Multi-geo fan-out: one script iterates over N `country_` flags, producing N exit IPs that all hit the same target URL so the localized results can be compared.

📝 Recipe 7.2, Multi-country SEO/SERP & ad-verification checks

Prerequisites: active plan, Python 3 + `requests`, and a target URL you're authorized to query (respect its rate limits and terms).

1. Define the list of countries you want to verify, as ISO codes (`US`, `DE`, `JP`, `BR`, …).
2. For each country, build a proxies dict with that `country_` flag on the smart username.
3. Fetch the **same URL** through each country's proxy, one at a time.
4. Store each response (or the fields you care about, rankings, prices, ad slots) keyed by country.
5. Compare the results across countries to spot geo-differences.

```
import requests

URL = "https://example.com/search?q=running+shoes"
countries = ["US", "DE", "JP", "BR"]
results = {}

for cc in countries:
    proxy = f"http://user_XXXX,country_{cc}:PASSWORD@portal.anyip.io:1080"
    proxies = {"http": proxy, "https": proxy}
    r = requests.get(URL, proxies=proxies, timeout=10)
    results[cc] = r.text

for cc in countries:
    print(cc, "->", len(results[cc]), "bytes")   # then diff the parsed fields
```

✅ Expected result: a response per country, and the parsed rankings/prices/ads **differ per country**, proving you're seeing each market's localized page, not your own.

Recipe C, Sticky mobile QA across a multi-step flow

Some QA flows need the exit IP to stay *constant* for the whole run, a login, then a cart, then a checkout, all from the same session. Here you add `type_mobile` for a real 4G/5G footprint and a `session_` flag to hold one IP.

📝 Recipe 7.3, Sticky mobile-network app/API QA run

Prerequisites: active plan, Python 3 + `requests`, and a QA target that is **your own app/API** or one you're authorized to test.

1. Choose a session name (1 to 32 chars, letters/digits only) for this QA run, e.g. `qa_run1`.
2. Build a proxies dict with `type_mobile` and `session_qa_run1` so the IP sticks across the flow.
3. Confirm the exit IP once at the start by hitting `http://ip-api.com/json` and recording the address.
4. Run each step of the QA flow (e.g. auth → fetch profile → submit action) through the **same** proxies dict.
5. Re-check the exit IP at the end and assert it matches the start.

```
import requests

PROXY = "http://user_XXXX,type_mobile,session_qa_run1:PASSWORD@portal.anyip.io:1080"
proxies = {"http": PROXY, "https": PROXY}

def ip():
    return requests.get("http://ip-api.com/json", proxies=proxies, timeout=10).json()["query"]

start_ip = ip()
requests.get("https://your-app.example.com/api/login",   proxies=proxies, timeout=10)
requests.get("https://your-app.example.com/api/profile", proxies=proxies, timeout=10)
requests.get("https://your-app.example.com/api/action",  proxies=proxies, timeout=10)
end_ip = ip()

assert start_ip == end_ip, f"IP changed mid-flow: {start_ip} != {end_ip}"
print("stable mobile IP across flow:", start_ip)
```

✅ Expected result: the **same exit IP** reported at the start and end of the flow, so every QA step is exercised from one stable mobile identity.

📋 Scraper reliability checklist

- [ ] I confirmed I'm authorized to collect this data and checked `robots.txt` and rate limits (see Chapter 9).
- [ ] Rotating (no `session_`) for bulk scraping; sticky `session_` only when a flow needs one IP.
- [ ] Every fetch uses a **bounded** retry with exponential backoff, no `while True`.
- [ ] I back off on HTTP 429 instead of retrying immediately.
- [ ] There's a `time.sleep()` between requests to respect the target's limits.
- [ ] Image and video loading are disabled to cut data use by up to 80%.
- [ ] Every request has a `timeout` so a hung peer can't stall the job.
- [ ] I spot-check the exit IP to confirm rotation (or stickiness) is behaving.

💼 Business angle

Reliable collection is a cost story: with a <0.2% ban rate on major platforms you re-run fewer jobs, and disabling images/video plus proper backoff can cut wasted GB by up to 80%, so more of every plan's traffic turns into usable rows instead of retries and error pages.

📌 Key takeaways

- The proxy is one string in a `proxies` dict, `type_`, `country_`, and `session_` flags decide the network, geography, and stickiness.
- Rotating (no session) for bulk scraping; multi-country fan-out by looping `country_` flags; sticky `type_mobile,session_` for a stable QA identity.
- Always retry with bounded exponential backoff and back off on 429, a `while True` loop can burn 10 GB on error pages.
- Disable images/video to save up to 80% of data, and only ever collect what you're authorized to (Chapter 9).

🔗 Go deeper

- anyIP docs, Integration Guides & Network Types, source for the `proxies` dict form, `type_residential` / `type_mobile`, and rotating vs sticky behavior.
- ip-api.com/json, the IP-echo endpoint used to verify rotation and session stickiness.
- Python `requests`, proxies, how the HTTP/HTTPS proxy dict is passed per request.

8. When Things Break, Errors, UDP & Data Hygiene

⚡ The gist

- Every anyIP failure maps to an HTTP status, 407 auth, 400 bad flags, 403 forbidden host, 409 collision, 413 IP cap, 502/503/504 peer trouble, and each has a known first fix.
- The exact error code is in the `X-Anyip-Error` response header. Read it before you change anything.
- Five user mistakes cause most tickets: wrong port for the auth method, an unsaved IP whitelist, commas your software rejects, VPN + proxy double-hops, and expired plans.
- "Ghost usage", browser updates, telemetry, images, and runaway retry loops, can eat your GB. Disabling images and video autoplay saves up to 80%.
- UDP / HTTP-3 (QUIC) is in beta: SOCKS5 only, same port 1080. Test at cloudflare-quic.com.

Reading time: ~10 min

The error-code map

anyIP fails loudly and precisely. When something goes wrong, the gateway returns a standard HTTP status plus a machine-readable code in the `X-Anyip-Error` header, for example: `x-anyip-error: 407 Proxy Authentication Required (code: region_country_is_missing)`. That header is your diagnosis; the status number tells you which family of problem you have.

Run `curl -v` (or check response headers in your HTTP library) and match what you see against the map below. Six families cover everything.

407, Authentication	400, Bad flags	403, Forbidden host
wrong_customer_credentials bad_basic_auth_format ip_whitelisted_bad_port region_country_is_missing **Fix: match port to auth method**	bad_asn session_too_long (>10080) bad_type **Fix: check flag spelling & limits**	host_not_allowed (banks, government sites blocked by design) **Fix: target is off-limits, always**
409, Collision exhausted	**413, IP cap**	**502 / 503 / 504, Peers**
session_asn_ip_exhausted_for_user collision_strict_max_ip_used_for_user (sessasn / sessipcollision strict) **Fix: relax the strict flags**	max_ips_reached_for_user (internal 2,000-IP cap) **Fix: reuse sessions, don't hoard IPs**	no_peers_available peer_busy peer_timeout **Fix: relax targeting / rotate**

Figure 8.1, Error-code map: six status families, each with its first fix. The exact code arrives in the `X-Anyip-Error` header.

