Pihat Remote-Control Service — Setup & Operation Guide

How to set up and operate remote control of your Pihat cellular devices. Covers the two control paths — the **HTTP API over the Cloudflare tunnel** (real-time, when the device is online) and **SMS** (out-of-band, when the device has gone dark) — plus how to add devices, send commands, and recover a device that has lost its data path.

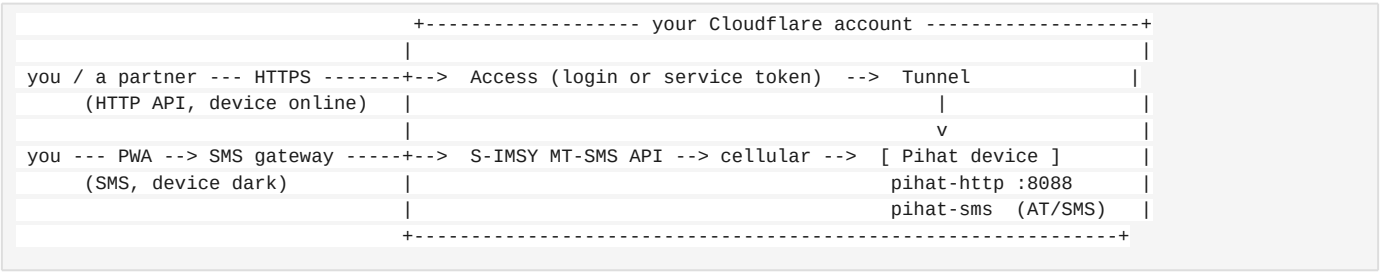
This guide is **BYO-CF**: everything runs in *your own* Cloudflare account. It works the same on both supported HATs (Quectel RG255C-GL / EBD070 and Teltonika ALA440 / EBD021) — the device software is hardware-agnostic.

For the underlying pieces, see: - `HOWTO.md` — provisioning a device (golden image, tunnel, SSH) - `app/README.md` — the `pihat-sms` and `pihat-http` modules - `sms-gateway/README.md` — the gateway Worker + management console (HTTP + SMS) - `MODEM-AT-COMMANDS.md` — raw AT diagnostics

Contents

1. How the service works

Each device runs two small daemons that share **one** set of command handlers and **one** modem (serialised by a file lock), so a command added once works on both. They're two front doors on the same controls — **not** competitors:



Path	When to use	Carries
HTTP API (`pihat-http`) over the tunnel + Access	device is online	real-time JSON, no SMS cost, partner automation via service tokens
SMS (`pihat-sms`) via the management console	device is **dark** (no data / tunnel down)	the same commands, out-of-band over the cellular control plane

The **management console** (the gateway's PWA) drives a device over **either** path — you pick the transport per command (**Auto / HTTP / SMS**), so you choose a *moment*, not a tool. **Auto** uses HTTP when the device is online and falls back to SMS when it's dark. Partners and automation can also call the HTTP API directly with a service token.

The HTTP API binds to `127.0.0.1` only — it is reachable **solely** through the tunnel, with access enforced at the Cloudflare edge. SMS keeps working even when the data session is down, which makes it the recovery lever (see §8).

---

2. Prerequisites

- A provisioned Pihat device with a working **tunnel** and **SSH** (see `HOWTO.md`). You should already be able to reach it at `pi-<id>-ssh.<your-domain>`.
- A Cloudflare account with your domain in it, and **Zero Trust** enabled (the team domain, e.g. `your-team.cloudflareaccess.com`).
- `cloudflared` installed on your laptop (for tunnel SSH and the API client examples).
- For SMS: the `pihat-sms-gateway` Worker deployed and a webhook channel configured in the S-IMSY portal (see `sms-gateway/README.md`).

> **Windows / Git Bash note:** on some setups the schannel TLS stack fails the OCSP revocation check and `curl` errors out. If you hit that, add `--ssl-no-revoke` to the `curl` commands below.

---

3. Set up a device

The control modules run on any device with a HAT. **Newer Pihat golden images ship with them pre-installed** — check with `curl -s localhost:8088/health` (an `{"ok": true}` means `pihat-http` is already running). Otherwise install (or update) with the one-line installer — idempotent, safe to re-run:

```
curl -fsSL https://downloads.pihat.net/install.sh | sudo bash
```

(Or run the modules by hand from a repo checkout in `app/`: `sudo ./install-sms.sh` then `sudo ./install-http.sh`.)

