pebble.

The User Guide

Turn a small FreeBSD box into a calm,
well-behaved router. One YAML file. No drama.

Brain-friendly edition. Short sections. Big signposts. Every chapter opens with a **TL;DR**, says roughly how long it takes, and ends with a **Done when** check, so you always know when you can stop. Skip about freely: nothing here punishes you for reading out of order.

Licence: AGPL-3.0-or-later

Written in British English

Start here: pick your path

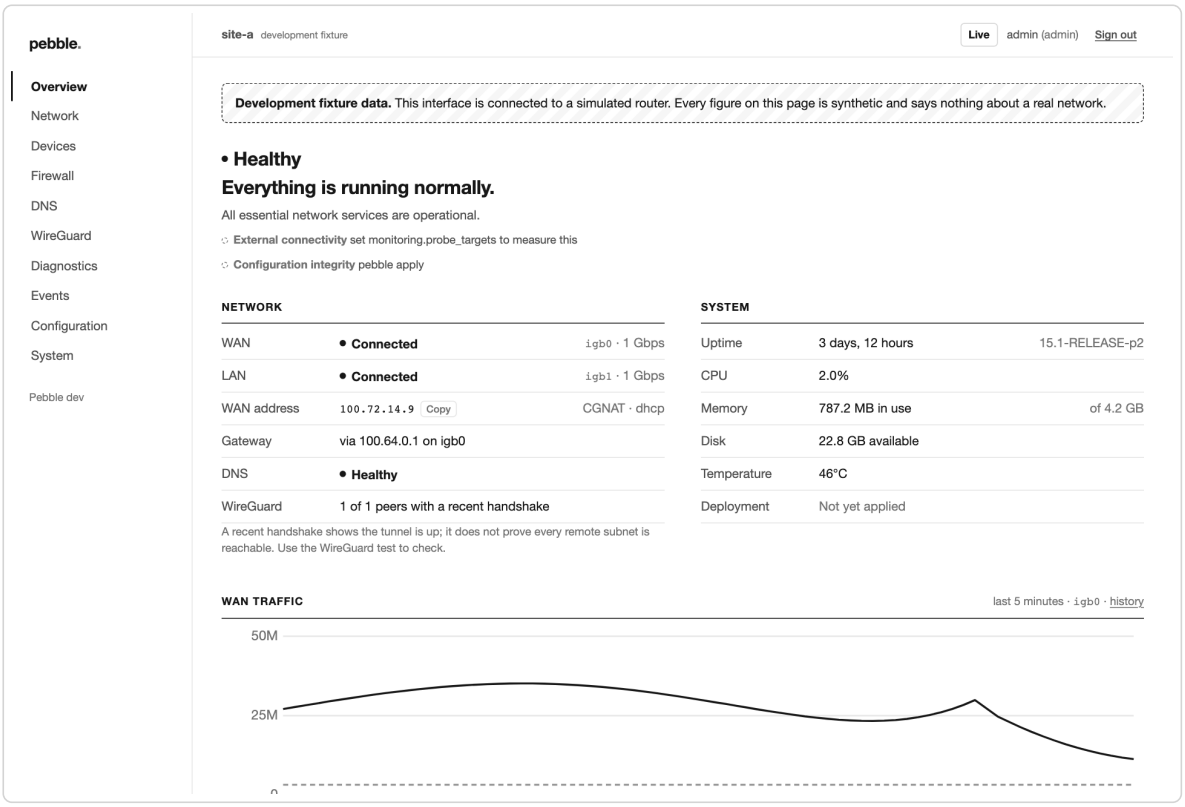
You do not need to read this whole guide. Find the row that matches you and jump.

I want to…	Go to	Time
Understand what Pebble actually is	Chapter 1	1 min
Build a router from a fresh FreeBSD install	Chapter 3	20 min
Change something without breaking the Internet	Chapter 5	5 min
Do a common job (reserve an IP, forward a port, block a device, use a VPN…)	Chapter 6	2 min each
Join two sites with WireGuard	Chapter 7	30 min
Work out why the Internet died at 3 a.m.	Chapter 8	5 min
Use the web interface	Chapter 9	10 min
Find out when a device starts acting strangely	Chapter 10	5 min
Fix something that's wrong *right now*	Chapter 12	—
Just the commands, please	Chapter 13	—

The key, so you can scan

Marker	Means
TL;DR	The whole section in three lines. Read only this if you're in a hurry.
	Roughly how long it takes
Done when	How you know you've finished and can go and make tea
Tip	Nice to know. Safe to skip.
Careful	Read this one. It's here because someone got bitten.
Honesty box	What has and hasn't been tested. No spin.
Screenshots	Taken from Pebble running against a *simulated* router. The dashed "Development fixture data" banner is there on purpose: every number in them is synthetic.

Honesty box. Pebble runs its author's home router (FreeBSD 15.1), and its logic is heavily unit-tested. It is still young, with one real deployment: keep a console within reach the first time you deploy it. The full status table is in the project README.



The web overview: four questions answered at a glance. (Simulated router.)