Error quick reference

Status	Code (in `X-Anyip-Error`)	What it means	First fix
407	`wrong_customer_credentials`	Username or password wrong	Re-copy both from the dashboard
407	`bad_basic_auth_format`	Auth string malformed	Use `user_ID:password` exactly
407	`ip_whitelisted_bad_port`	Whitelist auth on a user:pass port (or vice versa)	Whitelist → your 2000 to 2999 port; user:pass → 1080/443
407	`region_country_is_missing`	`region_` flag without its parent `country_`	Add the parent flag first
400	`bad_asn`	ASN number invalid	Check the ASN against a known list
400	`session_too_long`	`sesstime` above 10,080 minutes	Stay within 1 to 10,080 (max 7 days)
400	`bad_type`	Unknown network type flag	Only `type_residential` or `type_mobile`
403	`host_not_allowed`	Target is a forbidden host (banks, government)	None, these blocks are permanent by design
409	`session_asn_ip_exhausted_for_user`	No more unique IPs on that ASN (`sessasn_strict`)	Drop or relax the strict flag
409	`collision_strict_max_ip_used_for_user`	Unique-IP guarantee exhausted (`sessipcollision_strict`)	Fewer parallel strict sessions
413	`max_ips_reached_for_user`	Hit the internal 2,000-unique-IP cap	Reuse sticky sessions instead of minting new IPs
502	`no_peers_available`	No peer matches your targeting	Relax targeting (drop city/ASN, widen to country)
503	`peer_busy`	Peer can't take the request right now	Retry with backoff / rotate to a new IP
504	`peer_timeout`	Peer didn't answer in time	Rotate and retry with backoff

The top-5 user mistakes

Before you suspect the network, rule out these five. They account for most "it doesn't connect" moments:

1. **Port / auth mismatch.** Username-and-password auth lives on ports **1080** and **443**; IP-whitelist auth lives on your assigned port in **2000 to 2999**. Cross them and you get a 407.
2. **Forgot to save the IP whitelist.** Adding your IP in the dashboard does nothing until you hit Save, and it breaks again silently when your home IP changes.
3. **Commas your software rejects.** Some tools treat commas in usernames as field separators. anyIP accepts pipes as a drop-in: `user_123|country_US`.
4. **VPN + proxy double-hop.** Running a VPN and the proxy simultaneously produces timeouts. Turn the VPN off.
5. **Expired plan or 0 GB.** You need *both* active subscription time *and* remaining GB to connect. Either one at zero means no connection.

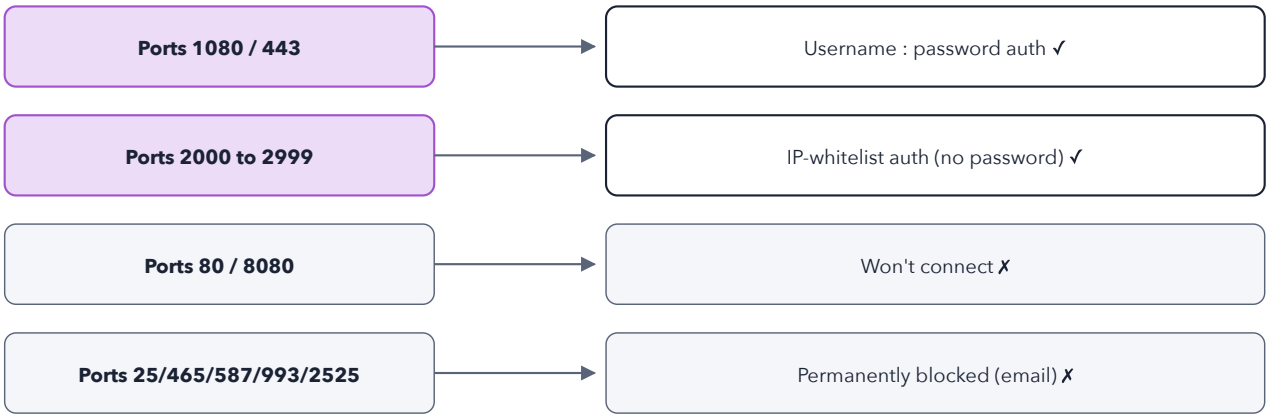


Figure 8.2, Ports & auth matrix: which port pairs with which auth method, and which ports never work.

📝 Recipe 8.1, Diagnose a 407 in 3 checks

Prerequisites: your proxy credentials (or whitelist port) at hand; `curl` installed; a plan with active time and remaining GB.

1. **Check the port matches the auth method.** User:pass belongs on 1080 or 443; IP whitelist belongs on your assigned 2000 to 2999 port:

   ```
   # user:pass auth, ports 1080 or 443
   curl -v -x "http://user_XXXX:PASSWORD@portal.anyip.io:1080" http://ip-api.com/json

   # IP-whitelist auth, your assigned port in 2000-2999, no credentials
   curl -v -x "http://portal.anyip.io:2000" http://ip-api.com/json
   ```

   A mismatch shows up as `ip_whitelisted_bad_port` in the `X-Anyip-Error` header.

2. **Check the whitelist was saved and is current.** In the dashboard: Proxies → Get Proxy Details → IP Whitelisting tab, confirm your *current* public IP is listed, a port is assigned, and Save was actually clicked. Dynamic home IPs silently invalidate old entries.

3. **Check the full password was copied.** Re-copy the entire password from the dashboard (no trailing spaces, no truncation) and re-run the curl from step 1, watching the `X-Anyip-Error` header for `wrong_customer_credentials` or `bad_basic_auth_format`.

✅ Expected result: the request returns HTTP 200 and ip-api.com reports a proxy IP different from your own.

"Ghost usage": where your GB really goes

Plans are billed by traffic, so every byte through the proxy counts, including bytes you never asked for. The usual culprits, in descending order of pain:

- **Browser background processes**: auto-updates (~200 MB per update), sync and telemetry chatter, all routed through your proxy if the whole browser is proxied.
- **Silent local AI model downloads**: modern browsers can pull 4 to 5 GB of model files without asking.
- **Images and video**: 5 to 10 MB per page adds up fast. Disabling image loading and video autoplay saves up to **80%** of traffic.
- **Runaway retry loops**: a `while True` retry with no delay burns GB on error pages (see the warning below).

- [ ] Disable image loading in any proxied browser or scraper profile.
- [ ] Disable video autoplay.
- [ ] Turn off (or route around the proxy) browser auto-update, sync, and telemetry.
- [ ] Watch for silent multi-GB local AI model downloads in proxied browsers.
- [ ] Replace every bare `while True` retry with bounded retries + exponential backoff.
- [ ] If usage still looks wrong, rotate your proxy password, shared or compromised credentials mean someone else is spending your GB.

💼 Business angle

Ghost usage is pure waste on a per-GB bill: a single unnoticed browser update can cost as much as thousands of useful page fetches, and one silent AI model download can wipe out a Micro plan's entire monthly allowance. Cutting images and autoplay alone reclaims up to 80% of spend, the cheapest optimization in this book.

⚠️ Watch out

The docs' cautionary tale, verbatim: "A common coding error is a retry loop without a delay... Your script fires 10,000 requests per second for an hour, burning 10GB of data on error pages alone." Failed requests that reach anyIP's server still count against your GB, always back off on errors instead of hammering.

UDP & HTTP-3 (QUIC), beta

anyIP supports UDP tunneling in beta, which unlocks HTTP/3 (QUIC) through the proxy. Two constraints define it: UDP works **only via SOCKS5**, on the same port **1080** with standard user:pass auth, and your client must support SOCKS5 UDP Associate. Many common HTTP tools, including Python `requests` and standard `curl`, do not proxy UDP at all, so a browser configured for SOCKS5 is the easiest test rig.

Why bother? A modern HTTP/3 footprint blends with real residential QUIC traffic, a cleaner, more current network signature for your authorized testing and data-collection work.

📝 Recipe 8.2, Verify UDP / HTTP-3 tunneling

Prerequisites: a client that supports SOCKS5 UDP Associate (a browser with SOCKS5 proxy settings works; Python `requests` and standard `curl` won't); your user:pass credentials.

1. Configure your client as a **SOCKS5** proxy, host `portal.anyip.io`, port `1080`, with your username and password:

```
Protocol: SOCKS5
Host:     portal.anyip.io
Port:     1080
Username: user_XXXX
Password: YOUR_PASSWORD
```

2. Confirm the client actually supports UDP over SOCKS5 (UDP Associate), without it, traffic silently falls back to TCP.
3. Browse to https://cloudflare-quic.com/ through the proxy.
4. Read the verdict on the page.

✅ Expected result: the page reports "your browser used HTTP/3", UDP is tunneling through the proxy. If it reports an older protocol, recheck step 2.

📌 Key takeaways

- Diagnose from the status family: 407 auth, 400 bad flags, 403 forbidden host, 409 strict-flag exhaustion, 413 the 2,000-IP cap, 502/503/504 peer trouble (relax targeting, rotate, back off).
- The precise code is always in the `X-Anyip-Error` header, read it with `curl -v` before changing anything.
- Ports are the #1 tripwire: 1080/443 for user:pass, 2000 to 2999 for IP whitelist, 80/8080 never, email ports permanently blocked.
- Guard your GB: kill images, autoplay, background updates, and delay-free retry loops, up to 80% savings; rotate credentials if usage looks haunted.
- UDP/HTTP-3 is SOCKS5-only on port 1080 (beta); prove it works at cloudflare-quic.com.