Set a device admin password — the SMS daemon **ignores all SMS until you do** (fail closed):

```
echo 'SMS_ADMIN_PASSWORD=choose-a-strong-password' | sudo tee -a /opt/pihat/config.env
sudo chmod 600 /opt/pihat/config.env
sudo systemctl restart pihat-sms
```

Verify locally:

```
systemctl is-active pihat-sms pihat-http
curl -s localhost:8088/health      # -> {"ok": true}
```

> The admin password is **machine-to-machine** — the SMS gateway prepends it for you and an end user never types it. See §9.

Details of the modules, every config key, and caveats: `app/README.md`.

---

4. Expose the HTTP API (Cloudflare)

This is "just config in CF", the same pattern as the SSH ingress. Do it once per device. (Substitute your device id and domain — e.g. `pi-a1b2c3d4`, `example.com`.)

4.1 Add the ingress rule

On the device (SSH) — edit `/etc/cloudflared/config.yml` so the api hostname maps to the local API, alongside the existing SSH rule, **above** the `404` catch-all:

```
ingress:
 - hostname: pi-<id>-ssh.<your-domain>
   service: ssh://localhost:22
 - hostname: pi-<id>-api.<your-domain>      # <-- add this
   service: http://localhost:8088
```

```
  - service: http_status:404
```

Then:

```
sudo systemctl restart cloudflared
```

4.2 Add the DNS record

CF dashboard -> **DNS -> -> Add record**: - **Type:** `CNAME` - **Name:** `pi-<id>-api` - **Target:** `<tunnel-id>.cfargotunnel.com` (the tunnel id is the `tunnel:` line in `config.yml`) - **Proxy status: Proxied** (orange cloud — required) - **Save**

4.3 Put Cloudflare Access in front

CF **Zero Trust -> Access -> Applications -> Add an application -> Self-hosted**: - **Application domain:** `pi-<id>-api` . `<your-domain>` - Attach **two policies** (create them once, then reuse on every device):

Policy	Action	Include rule	Purpose
`Operators`	**Allow**	Emails ending in `@your-domain` (or specific emails)	a person logs in via browser
`Partner token`	**Service Auth**	Service Token -> your partner token (or *Any Access Service Token*)	machine/partner auth, **no** login

- **Save the application.** Confirm **both** policies appear in the app's Policies list (Access evaluates Bypass/Service-Auth policies first).

> **Policies live in a library and must be *attached* to each app.** Creating a policy under *Access -> Policies* does nothing until you add it to the application. The "Used by applications" count only rises once attached.

4.4 Create a partner service token

CF **Zero Trust -> Access -> Service Auth -> Service Tokens -> Create Service Token**. Copy the **Client ID** (ends in `.access`) and **Client Secret** — the secret is shown **once**. Store it in your password manager; never commit it.

4.5 Verify

From your laptop:

```
# No credentials -> blocked at the edge (302 redirect to login)
curl -s -o /dev/null -w "%{http_code}\n" https://pi-<id>-api.<your-domain>/health
# -> 302

# With the partner token -> straight through
curl -s https://pi-<id>-api.<your-domain>/health \
 -H "CF-Access-Client-Id: <client-id-ending-in.access>" \
 -H "CF-Access-Client-Secret: <client-secret>"
# -> {"ok": true}
```

The full endpoint list is in `app/README.md`.

---

5. Set up the management console

The gateway is a Cloudflare Worker that serves the **management console (PWA)** and brokers commands to your devices over **HTTP** (real-time, via the tunnel) **and SMS** (out-of-band, via the S-IMSY network API) — one console, both transports. The one-time Worker setup (D1 database, schema, secrets, deploy) and the SMS webhook-channel configuration are in `sms-gateway/README.md`.

Enable the HTTP transport. Give the gateway the **Cloudflare Access service token** as Worker secrets so it can call devices' APIs server-side (the token never reaches the browser). Reuse the *same* token as §4 — one token works across every device that shares the `Partner token` policy:

```
cd sms-gateway
npx wrangler secret put CF_ACCESS_CLIENT_ID       # the …■.access Client ID
npx wrangler secret put CF_ACCESS_CLIENT_SECRET   # the Client Secret
npx wrangler deploy
```

Enrol a device — over HTTP, no ICCID typed. In the console click **+ Add by HTTP host** and enter the device's api hostname (`pi-<id>-api.<domain>`). The gateway calls the device's `/info`, captures its **ICCID + IMEI + hardware**, and creates the registry entry. The **ICCID stays the operational handle** (your service identifier, what you use with customers); the **IMEI is paired** as the hardware identity and refreshes on every `/info`, so it never goes stale against a moved SIM. SMS commands then use the same entry. (Model: `sms-gateway/DESIGN-box-registry.md`.)