Contents

Start here: pick your path **3**

The key, so you can scan 3

1 Pebble in 60 seconds **7**

The mental model (this is the important bit) 7

What Pebble looks after 8

What Pebble is *not* 8

2 Before you begin **11**

Checklist 11

3 Install: from nothing to router **13**

Step 1: get the binary on to the box 13

Step 2: find out which port is which 13

Step 3: bootstrap (the big one) 13

Step 4: check, then confirm 14

4 A quick tour of pebble.yaml **15**

5 The golden loop: changing things safely **17**

The loop, step by step 17

Which changes need confirming? 17

The safety net, explained in one breath 17

Changed your mind? 18

6 Recipes: the everyday stuff **19**

7 Site-to-site WireGuard **27**

The idea 27

Step 1: keys 27

Step 2: the tunnel (on site A) 27

Step 3: say what may cross 28

Step 4: apply and test 28

8 Seeing what's going on **31**

The quick look . . . 31

The live dashboard . . . 31

The flight recorder (optional, and brilliant) . . . 32

9 The web interface **35**

Switch it on . . . 35

What's in there . . . 36

Making changes in the browser . . . 40

Two kinds of account . . . 40

10 Device DNA: noticing when a device changes **43**

Switch it on . . . 43

It needs time to learn . . . 44

When something looks odd . . . 45

Look closer at one device (investigations) . . . 46

Get told by email . . . 47

Get told on your phone . . . 48

Reports . . . 49

Forget everything . . . 49

11 Backups, recovery and upgrades **51**

Back up . . . 51

Restore . . . 51

Upgrade Pebble . . . 51

12 Something's wrong **53**

13 Cheat sheet **57**

14 Glossary (jargon, decoded) **61**

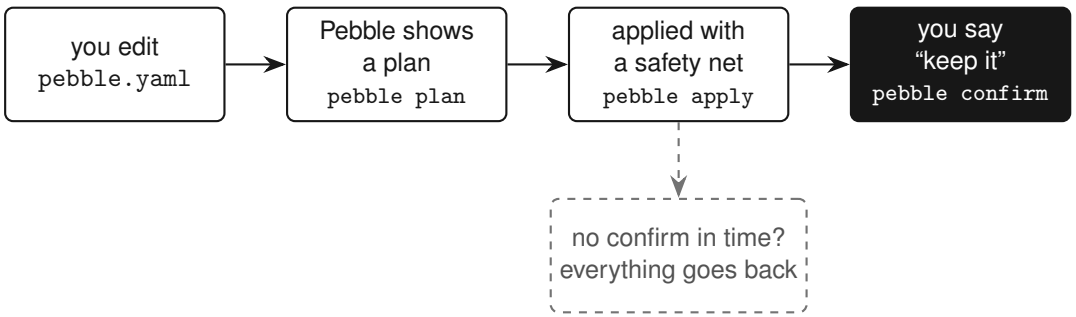
CHAPTER 1

Pebble in 60 seconds

TL;DR

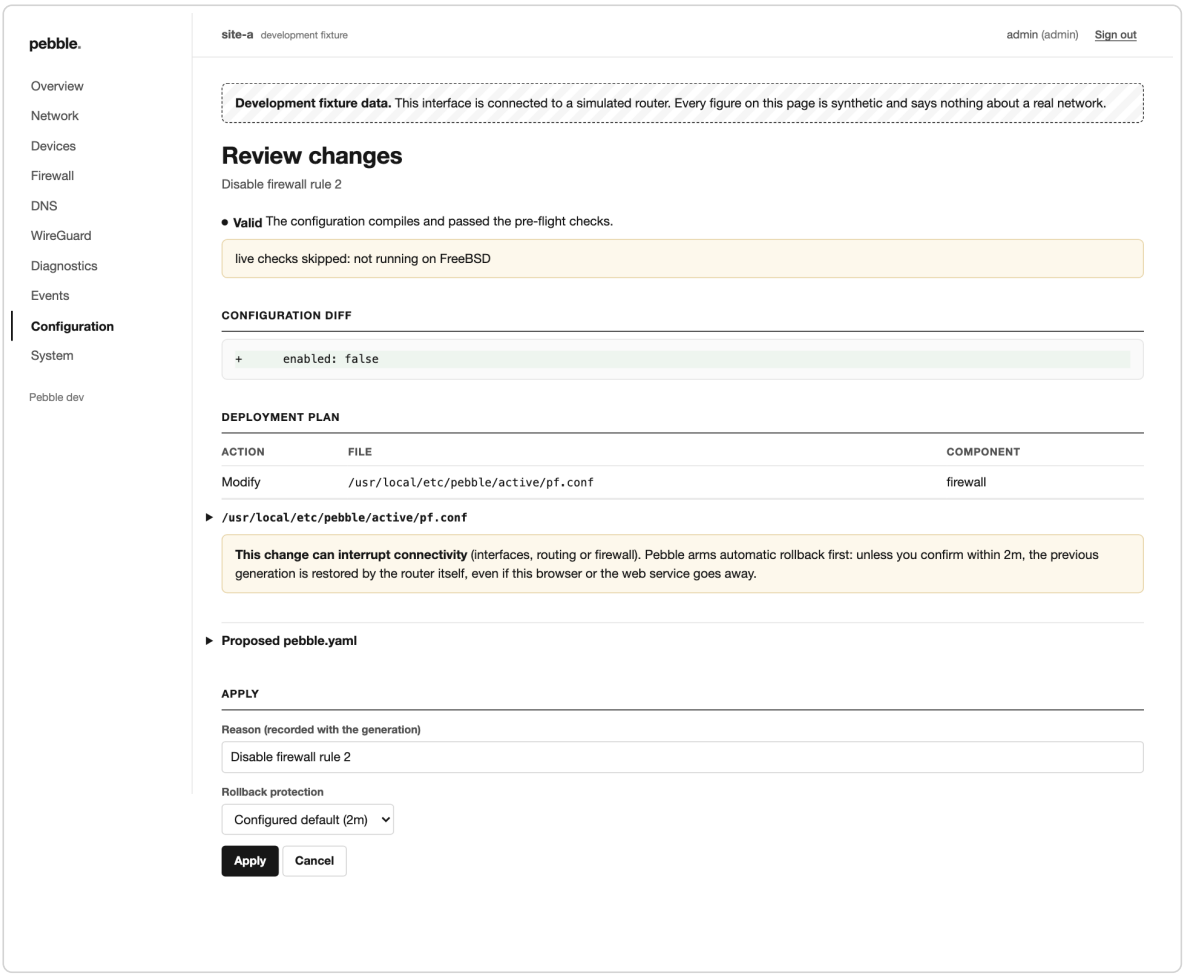
- You describe your router in **one file**: `pebble.yaml`.
- Pebble turns that into **normal FreeBSD configuration** and applies it **safely**.
- Then it **gets out of the way**. FreeBSD does the routing, not Pebble.

The mental model (this is the important bit)



That’s it. That’s the whole product. Everything else is detail.

In the web interface, the *plan* step is a review page showing exactly what will change:



Review before apply: the diff, the files that change, and a warning that this one can interrupt connectivity.

What Pebble looks after

- **WAN** (DHCP, static or PPPoE) and **one LAN**, with NAT
- **IPv6** from your ISP (DHCPv6 or static), handed to your devices automatically
- **Firewall** (PF): deny by default, your rules in order, port forwards
- **DHCP** (dnsmasq): addresses, reservations
- **DNS** (Unbound): your own resolver, DNSSEC, local names, ad blocking
- **WireGuard**: routed site-to-site tunnels, and VPN provider exits (ProtonVPN, Mullvad…) for just the devices you choose
- **Per-device Internet**: each device goes out via the WAN, a VPN, or not at all
- **Safe changes**: every change can be undone, automatically if needed
- **Optional extras**: a flight recorder, a terminal dashboard and a web interface

What Pebble is *not*

- Not OPNsense or pfSense. Deliberately smaller.

- No cloud, no telemetry, no account with anyone.
- No VLANs, no multi-WAN.

> **Tip.** “Pebble gets out of the way” is literal. Delete the `pebble` binary and your router keeps routing, filtering and handing out addresses. Everything it set up is ordinary FreeBSD.

Done when you could explain *edit* → *plan* → *apply* → *confirm* to a friend down the pub.

CHAPTER 2

Before you begin

TL;DR

- One amd64 box with **two network ports**, running **FreeBSD 15.1**.
- A **keyboard and screen** plugged in for the first setup.
- Ten minutes of nobody needing the Internet.

5 minutes to check

Checklist

- ☐ An amd64 machine with **two Ethernet ports**. A Fujitsu FUTRO S930 thin client is the reference box; most mini PCs and VMs are fine.
- ☐ **FreeBSD 15.1-RELEASE** installed (a plain install is perfect).
- ☐ **Local console** access: a real keyboard and screen, or a VM console. Not SSH. You'll see why in a moment.
- ☐ You know which port will face the **Internet (WAN)** and which faces **your network (LAN)**.
- ☐ You've picked a LAN address range. `192.168.10.1/24` is the default and is fine for most homes.
- ☐ If your ISP gave you a **user name and password**, have them handy: that's PPPoE (see the recipe in chapter 6).

⚠ **Careful.** The very first deployment replaces the firewall and the network settings. Do it from the **local console**. If you do it over SSH and something's wrong, you're locked out until the automatic rollback kicks in (two minutes by default), which feels like a very long time.

Done when every box above is ticked.

CHAPTER 3

Install: from nothing to router

TL;DR

1. Copy the `pebble` binary onto the box.
2. Run `pebble bootstrap` with your two ports.
3. Run `pebble confirm`. You have a router.

About 20 minutes, most of it waiting for packages

Step 1: get the binary on to the box

You need the `pebble` binary for FreeBSD. It's a single file with everything else built in, and comes in the release:

```
pebble-<version>-freebsd-amd64.tar.gz   # the binary is bin/pebble inside
```

Copy it across (USB stick, `scp`, carrier pigeon) and make it executable:

```
chmod +x pebble
```

Step 2: find out which port is which

```
./pebble interfaces
```

You'll see names like `igb0`, `igb1`, `em0` or `re0`.

> **Tip.** Not sure which is which? Unplug one cable, run the command again, and see which one changed to "no carrier". Pebble will **never guess** for you, on purpose.

Step 3: bootstrap (the big one)

From the **local console**, swapping in your own ports, address, name and time zone:

```
./pebble bootstrap --wan igb0 --lan igb1 --lan-address 192.168.10.1/24 \
    --hostname pebble --timezone Europe/London --apply
```

This one command:

- checks the machine is suitable
- installs **Unbound** and **dnsmasq**
- loads the kernel modules for the firewall and WireGuard

- installs Pebble itself, its service scripts and examples
- creates `pebble.yaml` from your answers
- applies it, with the safety net switched on

Want to see what it *would* do first? Add `--dry-run`. Already got the packages? Add `--skip-packages`. No Internet on the box? Use `--packages /path/to/pkgs`.

Step 4: check, then confirm

Plug a laptop into the LAN port. Does it get an address? Can it open a website? Then:

```
pebble confirm
```

> ⚠ **Careful.** You've got **2 minutes** to confirm. Don't confirm, and Pebble quietly puts the machine back exactly as it was. That's a feature, not a bug.

✅ **Done when** a laptop on the LAN can browse the web and `pebble status` says everything's fine.

CHAPTER 4

A quick tour of pebble.yaml

TL;DR

- It lives at `/usr/local/etc/pebble/pebble.yaml`.
- One section per job: `network`, `dhcp`, `dns`, `firewall`, `wireguard`…
- Unknown or misspelt settings are **errors**, not silently ignored.