🔗 Go deeper

- anyIP documentation, Proxy Client Errors, Troubleshooting, and UDP Support pages, the source for every code and limit in this chapter.
- anyIP Billing & Data docs, how traffic is counted (uploads, headers, and failed requests included) and the "ghost usage" culprits.
- cloudflare-quic.com, Cloudflare's HTTP/3 test page used in Recipe 8.2.
- ip-api.com, the IP-echo endpoint used to verify a working connection in Recipe 8.1.

9. Play It Straight, Ethical & Compliant Use

> ⚡ **The gist**
>
> - The five golden rules that keep a proxy program legal, defensible, and boring, in the good way.
> - What anyIP itself refuses to route (banks, government, email) and why that's a feature, not a limitation.
> - GDPR/CCPA in plain English: "publicly visible" is not the same as "free to collect."
> - A green-light/red-light decision flow and a pre-job checklist to run before every new job.
>
> *Reading time: ~9 min*

Everything in the previous chapters makes you harder to block. This chapter is about the other half of the job: making sure the traffic you send *deserves* to get through. That's not a lecture, it's operational risk management. A scraping program that ignores authorization, robots.txt, or privacy law doesn't fail loudly on day one; it fails expensively on day 400, in the form of a legal letter, a terminated account, or a headline. The rules below are what a careful team does by default.

The golden rules

Five rules cover almost every situation. If a job satisfies all five, it's very likely fine. If it fails any one, stop and fix that before you write a line of code.

1. **Only access data and systems you're authorized to access.** "Authorized" means the site serves the content publicly, or you have an account whose terms permit your access, or you have explicit permission. Bypassing a login wall, a paywall, or an access control you weren't given is where "scraping" turns into unauthorized access, the territory of CFAA-style claims in the US and their equivalents elsewhere.
2. **Respect robots.txt and each site's Terms of Service.** The Robots Exclusion Protocol (RFC 9309) is the standard, machine-readable way a site tells crawlers what's off-limits. Fetch `/robots.txt` before you crawl, honor `Disallow` rules for your user agent, and read the ToS of any site you hit at scale. It costs one HTTP request and ten minutes.
3. **Rate-limit, and identify yourself where appropriate.** Spread requests out, add backoff on errors (see Chapter 7), and for cooperative crawling set an honest `User-Agent` with a contact address. A polite crawler at 1 req/s is rarely anyone's problem; a firehose is everyone's problem, including yours, since it burns your GB.
4. **Collect the minimum data the job needs.** If the task is price monitoring, collect prices, not the reviewer names, avatars, and emails that happen to be on the page. Every field you don't store is a field you never have to secure, justify, or delete.
5. **Never collect personal data unlawfully.** Names, emails, phone numbers, photos, user IDs, and precise locations are personal data. Collecting them triggers privacy-law obligations (next section) even when the page is public.

A responsible network by design: what anyIP blocks

anyIP enforces some of these lines at the network level. Certain destinations simply will not resolve through the proxy, by design, in the provider's words, "to prevent fraud and preserve the reputation of our network." The docs state these restrictions are strict and cannot be removed:

Blocked category	Examples	Why
Financial institutions	PayPal, Stripe, Bank of America, Chase, HSBC and similar	Residential-IP access to banking and payment systems is a classic fraud pattern; blocking it protects both the peers and the pool's reputation.
Government & public-sector sites	Tax portals, immigration/visa systems, social security & benefits, government ID/auth services	Masking identity to public authorities has almost no legitimate proxy use case and enormous abuse potential.
Email ports	25, 465, 587, 993, 2525 permanently blocked	The network "is not designed for mailing", open email ports on residential IPs are a spam engine waiting to happen.

Read this the right way: it's a feature. Every blocked fraud attempt keeps the IP pool's reputation clean, which is exactly what produces the low ban rates you're paying for. It also acts as a guardrail for your own team, a misconfigured job that wanders toward a bank returns a `403 host_not_allowed` instead of an incident report. A provider that filters abuse at the gateway is a provider whose IPs stay trusted; you and the network are on the same side of this trade.

The pre-flight question: should I run this job?

Before any new scraping or automation job, walk the flow below. It takes thirty seconds and it's the cheapest compliance review you'll ever run.

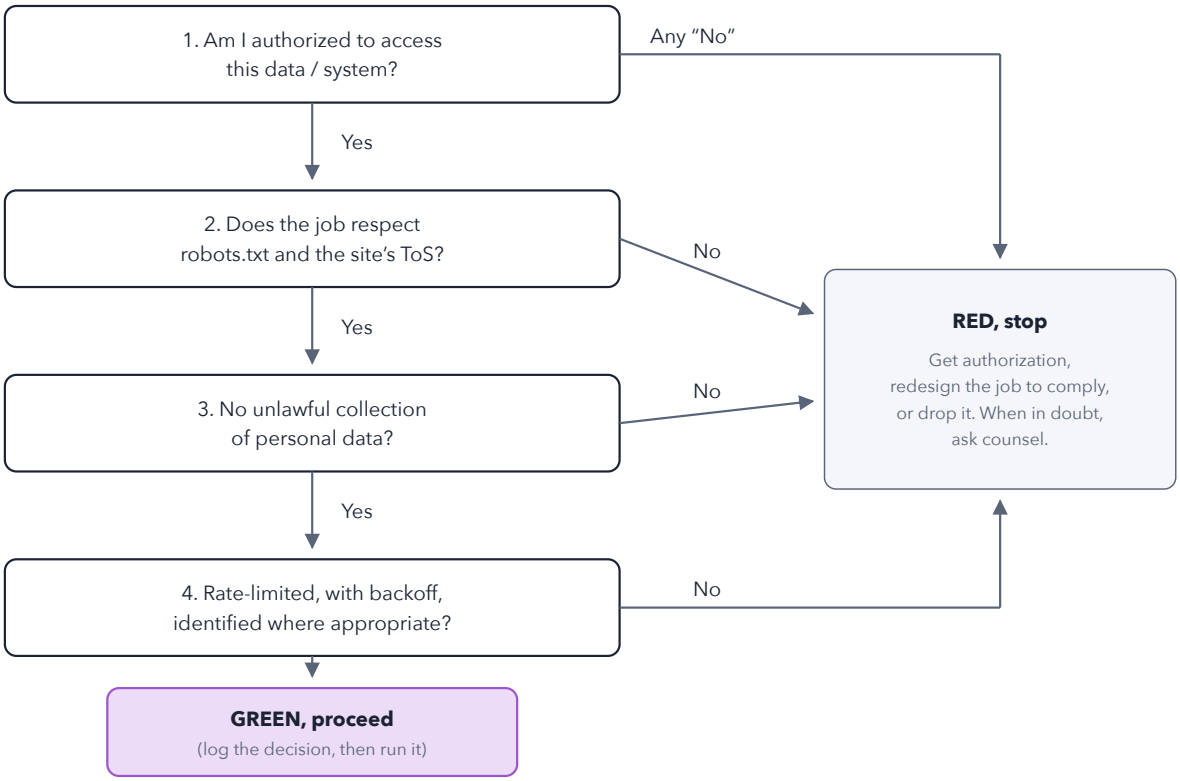


Figure 9.1, "Should I run this job?" All four gates must be green; any single "No" means stop and fix it before running.

GDPR & CCPA in plain English

You don't need to be a lawyer to stay out of the obvious traps. The two regimes you'll hear about most are the EU's GDPR and California's CCPA/CPRA, and their practical rules for scrapers boil down to this:

- **Personal data needs a lawful basis.** Under GDPR you must be able to name *why* you're allowed to process someone's data (consent, contract, legitimate interest…). "It was on a public webpage" is not on the list by itself.
- **Public ≠ exempt.** Scraping publicly visible profiles, reviews, or listings still triggers privacy obligations the moment the data identifies a person. Regulators have fined companies specifically for large-scale scraping of public profiles.
- **Honor deletion and opt-out requests.** If someone asks that their data be removed, you need a way to actually find and delete it, which is much easier if you minimized what you collected in the first place.
- **Don't build shadow profiles.** Merging scraped fragments from multiple sites into a dossier on an individual who never interacted with you is exactly the pattern privacy law is aimed at. Avoid it entirely.
- **Know where your data subjects are.** GDPR applies based on whose data it is, not where your servers are. If the pages you scrape contain EU residents' personal data, GDPR is in scope even for a US company.