Notes: - Devices also **auto-register** from an inbound SMS (an MO reply), but that depends on the carrier-dependent reply path — **HTTP enrolment is the reliable route**. - Per-device admin passwords are stored in the gateway and prepended automatically — the app never sees them; safe rotation is built in.

---

6. Operating the service

Remote shell (CF SSH)

```
ssh -o ProxyCommand="cloudflared access ssh --hostname %h" \
   <user>@pi-<id>-ssh.<your-domain>
```

First connection opens a browser for the Access login. This rides the tunnel over the modem — it works with wifi off, exactly as in the field.

HTTP API (device online)

Reads are `GET`, actions are `POST`. Authenticate with the service token (partner automation) or be logged in via the browser (people). Examples:

```
H1="CF-Access-Client-Id: $CF_ID"; H2="CF-Access-Client-Secret: $CF_SECRET"
BASE="https://pi-<id>-api.<your-domain>"

curl -s "$BASE/status"  -H "$H1" -H "$H2"          # uptime, IP, signal, network
curl -s "$BASE/info"    -H "$H1" -H "$H2"          # IMEI / IMSI / ICCID / make / model
curl -s -X POST "$BASE/restart-data" -H "$H1" -H "$H2"
curl -s -X POST "$BASE/network" -H "$H1" -H "$H2" \
    -H 'content-type: application/json' -d '{"mno":"EE"}'
```

Replies are `{"ok":true,"result":"…"}`.

Via the management console (HTTP or SMS)

In the **PWA**: select the device -> choose the command -> pick **Transport** (**Auto** / **HTTP** / **SMS**) -> **Send**. - **HTTP** returns the result **inline, instantly**, and logs it to Activity. - **SMS** sends out-of-band; the reply appears in Activity where the carrier delivers it. Each text is `<password> <command>` — the gateway adds the password. - **Auto** uses HTTP when the device's online indicator is green, else SMS.

Enrol a device first with **+ Add by HTTP host** (§5) — the console captures its api host, ICCID and IMEI for you, no number typed.

---

7. Command reference

The same capabilities are exposed on both paths. SMS phrasing is matched on **natural language**, not exact keywords (e.g. `change network to EE`).

Capability	HTTP	SMS (after the password)	Notes
Health	`GET /health`	—	liveness only

Status	`GET /status`	`status`	uptime, usb0 IP, signal, network
Signal	`GET /signal`	`signal`	CSQ + dBm, operator
Device info	`GET /info`	`device info`	IMEI / IMSI / ICCID / make / model
List networks	— *(SMS only)*	`list networks`	**slow** operator scan; deliberately not exposed on HTTP
Speed test	`POST /speedtest` `{"cap_mb":N}` -> poll `GET /speedtest-result`	—	usb0 down/up/ping; **saturates the link, costs data**, briefly starves the tunnel (async)
Reboot Pi	`POST /reboot`	`reboot`	
Reset modem	`POST /reset-modem`	`reset modem`	`AT+CFUN=1,1`
Restart data	`POST /restart-data`	`restart data`	restarts `pihat-lte`
Restart tunnel	`POST /restart-tunnel`	`restart tunnel`	restarts `cloudflared`
Recover	`POST /recover`	`recover`	reset modem -> restart data -> restart tunnel

Change network	`POST /network {"mno":"EE"}`	`change network to <name|code>`	disruptive; `EE`/`23430`/`auto`
Set password	(via gateway rotate)	`set-password <new>`	rotation is gateway-managed; see `sms-gateway/README.md`

Action commands are fire-and-forget; query commands need a reply channel to be useful (HTTP returns it inline; SMS replies where the carrier delivers MO-SMS).

---

8. Recovery & troubleshooting

Device has gone dark (no data / tunnel down)

This is exactly what SMS is for. The modem stays **registered** for SMS even when the data session is dead (different bearer), so an SMS gets through.

1. In the PWA, send **`recover`** to the device.
2. It resets the modem, restarts data, and restarts the tunnel (~60–90s).
3. Watch the tunnel return to **HEALTHY** in the CF dashboard, then reconnect.

If the data path is already restored but the tunnel still won't come back — e.g. you moved the SIM to a working-data carrier and `ping -I usb0` works, yet the dashboard still shows the tunnel down — send **`restart tunnel`** on its own. cloudflared crash-loops while the WAN is down and `systemd` can rate-limit it into a **stopped state**, so it doesn't always reconnect by itself once data returns; a single `restart tunnel` clears that. (`recover` does this too, as its last step.)