5 minutes

Here's a complete, working home router. Read the comments; that's the tour.

```
version: 1

system:
  hostname: pebble
  timezone: Europe/London

network:
  wan:
    device: igb0          # the Internet side
    ipv4:
      mode: dhcp          # or: static (address and gateway), or pppoe
  lan:
    device: igb1          # your side
    ipv4:
      address: 192.168.10.1/24
  ipv6:
    mode: disabled        # or dhcpv6: see the IPv6 recipe

dhcp:
  enabled: true
  authoritative: true
  start: 192.168.10.100   # addresses handed out to devices
  end: 192.168.10.199
  lease_duration: 12h
  domain: home.arpa       # devices become name.home.arpa

dns:
  enabled: true
  recursive: true         # ask the root servers directly, no ISP in the middle
  dnssec: true            # check signatures
  cache_mb: 32
  blocking:
    enabled: true         # ad and tracker blocking (add lists later)
    response: nxdomain
    update_interval: 24h
    max_domains: 250000
    lists: []
    allowlist: []
    denylist: []
```

```
wireguard:
  interfaces: []

firewall:
  default_inbound: deny    # nothing from the Internet gets in...
  default_forward: deny    # ...or through, unless a rule says so
  default_outbound: allow
  nat:
    enabled: true
    outbound: wan

management:
  ssh:
    enabled: true
    allow_from:
      - 192.168.10.0/24    # SSH only from your LAN

updates:
  automatic_install: false

recovery:
  rollback_timeout: 120s  # how long you get to confirm
  keep_generations: 5     # how many old versions to keep for rollback
```

Tip. Every single setting, with an explanation, is in `/usr/local/share/examples/pebble/annotated.yaml`. Keep it open in another window.

Done when you can find your LAN address and DHCP range in the file without searching.

CHAPTER 5

The golden loop: changing things safely

TL;DR

```
pebble plan --diff    # look
pebble apply          # do
pebble confirm        # keep
```

1 minute per change, once it's a habit

The loop, step by step

1. **Edit** `/usr/local/etc/pebble/pebble.yaml`.
2. **Look before you leap** with `pebble plan --diff`. This changes *nothing*. It shows every file that would change, line by line, and warns you about anything risky.
3. **Apply** with `pebble apply`. Pebble validates everything with FreeBSD's own tools, takes a snapshot, applies it and runs health checks.
4. **Confirm** (if asked) with `pebble confirm`.

Which changes need confirming?

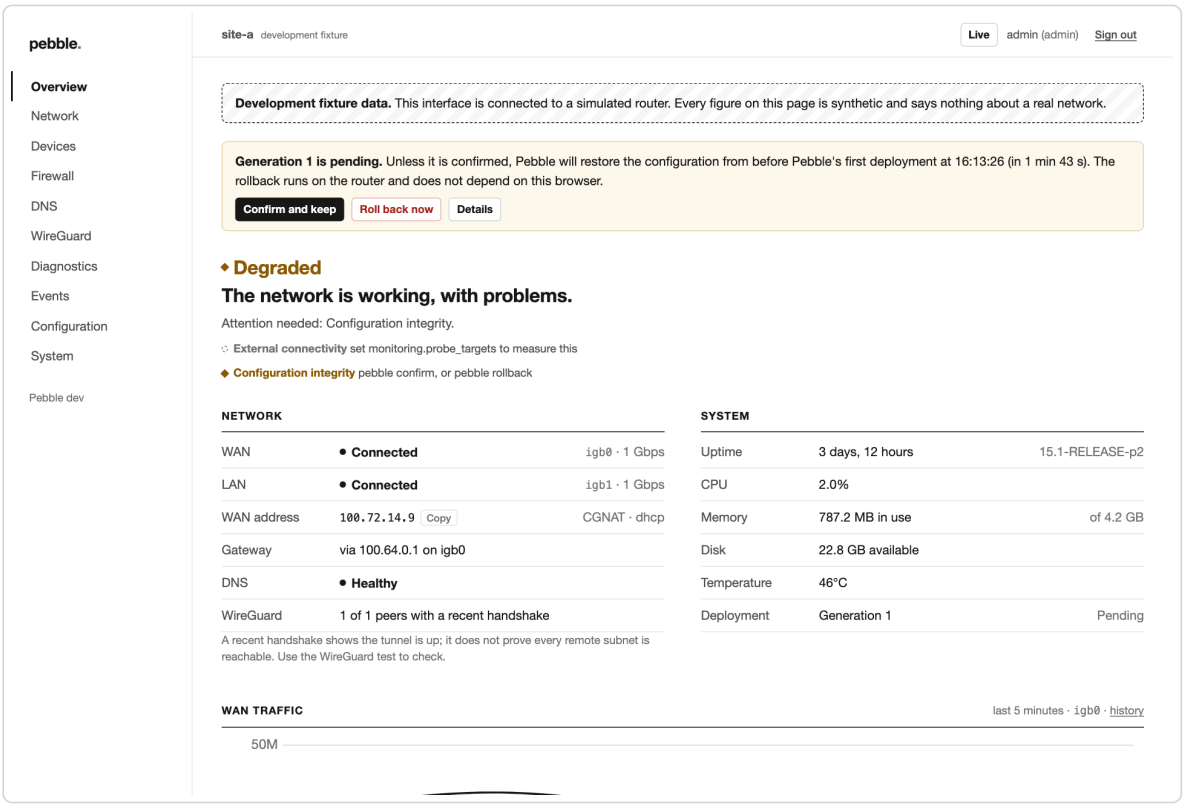
Change	Needs `pebble confirm`?
Firewall rules, port forwards	**Yes**
Interfaces, addresses, routes	**Yes**
WireGuard	**Yes**
Web interface settings	**Yes**
DNS records, blocklists	No, if health checks pass
DHCP reservations	No, if health checks pass

Want a confirmation step for everything? `pebble apply --confirm-always`. Need longer than two minutes? `pebble apply --rollback-after 5m`.

The safety net, explained in one breath

If you don't confirm in time, because SSH dropped, the firewall locked you out, the power went or you simply wandered off, **the previous working configuration comes back on its own**. Three independent things make sure of it: a timer, a check every minute, and a check at boot before the network even starts.

In the web interface, a pending change looks like this on every page:



The countdown banner. The rollback runs on the router, not in your browser.

Changed your mind?

```
pebble rollback                  # undo the pending change, or go back one version
pebble rollback --generation 3   # go back to a specific version
pebble status                    # see which generation you're on
```

Tip. Every successful apply is a numbered **generation**. They're like save points in a game. Pebble keeps the last five by default.

Done when "plan, apply, confirm" happens without you thinking about it.

CHAPTER 6

Recipes: the everyday stuff

TL;DR

Each recipe is: **add this YAML** → **run the golden loop**. That's every recipe. Pick the one you need and ignore the rest.

2 minutes each

Connect with a user name and password (PPPoE)

If your ISP gave you a user name and a password (common with DSL and fibre), your WAN is **PPPoE**. First put the password in a file that only root can read:

```
install -m 600 /dev/null /usr/local/etc/pebble/keys/pppoe.key
vi /usr/local/etc/pebble/keys/pppoe.key      # the password, on the first line
```

Then:

```
network:
  wan:
    device: igb0                  # the port your modem or ONT plugs into
    ipv4:
      mode: pppoe
    pppoe:
      username: you@your-isp.example
      password_file: /usr/local/etc/pebble/keys/pppoe.key
```

Check it: `service pebble pppstatus` shows the session's address once it's up, and `pebble diagnose` should show the WAN address and default route as healthy.

> **Careful.** The Internet drops for a few seconds while the session dials, so make this change from the LAN, not from outside. If your ISP needs a VLAN tag on the line, see *PPPoE* in `docs/networking.md`.

> **Tip.** Changed only the password? `pebble apply` won't notice (the configuration is the same), so run `service pebble pppup` to redial.

Turn on IPv6

If your ISP offers IPv6 (most hand out a "delegated prefix" by DHCPv6, over PPPoE or plain Ethernet), this is all it takes:

```
network:
  ipv6:
    mode: dhcpv6
    ula_prefix: fd5e:6b1d:2c3a:1::/64    # optional, but make your own (below)
```

Your devices then give themselves IPv6 addresses from the prefix the ISP hands out, and reach the IPv6 Internet. The firewall still blocks everything coming in that you didn't ask for.

There's no NAT on IPv6, so the firewall is what keeps your devices private.
The `ula_prefix` is a private, never-changing address range for your LAN. With it, the router answers DNS over IPv6 too. Make your own random one:

```
printf 'fd%s:%s%s:%s%s:1::/64\n' $(openssl rand -hex 5 | sed 's/../& /g')
```

Check it: `pebble diagnose` shows **IPv6 default route** and **LAN IPv6 prefix**. On a phone or laptop, visit an IPv6 test site.

> **Tip.** Your ISP gives a /48 or a /64 rather than the usual /56? Add `delegation_length: 48` (or 64). If it sets IPv6 up by hand instead, use `mode: static`: see *IPv6* in `docs/networking.md`.