These are general principles, not legal advice, thresholds, exemptions, and enforcement vary by jurisdiction and change over time. For anything beyond clearly non-personal data (prices, specs, availability, your own accounts' data), get an hour with counsel before the job ships, not after the complaint arrives.

What proxies are not for

For the avoidance of doubt, none of the techniques in this book are for:

- Fraud of any kind, payment fraud, chargeback abuse, or misrepresenting who you are to obtain money or goods.
- Account takeover, credential stuffing, or accessing accounts that aren't yours.
- Creating fake or bulk accounts to manipulate platforms, reviews, votes, or ad systems.
- Evading law enforcement, sanctions, or security controls.
- Circumventing a ban where doing so breaks the law or a binding agreement you accepted. A platform ban is usually a contract decision; routing around it with fresh IPs doesn't undo the contract.

The legitimate uses, price and market intelligence, SEO and ad verification, geo-accurate QA, research on public data, are more than enough to justify the tool. That's what the recipes in this book are built around, and it's why anyIP hard-blocks the destinations most associated with the list above.

> 💼 **Business angle**
>
> Getting this wrong is rarely a small mistake. Unauthorized-access claims (CFAA-style in the US) and GDPR penalties, which can reach a percentage of global revenue for serious violations, turn a data pipeline into a board-level problem, and the reputational cost of a scraping scandal outlasts any fine. Getting it right is cheap: the checklist below takes minutes per job, and a documented, compliant pipeline is an asset you can show customers, partners, and auditors.

Pre-job compliance checklist

Run this before every *new* job, and re-run it when a job's scope changes (new target site, new fields, new geography):

- [ ] I'm authorized to access this target, it's public content, my own account within its terms, or I have written permission.
- [ ] I've fetched and read the target's `robots.txt` (RFC 9309) and my crawler honors its Disallow rules.
- [ ] I've read the target's Terms of Service and my use doesn't breach them.
- [ ] The fields I collect are the minimum for the job, I've listed them and struck out anything I don't need.
- [ ] No personal data is collected, or if it is, I've named the lawful basis and counsel has signed off.
- [ ] Requests are rate-limited with backoff on errors (no bare retry loops), and I identify my crawler where appropriate.
- [ ] I know which jurisdictions' laws apply (mine, the target's, the data subjects') for this target and data type.
- [ ] There's a retention limit and a deletion path for whatever I store.
- [ ] I've logged this decision (target, purpose, fields, date) so future-me can prove the job was reviewed.

> ⚠️ **Watch out**
>
> The most common compliance mistake isn't malice, it's scope creep. A job approved to collect prices quietly starts storing reviewer names and photos because "they were in the JSON anyway." Public visibility does not make personal data free to keep: the moment identifiable people enter your dataset, GDPR/CCPA obligations attach. Re-run the checklist whenever the fields you store change, not just when the job is new.

> 📌 **Key takeaways**
>
> - Five golden rules: authorized access only, respect robots.txt (RFC 9309) and ToS, rate-limit and identify yourself, minimize data, never collect personal data unlawfully.
> - anyIP hard-blocks financial institutions, government/public-sector sites, and email ports (25/465/587/993/2525), a design choice that keeps the pool clean and you on the right side of the line.
> - Public data can still be personal data: lawful basis, deletion paths, and no shadow profiles apply even to scraped content.
> - Proxies are not for fraud, account takeover, fake accounts, or evading law-enforcement or binding agreements.
> - Walk Figure 9.1 and the pre-job checklist before every new job; any single "No" is a stop sign, not a speed bump.

🔗 **Go deeper**

- RFC 9309, Robots Exclusion Protocol, the official robots.txt standard your crawler should honor.
- anyIP Troubleshooting guide, the source for the blocked financial/government destinations and the reasoning behind them.
- anyIP Proxy Client Errors, includes `403 host_not_allowed`, the error you'll see on a blocked destination.
- GDPR full text (EUR-Lex, Regulation 2016/679), the primary source for EU data-protection obligations.
- CCPA, California Attorney General overview, plain-language summary of California consumer privacy rights.

10. The Business Case, Pricing, ROI & Alternatives

⚡ The gist

- One combined residential + mobile plan, billed by GB: four tiers from $25/5 GB (Micro) to $1000/500 GB (Enterprise), effective rate falls from $5/GB to $2/GB as you scale.
- A worked cost model: with images disabled, the Basic plan buys roughly 400,000 pages, about $0.20 per 1,000 pages (assumptions inside).
- Top-up adds GB but does **not** extend your expiry date; Renew adds +30 days. You need **both** active time and remaining GB to connect, and unused GB only rolls over if you renew within 7 days of expiry.
- There is no free trial: the Micro plan paid by card, covered by the 14-day money-back guarantee on unused data, is the de-facto trial.

Reading time: ~10 min

How anyIP prices: one plan, billed by the gigabyte

Most proxy vendors sell residential and mobile access as separate products, with mobile priced at a premium. anyIP sells one combined plan: every tier includes unlimited residential *and* mobile IPs, HTTP/HTTPS and SOCKS5, unlimited rotation and sessions, country/ISP/region/GPS targeting, and both authentication methods. The only variables between tiers are the monthly GB allowance, the overage rate, and the number of team members (credential sets).

That makes the buying decision one-dimensional: estimate your monthly traffic in GB, pick the tier whose allowance covers it, and note that the effective per-GB rate halves as you scale.

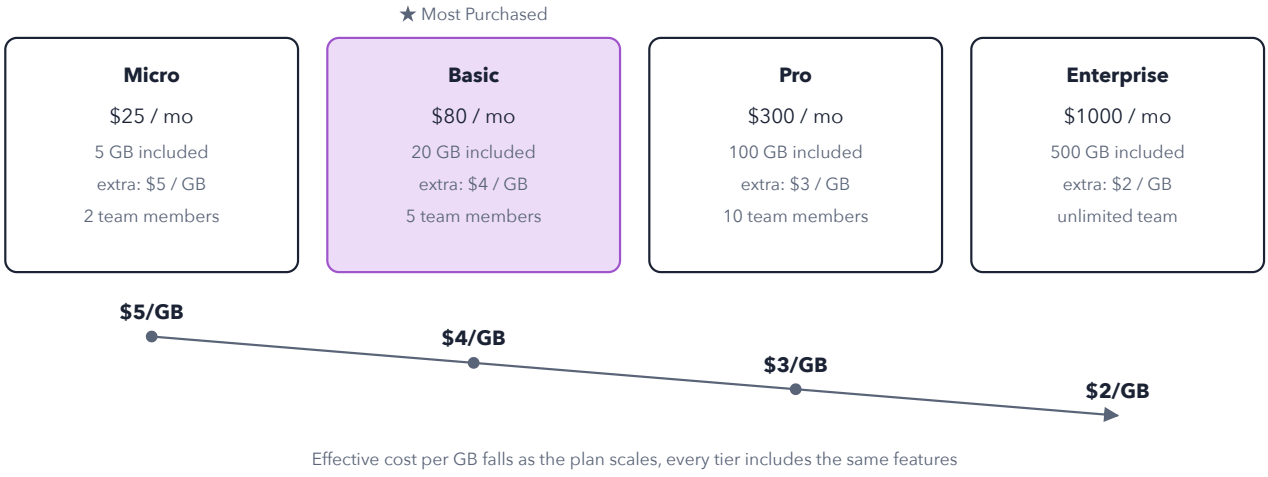


Figure 10.1, The four pricing tiers. Same feature set everywhere; only the GB allowance, overage rate, and team size change.

Plan	Price / mo	Included GB	Extra GB	Team members	Effective rate
Micro	$25	5 GB	$5/GB	2	$5.00/GB
Basic (Most Purchased)	$80	20 GB	$4/GB	5	$4.00/GB
Pro	$300	100 GB	$3/GB	10	$3.00/GB
Enterprise	$1000	500 GB	$2/GB	Unlimited	$2.00/GB

All tiers: unlimited residential + mobile IPs, HTTP/HTTPS + SOCKS5, IP rotation, unlimited sessions, country/ISP/region targeting and GPS coordinates, user:pass + IP-whitelist auth. Payments: credit card, crypto via NowPayments, or wire transfer.