> (!) **Known resilience gap.** On the **ALA440 (EBD021)**, the persistent `connect-lte.sh` watchdog only re-DHCPs `usb0` — it does **not** re-dial the modem. So a modem-only ALA440 that loses its data call **cannot self-recover** and is stranded until an SMS `recover` (or a power-cycle). Building automatic re-dial into `connect-lte.sh` is a tracked follow-up. SMS is the manual lever in the meantime.

SMS command runs on the device but no reply comes back

If a command is delivered and **processed** by the device (the device's `pihat-sms` journal logs `running cmd_…`) but the **reply text never arrives**, the command *ran* — only the **return SMS** didn't get through. This is a **mobile-originated (MO) SMS delivery limitation on the SIM's current network**, not the device or the gateway: some networks don't complete MO SMS for this kind of traffic.

- **Use HTTP for reliable two-way control** when the device is online — the result returns in the same call, with no SMS reply-path dependency. (**Auto** transport picks HTTP automatically; this is the recommended path.)
- If you specifically need the **SMS reply** on a given SIM and it isn't arriving, it's a carrier MO-delivery matter — **contact your connectivity provider / support** to check MO-SMS routing for that SIM.
- **Action** commands (reboot, recover, restart-…) still take effect even when no reply returns — they're fire-and-forget.

Device registers and SMS works, but no data (`ping -I usb0` 100% loss)

Diagnose in this order (the control plane working tells you nothing about data):

1. **Antenna.** A missing/loose antenna or the wrong connector (data TX needs the **main** port) lets registration/SMS through but starves data. Check `AT+CSQ` (want 20+) and `AT+COPS=?` — a bare connector often sees only the strongest cell. Good signal is necessary but not always sufficient.
2. **Carrier / coverage / bars.** Scan with `AT+COPS=?` (status: 1 available, 2 current, 3 forbidden). Note: bars enforced in your core can pin a SIM to one carrier; a clean deregister (`AT+COPS=2`) then explicit select is the honest move-test.
3. **Core data routing / provisioning.** If the modem has a healthy PDP (real IP, `CGATT:1`, `CGACT:1,1`) but still passes no traffic, the data path is failing in the core — data and SMS are provisioned/routed **separately**. Check that data is provisioned for the ICCID on its current network and that a routing policy isn't dropping the user plane.

Raw AT command reference: `MODEM-AT-COMMANDS.md`.

HTTP API returns `302` with a valid token

Access isn't matching a Service-Auth policy. On the app's **Policies** tab, confirm `Partner token` is **attached** and its action is **Service Auth** (not Allow), with Include -> your service token (or *Any Access Service Token*).

Tunnel "down" after a good bootstrap

Usually a routing issue (ethernet/wifi stealing the default route from LTE). See the tunnel/route troubleshooting in `HOWTO.md`.

---

9. Security model

- **HTTP API binds `127.0.0.1` only.** It is never exposed directly — only via the tunnel, gated by Cloudflare Access at the edge. No device-side API auth to manage; Access is the gate.
- **Two ways in, scoped separately:** people via the `Operators` (Allow/login) policy; partners via per-token `Partner token` (Service Auth) policies. Service tokens are revocable at the edge, per device.
- **Device admin password** (SMS) is machine-to-machine: stored in `config.env` (mode 600) and in the gateway; the gateway prepends it so the app and end user never see it. Supports safe rotation with a two-password window so a lost rotation can't lock out a remote device (see `sms-gateway/README.md`).
- **Never commit / never paste in chat:** `config.env` (holds the CF API token), tunnel credential JSON, Worker secrets, and Access service-token secrets.

---

10. Add-a-device checklist

Repeat per device to keep the fleet in sync:

- [ ] Device provisioned, tunnel + SSH working (`HOWTO.md`)
- [ ] `sudo ./install-sms.sh` and `sudo ./install-http.sh`
- [ ] `SMS_ADMIN_PASSWORD` set in `config.env` (mode 600); `pihat-sms` restarted
- [ ] `curl localhost:8088/health` -> `{"ok":true}`
- [ ] Ingress rule `pi-<id>-api -> http://localhost:8088` added; `cloudflared` restarted (§4.1)
- [ ] DNS `pi-<id>-api` CNAME -> `<tunnel-id>.cfargotunnel.com`, **Proxied** (§4.2)
- [ ] Access app over `pi-<id>-api.<domain>` with `Operators` + `Partner token` attached (§4.3)
- [ ] Verified: no-auth -> `302`, token -> `{"ok":true}` (§4.5)
- [ ] Gateway has `CF_ACCESS_CLIENT_ID` / `CF_ACCESS_CLIENT_SECRET` secrets set (console's HTTP transport, §5)
- [ ] Enrolled in the console via **+ Add by HTTP host** (captures ICCID/IMEI over HTTP); responds over HTTP **and** SMS