> **Careful.** Devices with an `egress` setting (blocked, or sent through a VPN) are kept off IPv6, so they can't sneak round the setting. They carry on over IPv4 as before.

Give a device a fixed address

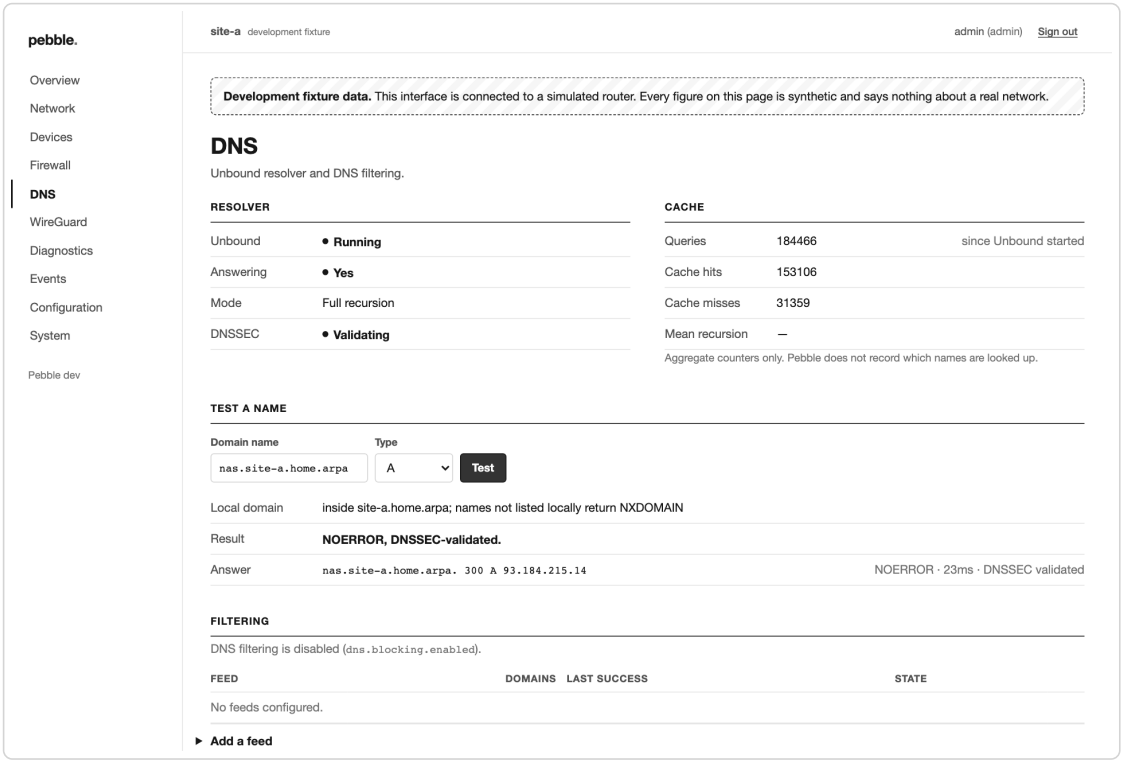
```
dhcp:
  reservations:
    - name: nas
      mac: "02:00:5e:00:53:10"
      address: 192.168.10.20
      hostname: nas
```

Bonus: `nas.home.arpa` now resolves on your network. To find a device's MAC address, look at `pebble dhcp leases`.

Add a local DNS name

```
dns:
  records:
    - name: printer.home.arpa
      type: A
      value: 192.168.10.30
```

Check it: `pebble dns test printer.home.arpa`



DNS, Test a name: every local policy that applies, then the real answer.

Forward a port to a device

```
firewall:
  port_forwards:
    - description: Web server on the NAS
      protocol: tcp
      external_port: "8443"
      target: 192.168.10.20
      target_port: "443"
      sources: [198.51.100.0/24]    # optional: who's allowed in
```

Careful. A port forward opens a hole to the whole Internet unless you set `sources`. Only forward what you really mean to.

Stop a device reaching the Internet

Give the device a reservation (above) and add one line:

```
dhcp:
  reservations:
    - name: kids-tablet
      mac: "02:00:5e:00:53:60"
      address: 192.168.10.60
      egress: none      # no Internet; the LAN and router still work
```

Or, in the web interface: **Devices** → set *Internet via* to **Blocked** → **Change…** → review → apply. To undo, set it back to **WAN**.

> **Tip.** This rule is checked *before* your own firewall rules, so no `pass` rule can quietly let the device back out. The device keeps its old access until it renews its lease onto the reserved address (or you reconnect it).

Block an address on the Internet for everyone

To stop *every* device reaching one Internet address or network, use a firewall rule:

```
firewall:
  rules:
    - description: Nobody talks to this tracker
      action: block
      interface: lan
      protocol: any
      source: lan
      destination: 203.0.113.80      # or a prefix: 203.0.113.0/24
```

Swap `source: lan` for one address (`192.168.10.30`) to block just one device from just that destination. Rules are checked **top to bottom; the first match wins**. Want to switch a rule off without deleting it? Add `enabled: false`. To block by *name* rather than address, use the DNS denylist instead (see *Block ads and trackers* below).

> **Careful.** Devices sent through a VPN exit or blocked with `egress` skip these rules for their Internet traffic: the per-device setting always wins.

Send one device through a VPN (ProtonVPN, Mullvad…)

Your VPN provider gives you a WireGuard configuration file. Copy its values into a WireGuard interface with `role: egress`, and save its `PrivateKey` value as `/usr/local/etc/pebble/keys/wg1.key` (mode 0600):

```
wireguard:
  interfaces:
    - name: wg1
      description: ProtonVPN UK
      role: egress                 # only devices you send use it
      address: 10.2.0.2/32         # [Interface] Address
      gateway: 10.2.0.1            # [Interface] DNS
      dns: [10.2.0.1]              # [Interface] DNS
      private_key_file: /usr/local/etc/pebble/keys/wg1.key
      peers:
        - name: proton-uk
          public_key: <[Peer] PublicKey>
          endpoint: <[Peer] Endpoint>
          persistent_keepalive: 25
          allowed_ips: [0.0.0.0/0]

dhcp:
  reservations:
    - name: laptop
      mac: "02:00:5e:00:53:21"
      address: 192.168.10.121
      egress: wg1                  # Internet via the VPN
```

> **Tip.** In the web interface, open **WireGuard** → **Upload a WireGuard configuration**, choose the provider's `.conf` file and check the filled-in form: the key is saved for you, and the interface is set up as a VPN exit. **New interface** creates a tunnel from scratch, with a fresh key pair whose public key the review page shows.

Or pick **VPN wg1** under *Internet via* on the **Devices** page. Everyone else keeps using your normal connection. What you get:

- **A kill switch.** If the tunnel drops, that device has *no* Internet, rather than quietly using your normal connection.
- **No DNS leaks.** The device is handed the VPN's DNS server when it next renews its lease.
- **Proof.** `pebble firewall test --in lan --src 192.168.10.121 --dst 1.1.1.1 --proto tcp --port 443` says the flow is passed by a `pebble-vpn-` rule.

> **Tip.** In use on a real router. Still, after setting it up, check that the device's public address is the VPN's (any "what is my IP" page).

Block ads and trackers (DNS)

```
dns:
  blocking:
    enabled: true
    lists:
      - name: stevenblack
        url: https://raw.githubusercontent.com/StevenBlack/hosts/master/hosts
        format: hosts
```

Apply, then fetch the list the first time:

```
pebble dns blocklist update
pebble dns blocklist test ads.example.com
```

After that it refreshes itself. A download that fails keeps the last good copy, so a broken list never breaks your DNS. Something you need got blocked? Add it to `allowlist`. Want to block a name yourself, with all its subdomains? Add it to `denylist` (for example `denylist: [tiktok.com]`).