The cost model: from GB to cost-per-1,000 rows

Because billing is pure traffic, ROI math is a division problem. Data usage counts download *and* upload, HTTP headers, and failed requests (anything that reached anyIP's server), so the lever that matters most is payload size per page.

Worked example (assumptions marked, recalculate with your own measurements):

- *Assumption:* average page ≈ **50 KB** with image loading and video autoplay disabled (Chapter 8's data-saving setup; disabling media can cut usage by up to 80%).
- 1 GB ≈ 1,000,000 KB ÷ 50 KB/page ≈ **~20,000 pages per GB**.
- Basic plan: $80 / 20 GB → 20 GB × 20,000 ≈ **~400,000 pages per month**.
- $80 ÷ 400 (thousand pages) ≈ **~$0.20 per 1,000 pages**. If each page yields one row, that is ~$0.20 per 1,000 rows.

On Enterprise the same math lands at ~$0.10 per 1,000 pages ($2/GB × 50 KB/page, same assumption). Heavier pages move the number linearly: at 500 KB/page (images on), Basic delivers only ~40,000 pages, i.e. ~$2.00 per 1,000 pages. Measure a real sample of your target with `curl -w '%{size_download}'` before you commit to a tier.

> 💼 **Business angle**
>
> The spend-justification pitch writes itself: one blocked-and-retried scrape burns both engineer hours and raw GB on error pages. At ~$0.20 per 1,000 successful pages on Basic, the proxy line item is usually dwarfed by the cost of the missing data it prevents, price the decision as cost-per-*successful*-request, not cost-per-GB.

Top-up vs Renew, don't lose money on the calendar

Your subscription has two independent resources: **time** (an active 30-day window) and **data** (remaining GB). You need **both** to connect. Two different buttons manage them, and mixing them up is the most common billing mistake:

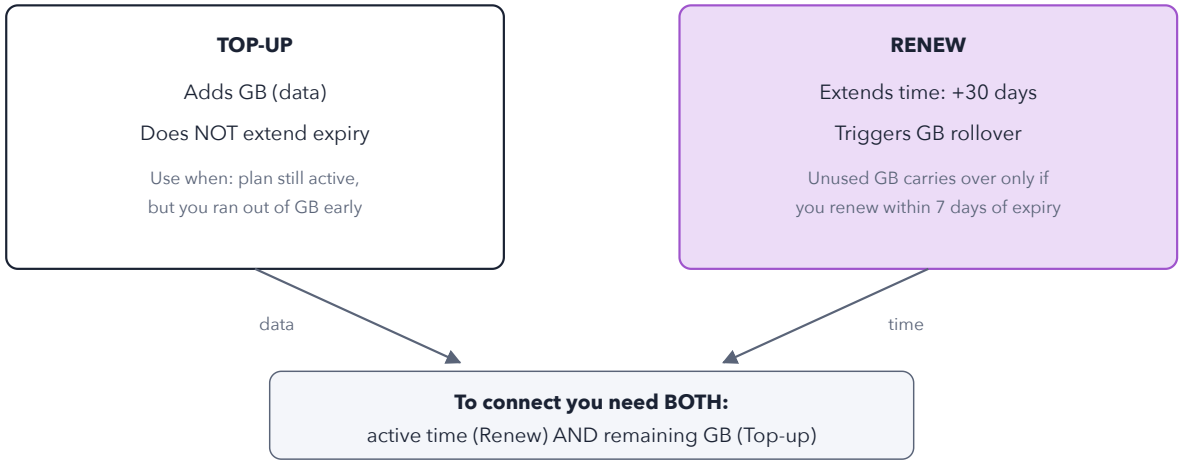


Figure 10.2, Top-up vs Renew. Two buttons, two different resources; the connection only works while you hold both.

> ⚠️ **Watch out**
>
> **The rollover/expiry trap.** Topping up GB the day before expiry does *not* buy you time, the plan still expires and your connections stop. And unused GB rolls over to the next cycle **only if you renew within 7 days of expiry** (renewing on the same plan, an upgrade, or a downgrade all keep the rollover). Renew on day 8 and the leftover data is gone. Put the renewal date on the team calendar with a 7-day-before reminder.

Trial, refunds, and payment fine print

There is no free trial, promo code, or loyalty discount. The intended path is the **Micro plan ($25/5 GB) paid by credit card**: the 14-day money-back guarantee covers **unused data on card payments only**, refunds are full (not prorated) and processed in 2 to 3 business days. Crypto payments (NowPayments, USDT/USDC) are definitive and **not refundable**, so never pilot with crypto. Wire transfer is available for teams that need it.

Who should adopt it, and instead of what?

The comparison below positions anyIP against the two obvious alternatives: cheap datacenter proxies and premium residential providers. Competitor cells are qualitative, pull current quotes before a procurement decision.

Axis	anyIP	Datacenter proxies	Premium residential competitors
IP type / trust	Real peer devices, residential and mobile, high trust	Server-farm IPs, easily fingerprinted, low trust	Residential, high trust; mobile usually a separate premium product
Detection risk	Ban rate < 0.2% on major platforms (docs spec)	High on protected/consumer sites; fine for unprotected targets	Low, comparable to anyIP
Price per GB	$5 → $2/GB by tier (Micro → Enterprise)	Typically the cheapest option, often not billed per GB	Typically higher per GB; anyIP positions itself as undercutting them
Residential + mobile in one plan	Yes, every tier, same price	No (neither type)	Often no, mobile priced separately at a premium
Refund	14-day money-back on unused data (card payments only)	Varies by vendor	Varies by vendor

📋 Plan-selection checklist

- [ ] I measured average payload per request on a real sample of my target (with images/video disabled) and converted my monthly volume into GB.
- [ ] My monthly GB fits inside a tier's allowance with ~20% headroom, overage rates ($5/$4/$3/$2 per GB) are the same as the tier's effective rate, so chronic overage means I should move up a tier.
- [ ] I counted credential sets needed (people, environments, tools): 2 (Micro), 5 (Basic), 10 (Pro), or unlimited (Enterprise).
- [ ] My targets are bot-protected consumer sites or need geo/carrier accuracy, otherwise cheap datacenter proxies may be enough.
- [ ] I'm piloting on Micro, paid by card (not crypto), so the 14-day unused-data refund acts as my trial.
- [ ] The renewal date is on the team calendar with a reminder inside the 7-day rollover window.
- [ ] My use case is authorized data collection, QA, or verification, and it isn't on anyIP's blocked categories (financial institutions, government portals, email ports).

What adopters report

Three case-study-style snapshots from published reviews (G2 / Trustpilot, 5.0 ratings):

- **Switching from a large competitor (Alex U., G2):** the team cut proxy spend after leaving a large competitor while keeping success rates steady, and a configuration issue was fixed by support in about 30 minutes. The classic migration profile: same reliability, lower per-GB bill.
- **Social-media operations (Md. Zahid Hasan S.):** "Reliable proxies for social media work", the sticky-session and per-session-unique-IP features from Chapter 6 doing their job for account-management workflows (run within platform rules, see Chapter 9).
- **The minimalist (Chinthaka J.):** "Simple standalone proxy that works", one connection string, no infrastructure to babysit, which is precisely the smart-username pitch from Chapter 3.

📌 Key takeaways

- Pricing is one-dimensional: pick the tier whose GB allowance covers your monthly traffic, $25/5 GB, $80/20 GB, $300/100 GB, $1000/500 GB, and the per-GB rate falls from $5 to $2 as you scale.
- With lean payloads (~50 KB/page, our stated assumption), Basic buys ~400,000 pages a month: roughly $0.20 per 1,000 pages.
- Top-up = more GB, same expiry. Renew = +30 days. You need both time and GB to connect, and unused GB survives only if you renew within 7 days of expiry.
- No free trial exists: Micro + card payment + the 14-day unused-data refund is the pilot path; crypto is non-refundable.
- Choose anyIP over datacenter proxies when targets are protected or geo-sensitive, and over premium residential vendors when you want mobile included in one plan at a lower per-GB rate.

🔗 **Go deeper**

- anyIP pricing page, the four tiers, payment methods, and refund terms summarized in this chapter.
- anyIP documentation, see the Billing & Data section for top-up vs renew, the 7-day rollover rule, and what counts toward data usage.
- anyIP homepage, testimonials (G2, Trustpilot) and positioning claims quoted above.