Stop devices sneaking round your DNS (DNS-over-HTTPS)

Some apps ignore your DNS and talk to DoH servers directly. Block those servers by address:

```
firewall:
  ip_lists:
    update_interval: 24h
    lists:
      - name: doh
        url:
    https://raw.githubusercontent.com/dibdot/DoH-IP-blocklists/master/doh-ipv4.txt
  rules:
    - description: Block DNS-over-HTTPS bypass
      action: block
```

```
      interface: lan
      protocol: tcp/udp
      source: lan
      destination: list:doh
      ports: ["443"]
```

Then:

```
pebble firewall iplist update      # first download; it refreshes itself after
pebble firewall iplist test 1.1.1.1
```

See what a device looks up (and what got blocked)

Switch on the per-device DNS history (it needs the flight recorder):

```
dns:
  query_log:
    enabled: true
    retention: 24h        # keep it short: 1h to 7d
monitoring:
  enabled: true
```

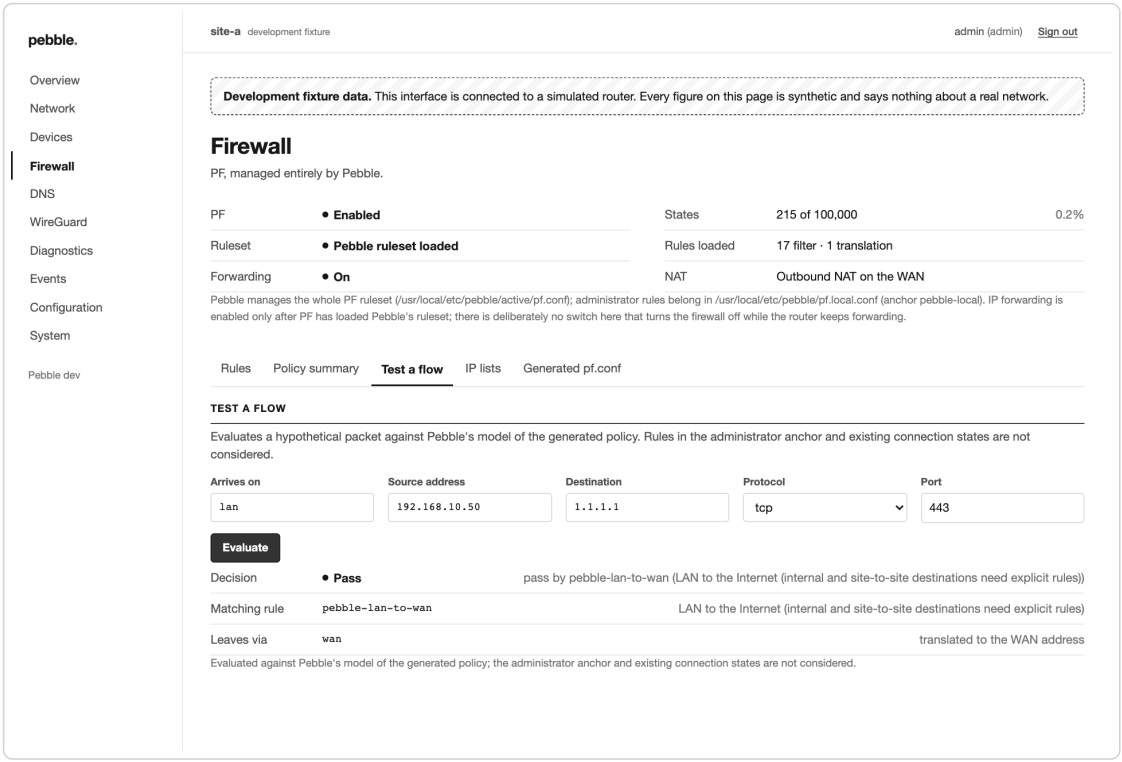
Then open **Devices**, click a device's traffic figures, and scroll to **DNS lookups**: every name it looked up, newest first, and whether it was answered or **Blocked by** which list. *Blocked only* answers "why doesn't this app work?" in seconds.

> ⚠ **Careful.** This is a record of what everyone on your network visits. Turn it on deliberately, tell the people it affects, and keep the retention short. Only admin accounts see the names; switching it off deletes the history.

"Would this be allowed?" (without trying it)

```
pebble firewall test --in lan --src 192.168.10.50 --dst 1.1.1.1 --proto tcp --
    port 443
```

You get *pass* or *block* and the rule responsible. Nothing is sent over the network.



The same test in the web interface (Firewall, Test a flow).

Done when your change is applied, confirmed (if asked) and works.

CHAPTER 7

Site-to-site WireGuard

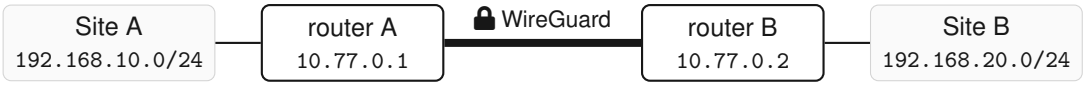
TL;DR

1. `pebble vpn genkey wg0` on each router.
2. Describe the tunnel and the other site in `pebble.yaml`.
3. Add firewall rules for what each side may reach. Apply. Test.

30 minutes, the first time

The idea

Two sites, each with its own LAN, joined by an encrypted tunnel. Traffic is **routed** (each side sees the other's real addresses) and **never NATed**. Nothing crosses the tunnel unless a firewall rule allows it.



Step 1: keys

On each router:

```
pebble vpn genkey wg0
```

This writes a **private key** to `/usr/local/etc/pebble/keys/wg0.key` (readable only by root) and prints the **public key**. Swap public keys between sites. Private keys never leave the box they were made on.

> **Tip.** Prefer the browser? **WireGuard** → **New interface** in the web interface does steps 1 and 2 in one go: it makes the key on the router, shows you the public key to send to the other site, and then you add site B as a peer on the same page.

Step 2: the tunnel (on site A)

```
wireguard:
  interfaces:
    - name: wg0
      address: 10.77.0.1/24
      listen_port: 51820
      private_key_file: /usr/local/etc/pebble/keys/wg0.key
      peers:
```

```
      - name: site-b
        public_key: "PASTE-SITE-B-PUBLIC-KEY-HERE"
        endpoint: site-b.example.net:51820
        persistent_keepalive: 25
        allowed_ips: [10.77.0.2/32, 192.168.20.0/24]
        routes: [192.168.20.0/24]
```

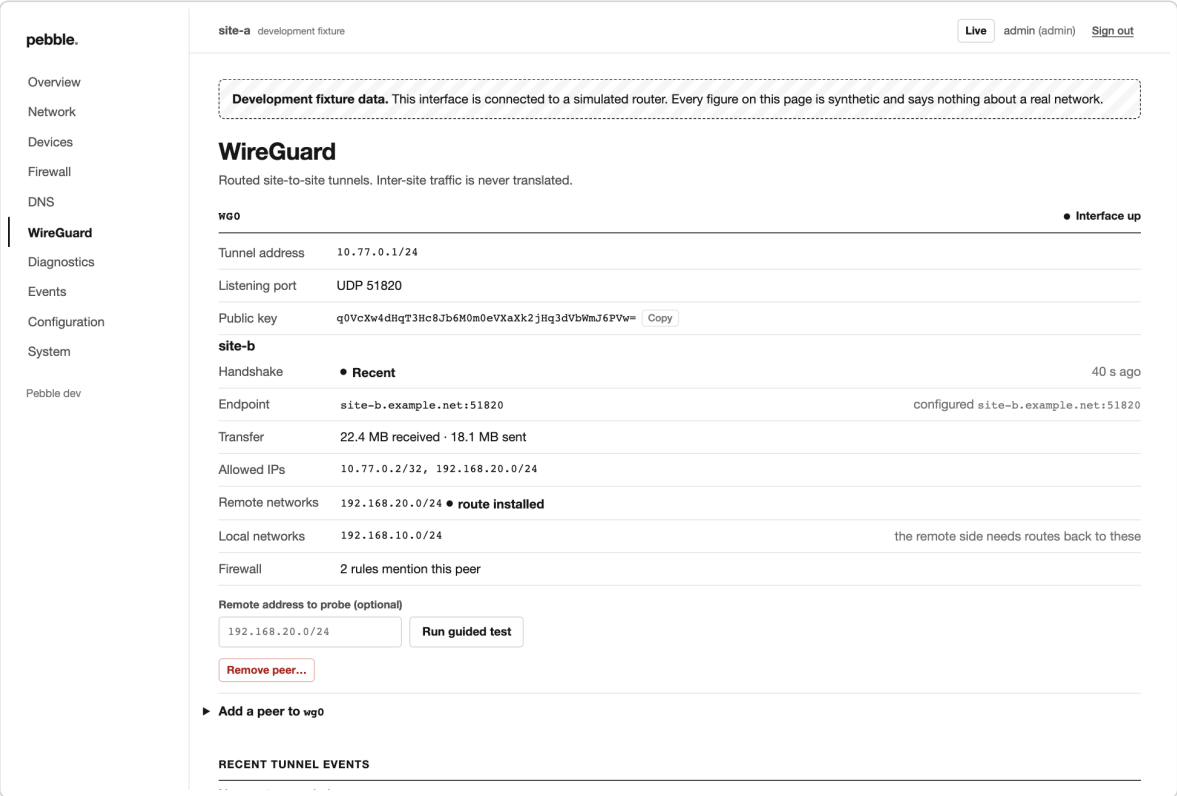
Step 3: say what may cross

```
firewall:
  rules:
    - description: Site A LAN may reach all of site B
      action: pass
      interface: lan
      source: lan
      destination: peer:site-b
    - description: Site B may reach the file server over SMB only
      action: pass
      interface: wg0
      protocol: tcp
      source: peer:site-b
      destination: 192.168.10.5
      ports: ["445"]
```

Step 4: apply and test

```
pebble apply && pebble confirm
pebble vpn status
pebble vpn test site-b
```

`pebble vpn test` walks through the connection step by step: the peer's settings, the tunnel interface, the handshake, the route, the firewall, the traffic counters and a reply from the far side. It tells you **exactly which step failed**.



The WireGuard page: handshake, endpoint, transfer, routes and firewall coverage for each peer.

Tip. Setting up the other end? `pebble vpn show site-b --endpoint your-router.example.net:51820` prints the settings site B needs.

Careful. A fresh handshake means the tunnel is up. It does **not** prove that every remote network is reachable. That's what `pebble vpn test` is for.

Done when `pebble vpn test site-b` passes every step.

CHAPTER 8

Seeing what's going on

TL;DR

- `pebble status`: a quick look.
- `pebble top`: a live dashboard in your terminal.
- `pebble explain outage`: what went wrong, and how sure Pebble is.

Seconds

The quick look

```
pebble status        # one-screen summary
pebble diagnose      # full health check, with evidence and what to do
```

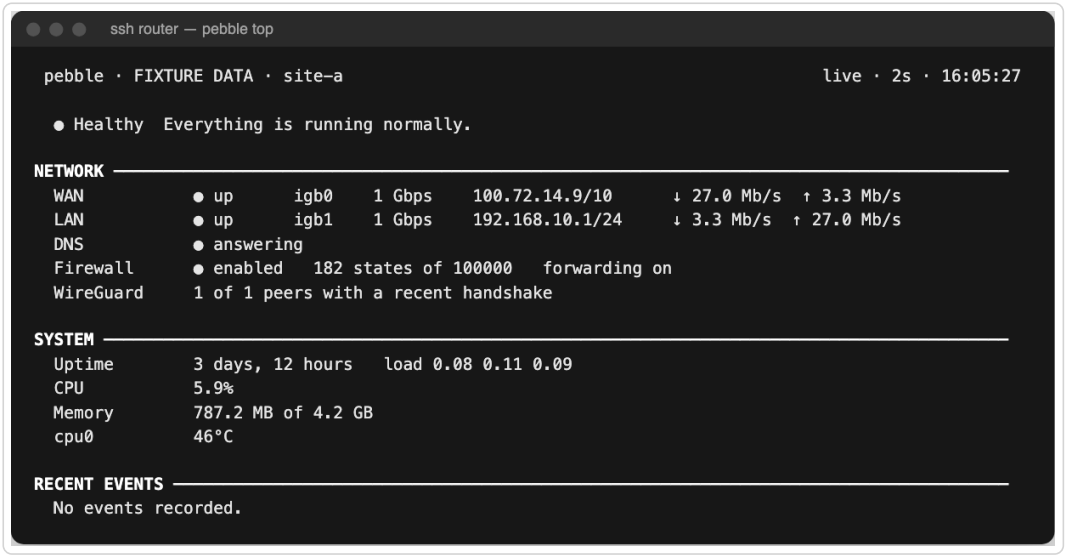
Health is one of four honest words: **healthy**, **degraded**, **unavailable** or **unknown**. Pebble never shows "healthy" when something critical has failed, and never guesses when it doesn't know.

The live dashboard

```
pebble top
```

Key	Does
`1` `2` `3` `4` or `Tab`	Overview, Interfaces, WireGuard, Events
`p` or space	Pause (handy for reading)
`+` / `-`	Slower / faster refresh
↑ ↓ (or `j` `k`)	Scroll
`q`	Quit

Works happily over SSH, needs neither the web interface nor the agent, and has `--ascii` and `--plain` for awkward terminals.



`pebble top` over SSH. (Simulated router; note the FIXTURE DATA label in the header.)

The flight recorder (optional, and brilliant)

Switch it on once:

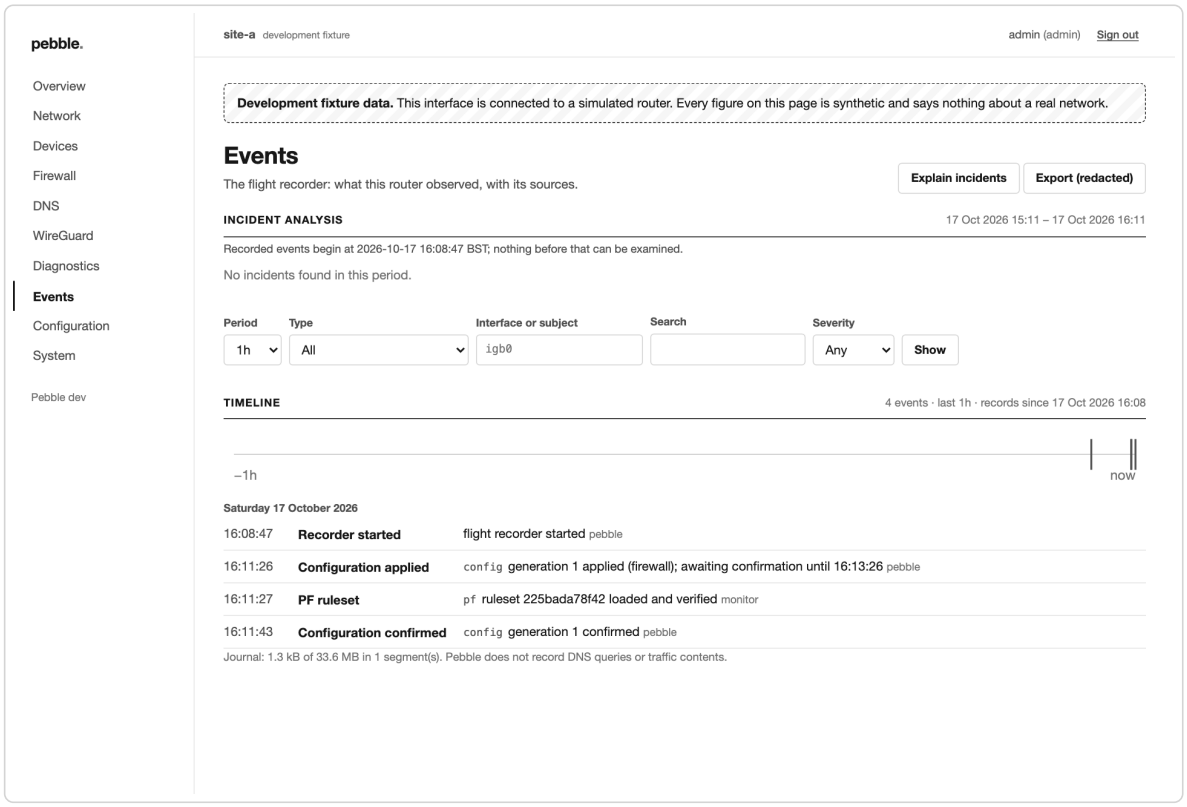
```
monitoring:
  enabled: true
```

From then on Pebble quietly records **events**: cables unplugged, driver hiccups, DHCP renewals, gateway changes, DNS failures, tunnel drops and configuration changes. It does **not** record your traffic or the websites you visit (unless you switch on the DNS lookup history; see the recipes). Storage is capped (32 MB by default).

```
pebble events --last 24h                        # the timeline
pebble events --severity warning --last 7d      # just the worrying bits
pebble explain outage --last 1h                 # the detective
```

`explain outage` sorts the evidence into four piles, so you always know how much to trust it:

Pile	Means
Observed	What actually happened, with times and sources
Likely explanation	What the sequence suggests
Not established	What the evidence *can't* tell you
Suggested next steps	What to try



The Events page: a timeline you can filter, plus *Explain incidents*.

> **Tip.** A classic: "watchdog timeout, then link down, then link up". Pebble will tell you the network card stopped sending, and will be honest that it can't tell whether the driver, the card, the cable or the modem is to blame.

Done when you've run `pebble top` and pressed every number key at least once. (Go on. It's fun.)

CHAPTER 9

The web interface

TL;DR

1. `pebble web user add yourname --role admin`
2. Add `management.web` to `pebble.yaml`, listening on your LAN address.
3. Apply, confirm, and browse to `https://192.168.10.1:8443/`.