11. Cheat Sheet & Learning Path

⚡ The gist

- Every flag, port, and error code from this book on a handful of tables, bookmark this chapter.
- One connection string to remember:
  `http://user_XXXX,flags:PASSWORD@portal.anyip.io:1080` .
- A 30/60/90-day path from first proxied request to a monitored, multi-geo production pipeline.
- A production-readiness checklist to run before any job goes live.

Reading time: ~6 min (then keep it open forever)

The one-page flag cheat sheet

Everything anyIP does is controlled by flags appended to your username. Flags are separated by a comma `,`, or a pipe `|` if your software rejects commas (e.g. `user_123|country_US`). This is the full set from the Configuration Reference.

Flag	Meaning	Example
`type_mobile`	Force a 4G/5G mobile IP	`user_XXXX,type_mobile`
`type_residential`	Force a Wi-Fi/broadband residential IP	`user_XXXX,type_residential`
`country_[ISO]`	Pin to a country (ISO code)	`user_XXXX,country_FR`
`region_[name]`	Pin to a region (country must be set first)	`country_US,region_texas`
`city_[name]`	Pin to a city (country + region must be set first)	`country_US,region_texas,city_dallas`
`asn_[number]`	Pin to an ISP/carrier by ASN	`country_US,asn_21928` (T-Mobile US)
`pool_[name]`	Target a curated multi-country pool (19 pools)	`user_XXXX,pool_europe`
`session_[name]`	Sticky IP, keep the same IP across requests (name: 1 to 32 chars, a to z A to Z 0 to 9)	`user_XXXX,session_job1`
`sesstime_[min]`	Sticky duration in minutes (1 to 10080; default up to 7 days)	`session_job1,sesstime_10`
`sessreplace_false`	Fail (`peer_not_found`) instead of rotating if the sticky IP drops	`session_job1,sessreplace_false`
`sessasn_strict`	Replacement IP must stay on the same ASN/ISP	`session_job1,sessasn_strict`
`sessipcollision_strict`	Guarantee no two sessions share the same IP	`session_a,sessipcollision_strict`
`lat_[x],lon_[y]`	GPS geofencing, closest available peer (not guaranteed)	`user_XXXX,lat_48.8584,lon_2.2945`

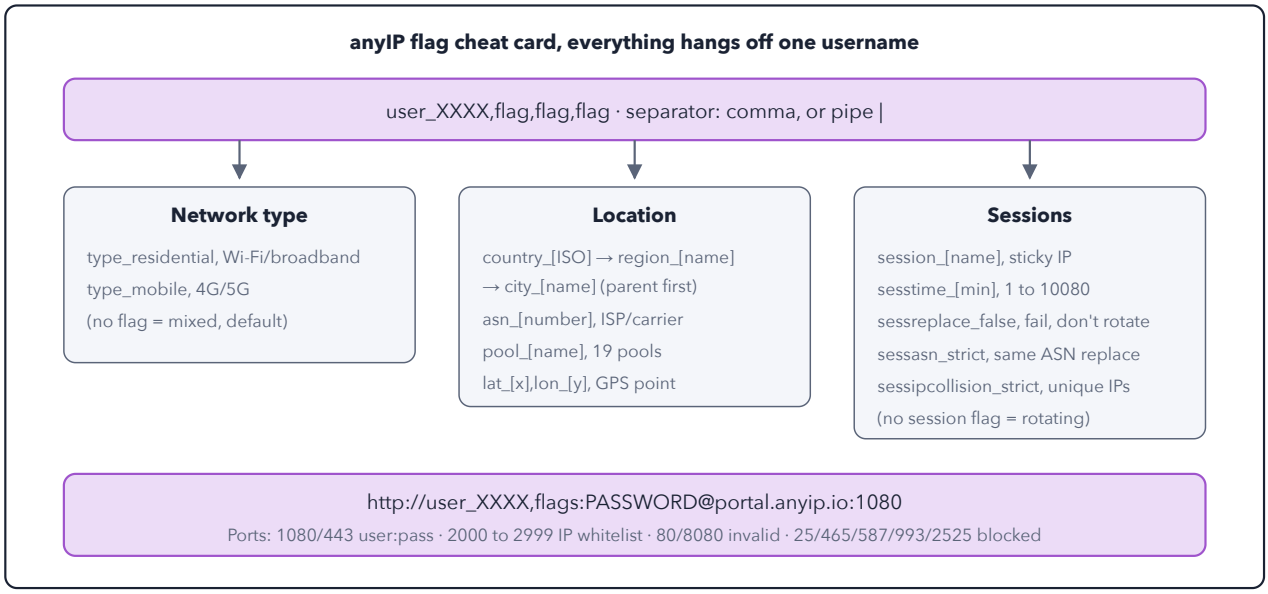


Figure 11.1, One-page flag cheat card: the smart username spine, its three flag families, and the connection string with valid ports.

Connection string & ports quick reference

One hostname, one string format, and a short list of ports. HTTP, HTTPS, and SOCKS5 all work on the same port.

Item	Value	Notes
Connection string	`http://user_XXXX,flags:PASSWORD@portal.anyip.io:1080`	Flags ride inside the username, before the `:PASSWORD`
Ports 1080 / 443	Username/password auth	Standard method; HTTP/HTTPS & SOCKS5 on the same port
Ports 2000 to 2999	IP whitelist auth (no password)	Pick your port in the dashboard; format `portal.anyip.io:2000`
Ports 80 / 8080	Invalid	Won't connect, a classic first-day mistake
Ports 25 / 465 / 587 / 993 / 2525	Permanently blocked	Email ports, the network is not designed for mailing
Entry nodes	`portal-na` · `portal-eu` · `portal-as`	Regional gateways (Americas / Europe-Africa / Asia-Oceania); `portal.anyip.io` auto-routes to the nearest

Error quick-lookup

Every anyIP error returns an `X-Anyip-Error` header with a code. Read the HTTP status first, it tells you which family of problem you have and what to try first.

Status	Means	First fix
407	Authentication problem, wrong credentials, bad format, or wrong port for your auth method	Check user/pass and use port 1080/443 (or your whitelist port 2000 to 2999)
400	Bad flag or bad request, invalid country, ASN, coordinates, session duration, type, or pool	Re-check flag spelling and values against the cheat sheet above
403	Forbidden host, the target is intentionally restricted (e.g. financial/government sites)	Don't retry; the restriction cannot be removed
409	Collision/strict-session exhausted, no more IPs satisfy `sessasn_strict` or `sessipcollision_strict`	Relax the strict flags or reduce concurrent sessions
413	2000-unique-IP cap reached (`max_ips_reached_for_user`)	Reuse sessions instead of minting endless new ones
502 / 503 / 504	No peers available, peer busy/not found, or peer/gateway timeout	Relax targeting (drop city/ASN flags), then retry with backoff

The 30/60/90-day learning path

You don't need the whole book on day one. Sequence your learning: get a working connection first, harden it, then scale it.

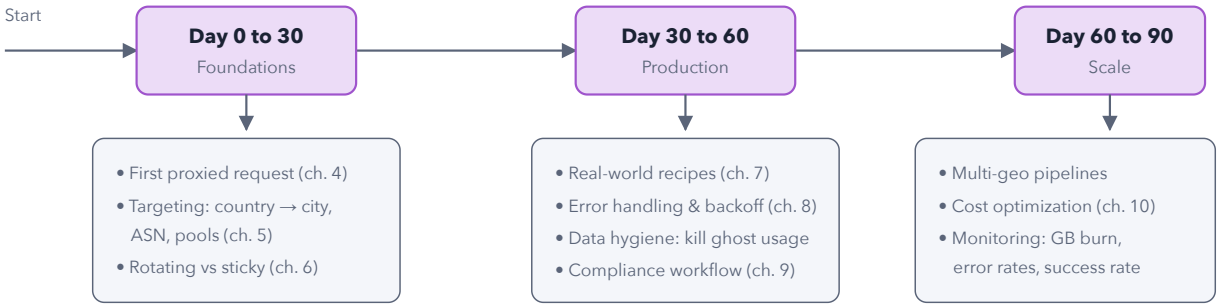


Figure 11.2, The 30/60/90-day learning path: foundations first, then production hardening, then scale.