10 minutes

It's **optional** and **off by default**. Routing never depends on it: stop it, and the router doesn't notice.

Switch it on

1. Make an account. There's **no default password**, ever.

```
pebble web user add yourname --role admin
```

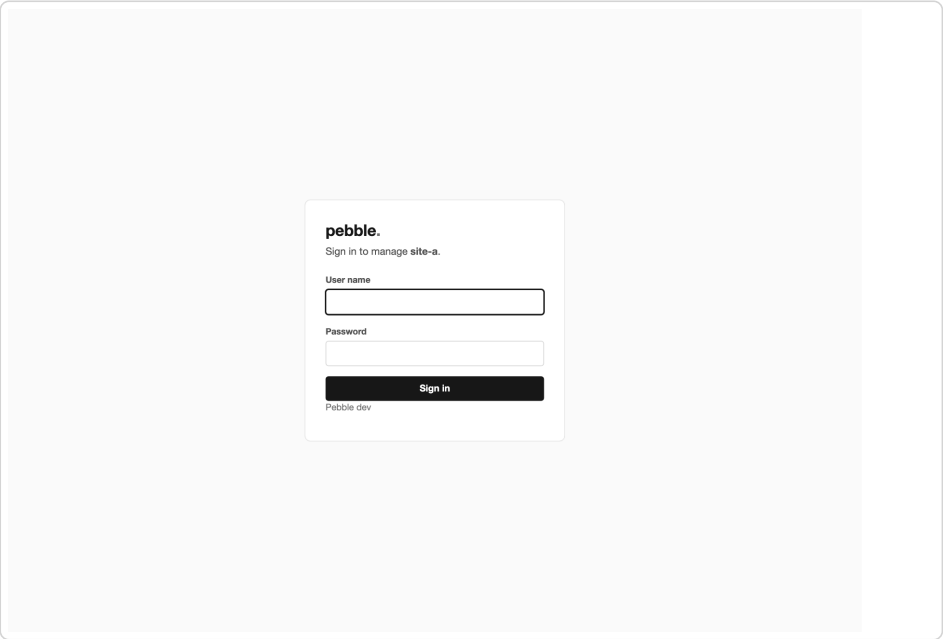
2. Add this under `management:` (next to `ssh:`), using your LAN address and range:

```
management:
  web:
    enabled: true
    listen: 192.168.10.1:8443
    allow_from: [192.168.10.0/24]
```

3. The golden loop:

```
pebble plan
pebble apply
pebble confirm
```

4. Open `https://192.168.10.1:8443/` in a browser.



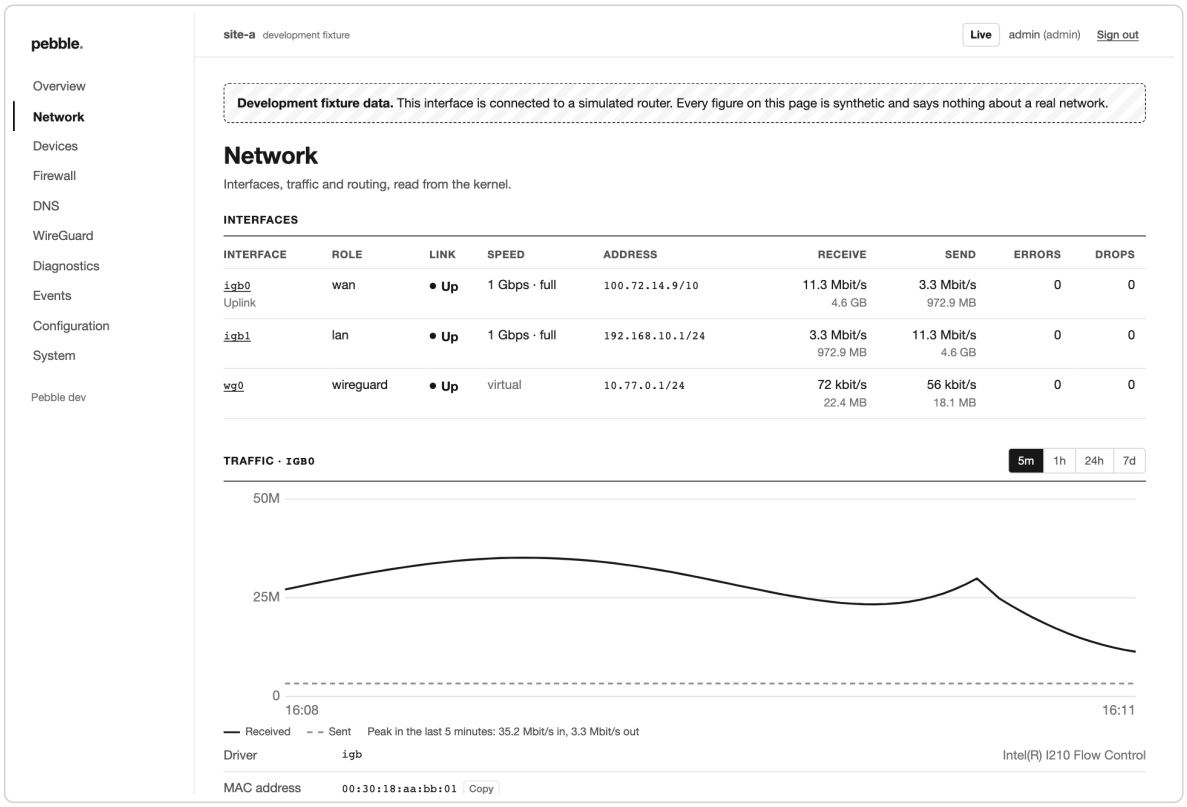
No default password. Ever.

Careful. Your browser will warn about the certificate, because Pebble made its own. Run `pebble web cert` on the router and check that the **SHA-256 fingerprint** it prints matches the one your browser shows. Only then click through.

What's in there

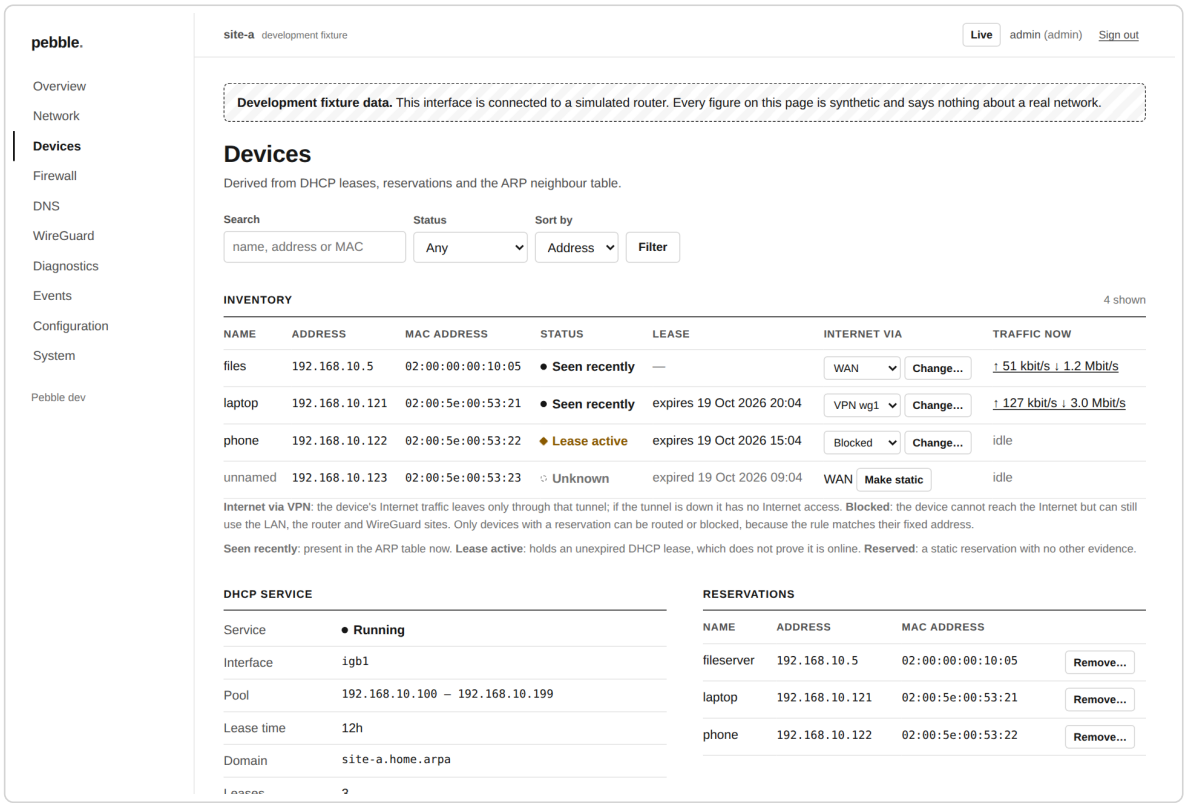
Page	What you get
Overview	Is the Internet OK? Are services OK? Anything odd? Anything to do?
Network	Interfaces, live traffic charts, routing table, "which route does this address use?"
Devices	Everything on your network, live traffic per device, and how each one reaches the Internet (WAN, VPN or blocked)
Map and **Traffic**	How your networks and devices connect, and who's using the bandwidth (observed figures)
Firewall	Rules with match counts, policy summary, flow tester, generated `pf.conf`
DNS	Resolver health, test a name, blocklists, allow and deny lists, local names
Investigations	Watch one device closely for a while (Chapter 10)
Device DNA	What each device usually does, incidents, alerts (Chapter 10)
WireGuard	Tunnels, peers, handshakes, guided connection test; add, upload (a `.conf` file) or remove a tunnel
Diagnostics	Health checks, ping, traceroute
Events	The flight recorder's timeline and incident explanations
Configuration	YAML editor, history, restore an older version
System	Hardware, services, storage, Pebble's own memory use