> 💼 **Business angle**
>
> A team that follows this path converts proxy spend into predictable data delivery: by day 30 you have a working connection, by day 60 error handling and data hygiene stop GB waste (disabling images/video alone can cut usage by up to 80%), and by day 90 you can model cost per successful request across geos, the number your budget owner actually cares about.

Production-readiness checklist

Run this before any job goes live. Every item maps back to a chapter in this book.

- [ ] Connection string verified with `curl -x "http://user_XXXX:PASSWORD@portal.anyip.io:1080" http://ip-api.com/json`, IP and country match intent (ch. 4).
- [ ] Correct port for my auth method: 1080/443 for user:pass, my assigned 2000 to 2999 port for IP whitelist, never 80/8080 (ch. 4, 8).
- [ ] Targeting flags follow the hierarchy (country before region before city) and use pipe `|` if my tool rejects commas (ch. 5).
- [ ] Rotating vs sticky decided deliberately; sticky jobs have a named `session_` and a `sesstime_` that fits the task (ch. 6).
- [ ] Strict session flags (`sessreplace_false`, `sessasn_strict`, `sessipcollision_strict`) used only where the job truly needs them, they can exhaust with 409s (ch. 6, 8).
- [ ] Retries use backoff with a cap, no while-True loops that burn GB on error pages (ch. 7, 8).
- [ ] Image loading and video autoplay disabled where possible to prevent ghost usage (ch. 8).
- [ ] Error handler reads the `X-Anyip-Error` header and routes by status: fix 407/400, stop on 403, relax targeting on 502 to 504 (ch. 8).
- [ ] Job passed the pre-job compliance checklist: authorized target, robots.txt respected, rate limits set, personal data minimized (ch. 9).
- [ ] Plan health checked: active subscription time AND remaining GB (you need both to connect); renewal within 7 days of expiry to keep rollover (ch. 10).
- [ ] Proxy password rotated if usage looks unexplained, compromised credentials are a real ghost-usage culprit (ch. 8).

📌 Key takeaways

- Thirteen flags cover everything: two network types, six location controls, and five session controls, joined by comma or pipe on one username string.
- Ports are simple: 1080/443 for user:pass, 2000 to 2999 for IP whitelist, 80/8080 never work, and the five email ports are permanently blocked.
- Errors sort by status: 407 = auth, 400 = bad flag, 403 = forbidden host, 409 = strict-session exhaustion, 413 = the 2000-IP cap, 502 to 504 = relax targeting and retry.
- Master the product in 90 days: foundations (connect, target, sessions), then production (recipes, errors, hygiene, compliance), then scale (multi-geo, cost, monitoring).
- The production-readiness checklist is the whole book in eleven boxes, run it before every new job.

🔗 Go deeper

- anyIP documentation, source of the flag reference, ports, and error codes on this page.
- anyIP dashboard, credentials, proxy profiles, IP whitelisting, and plan status.
- ip-api.com, the one-line verification endpoint used throughout this book.
- cloudflare-quic.com, verify HTTP/3 (QUIC) tunneling over SOCKS5 UDP (ch. 8).

12. Resources

> ⚡ **The gist**
>
> - Every official anyIP link from this book, in one place, grouped by what you need.
> - A docs map: which page answers which question, so you never search twice.
> - Bookmark three links (Quick Start, Configuration Reference, Troubleshooting) and you can solve 90% of day-to-day issues.
>
> *Reading time: ~3 min*

You don't need to memorize the docs, you need to know which page answers which question. The map below routes the five questions you'll actually ask to the page that settles them; the grouped lists that follow cover everything else.

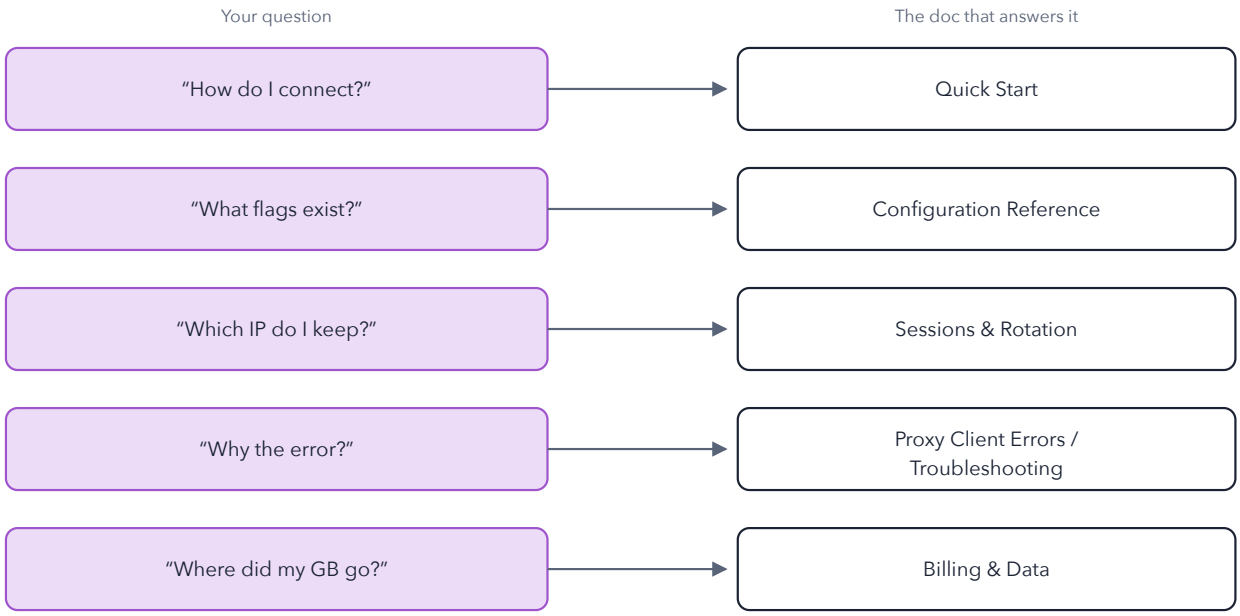


Figure 12.1, Docs map: the five questions you'll ask most, and the page that answers each one.

Product & marketing

- Homepage, the elevator pitch, network size, and ban-rate claims in one page.
- Pricing, the four per-GB tiers behind the cost model in Chapter 10.
- Residential proxies, what the residential pool offers and when to pick it.
- Mobile proxies, the highest-trust IP type, for the toughest targets.
- Locations, browse coverage before you commit to a geo-targeting plan.
- Integrations, which browsers, scrapers, and tools plug in out of the box.
- Status page, check here first when connections fail network-wide.
- Dashboard, credentials, usage, and whitelists live here.
- Account & plans, top up, renew, and manage your subscription (remember: you need both time and data).

Documentation

Docs home: anyip.io/docs.

- Introduction, the 10,000-foot view of how the service works.
- Quick Start, zero to first proxied request; the source for Recipe 4.1.
- Configuration Reference, every smart-username flag; the canonical companion to Chapter 3.
- Authentication Methods, user/pass vs IP whitelist, and which ports go with which.
- Network Types & Technical Specs, residential vs mobile pools and their technical limits.
- Location Targeting, country, region, city, ASN, and GPS flags with the must-specify-parent rule.
- Sticky vs Rotating Sessions, session flags, smart replacement, and rotate-on-demand.
- UDP Proxy Support, HTTP/3 (QUIC) tunneling for a modern traffic footprint.
- Integration Guides, step-by-step setup for common browsers and tools.
- Regional Pools Reference, all 19 named pools for broad-region targeting.
- Supported Countries, the full country list with codes to drop into your username.
- Proxy Client Errors, the error-code map from Chapter 8, in full.
- Troubleshooting, first-fix steps for the most common failures.
- Account Access Troubleshooting, locked out of the dashboard? Start here.
- Billing & Data, top-up vs renew, rollover rules, and where "ghost usage" comes from.
- Crypto Payment Troubleshooting, if you pay in crypto and the payment doesn't register.
- Dashboard V2 Migration, what changed if you joined before the new dashboard.
- API Reference, manage the service programmatically instead of via the dashboard.
- LLM index (llms.txt), a machine-readable docs index; handy for feeding the docs to an AI assistant.

Tools

- Proxy formatter, build a correct connection string without hand-typing flags.
- Proxy checker, verify a proxy is live and reporting the location you expect.
- Video tutorial: create a profile, a short walkthrough if you prefer watching to reading.
- Country data endpoint, the live list of targetable countries, as JSON.
- Region data endpoint, valid region names for a country (fill in the code).
- City data endpoint, valid city names for a region; use these exact spellings in your flags.

Testing endpoints

- ip-api.com/json, the IP-echo endpoint used in every verification step in this book.
- cloudflare-quic.com, confirms whether your traffic is really riding HTTP/3 (QUIC).

External standards (RFCs)

- RFC 1928, SOCKS Protocol Version 5, what actually happens on port 1080.
- RFC 9114, HTTP/3, the standard behind the QUIC tunneling in Chapter 8.
- RFC 9309, Robots Exclusion Protocol, the robots.txt standard your compliant scraper respects (Chapter 9).

📌 Key takeaways

- Route questions with the docs map: connect → Quick Start; flags → Configuration Reference; sessions → Sessions & Rotation; errors → Errors/Troubleshooting; usage → Billing & Data.
- Before debugging, check the status page, if the network is down, no flag will fix it.
- Use the live data endpoints to get exact region/city spellings instead of guessing them.

🔗 Go deeper

- Quick Start, if you bookmark one link, make it this: first request in under a minute.
- Configuration Reference, the single source of truth for every smart-username flag.
- Troubleshooting, your first stop when anything breaks.

Conclusion

You opened this book because your requests were dying with 403s and CAPTCHAs, and because a "German price check" was quietly serving you the US page. The fix was never a magic setting. It was a mental model, a bit of hygiene, and the judgment to know which jobs to run.

You now have all three. The specifics are behind you, chapter by chapter, and they add up to one shift: you can make the network do what you want on purpose, from a country-pinned request to a stable multi-step session, and explain the cost and the reasoning when someone asks.

If you take one idea with you, take this: **a ban is not bad luck. It is a signal you can engineer around.** Rate, reputation, and geography are the three levers, and every technique in this book is just a way of pulling one of them deliberately instead of hoping. Once bans become an engineering parameter, your data stops being a gamble and starts being something you can build decisions on.

So pick one real job, the smallest authorized thing you actually need, and ship it. Use the 30/60/90 path in Chapter 11 as your map, and keep Chapter 9 close: the fastest way to lose a proxy program is to point it somewhere you had no business pointing it. Do the boring, honest version well, and reliable web data becomes a tool you reach for without thinking.

📌 **Key takeaways**

- Blocks come from three levers, request rate, IP reputation, and geography; you now control all three on purpose.
- The whole system is driven from one username string, so changing behaviour never means changing code plumbing.
- Treat bans as an engineering parameter, not luck, and your data becomes trustworthy enough to decide on.
- Start with one small authorized job, follow the learning path, and let the ethics chapter keep you out of trouble.

A note on this guide

This book is an independent guide, written to be useful rather than promotional. Every product fact in it was drawn from anyIP's own public documentation and cross-checked; where the marketing and the docs disagreed, the book says so and sides with the docs. Where a number is a marketing figure, it is labelled as one. The goal throughout was to be pertinent and objective, not partial.

A few opinions did make it in, mostly about what "good" looks like: rate-limit like a good citizen, prefer clean failure to silent fraud, and never point a scraper somewhere you are not authorized to go. Those are stated as judgements, not facts, and Chapter 9 explains the reasoning so you can disagree on purpose.

One honest limitation: tools change. Ports, flags, pricing, and limits move over time, and a printed guide is a snapshot. Treat the specifics here as a strong starting point, and confirm anything load-bearing against the live documentation before you depend on it. The mental model ages well; the exact numbers may not.

> 💡 **Tip**
>
> Found something out of date, or a place where the book could be clearer? That feedback is the most valuable thing a reader can send, and it makes the next edition better for everyone.

Acknowledgments

A practical book stands on other people's careful work. This one is no exception.

Thank you to the anyIP team for documentation thorough enough to teach from. The clarity of their guides, error reference, and billing rules is what made an accurate, honest guide possible; any errors that remain are the author's, not theirs. (As the note before this makes clear, gratitude is not endorsement: this remains an independent, unofficial book.)

Thank you to the authors and maintainers of the open standards this book leans on, especially the Robots Exclusion Protocol (RFC 9309) that underpins the ethics chapter, and to the maintainers of the everyday tools, curl, Python, and its ecosystem, that make the recipes runnable on any machine.

And thank you, the reader, for choosing to learn the boring, honest version of this craft. Reliable web data is a quiet superpower, and the people who use it responsibly are the reason networks like this stay clean enough to be useful at all.

About the author

Christophe Paka is a self-taught developer and digital nomad with a deep passion for technology. He works at the intersection of building and hiring, as a technologist who recruits technologists rather than a recruiter who happens to work in tech. His YouTube channel on careers and jobs in tech has passed 700,000 views. He created this series to teach established and emerging technologies to more people, in a straightforward, visual format that gets to the point. He is always open to connecting with startup ecosystems across America, Europe, Asia, and Africa.

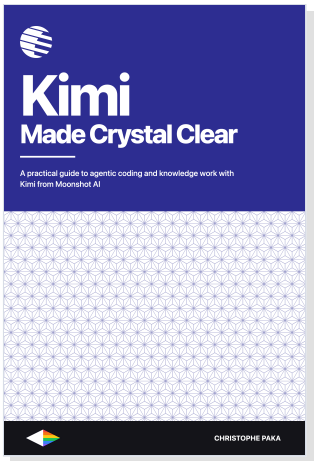
Thank you for reading

Thank you for spending your time with this book. If it made anyIP a little clearer and left you more confident to use it, it did its job. Keep it on a second monitor, come back to the recipes, and make the checklists your own.

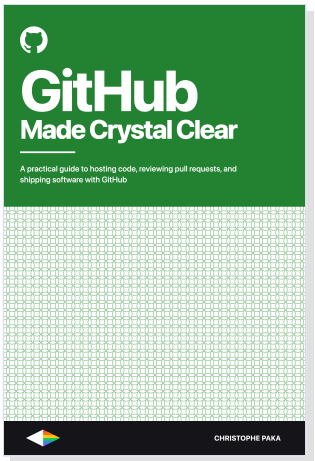
Keep exploring the series

More short, visual guides in the **Made Crystal Clear** series:

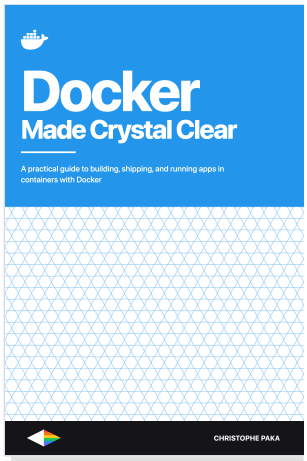
Git



Kimi



GitHub



Docker

We would love your feedback

Found an error, a rough spot, or something that clicked? Tell us and help improve the next edition of this book. Scan the code or visit:

madecrystalclear.dev/anyip/feedback

You can pick the chapter, choose a category, add a note, and attach a screenshot.

ANYIP · MADE CRYSTAL CLEAR

Turn IP bans from a mystery into an engineering parameter.

Your scrapers keep dying with 403s and CAPTCHAs. Your German price check is silently served the US page. This guide turns IP bans from a mystery into an engineering parameter, and shows you how to collect reliable, geo-accurate web data with anyIP, explained visually and hands-on.

WHAT YOU'LL LEARN

- [ ] Why a single IP gets blocked, and where a proxy network fits in your stack
- [ ] How to control everything from one 'smart username' string
- [ ] Targeting any country, city, carrier, or GPS point
- [ ] Rotating and sticky sessions without the surprises
- [ ] Three production-grade collection recipes in Python
- [ ] Diagnosing every proxy error and cutting wasted gigabytes
- [ ] Staying ethical and compliant, and modelling the real cost

Who it's for: *Developers, growth and marketing engineers, and data or QA teams who need reliable web data and geo-accurate testing.*

Christophe Paka
Christophe Paka is a self-taught developer, digital nomad, and creator of a 700K-view YouTube channel on tech careers. He builds this series to make established and emerging tech clear for everyone.

Explore the full **Made Crystal Clear** series and more books at **madecrystalclear.dev**