Network, with live traffic and the route lookup:



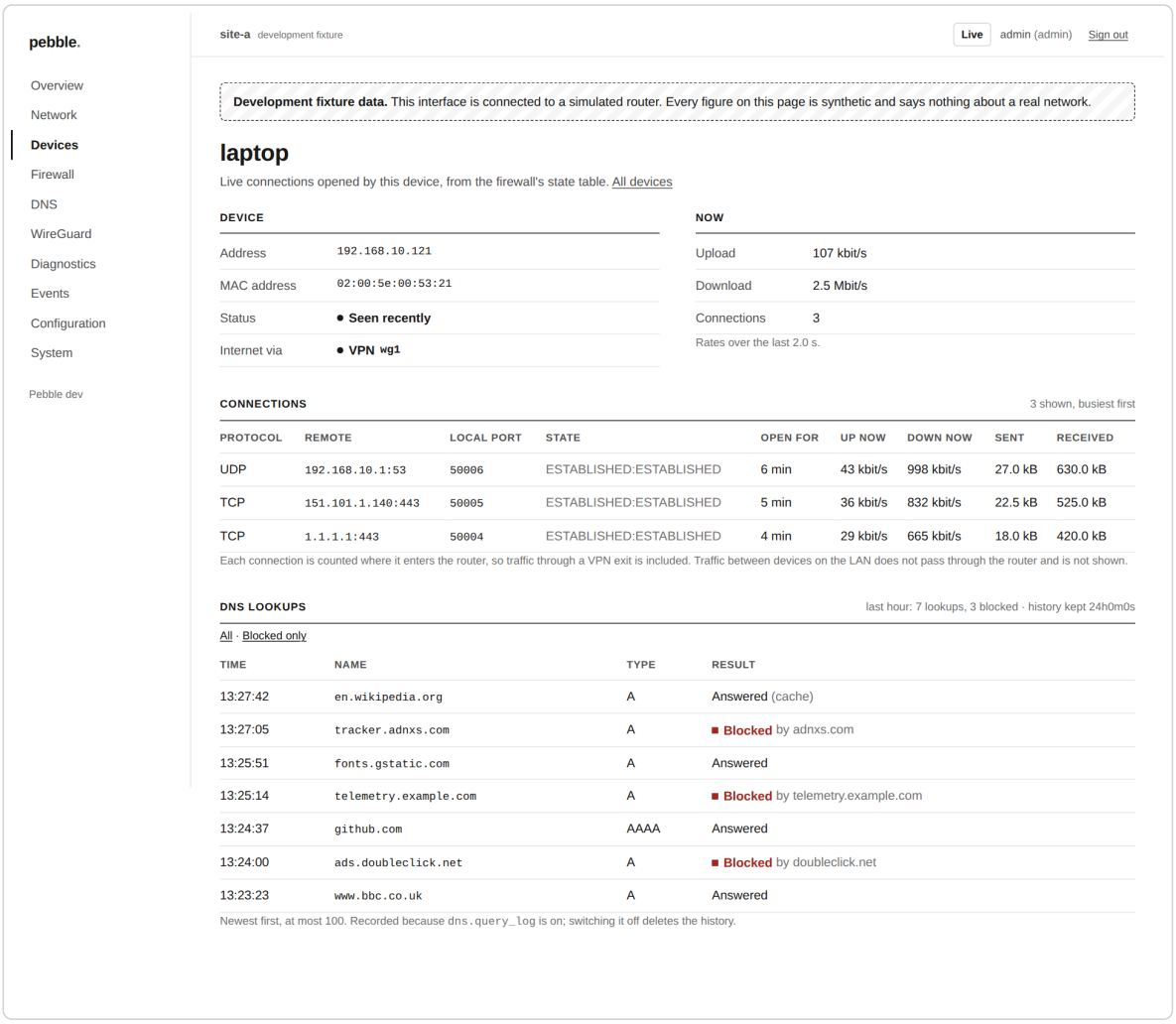
Charts use black and grey with solid and dashed lines, so you never need colour to tell them apart.

Devices, with honest status words (“seen recently” is not the same as “lease active”):



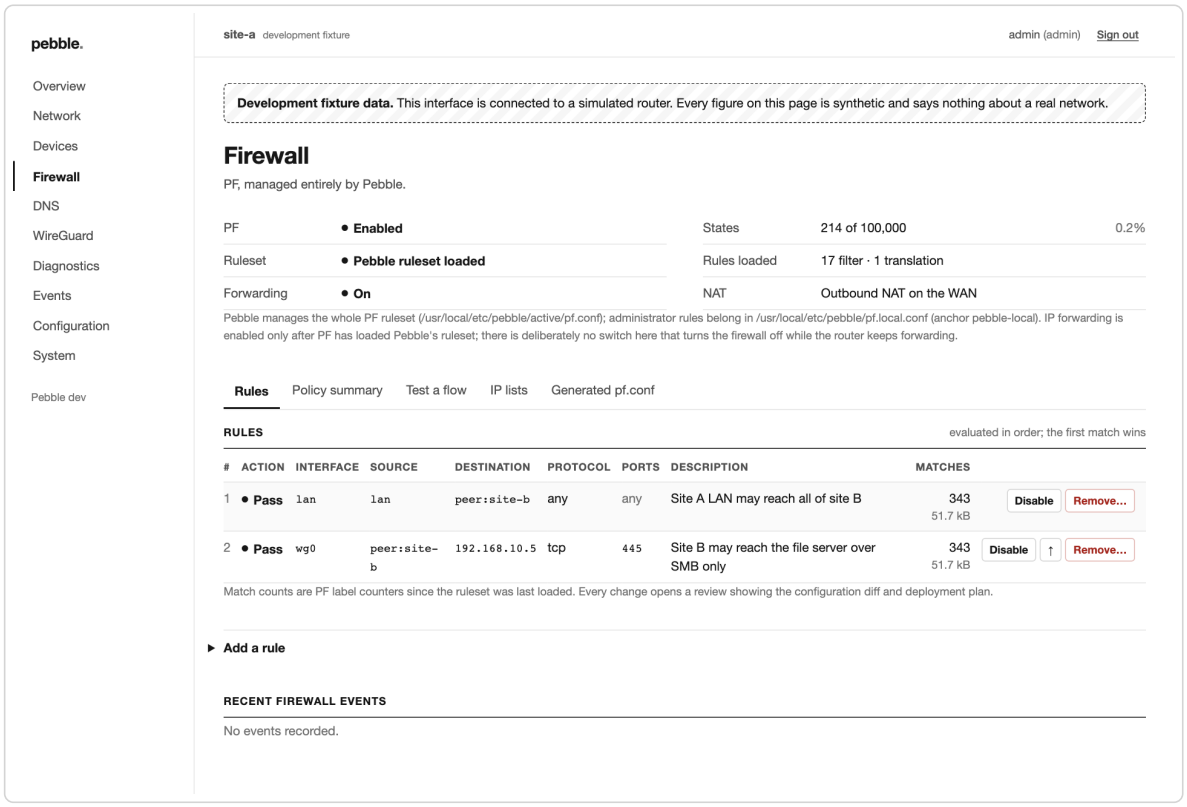
Each device’s live traffic, and how it reaches the Internet. Reserved devices can be switched between WAN, a VPN exit and Blocked; a dynamic one is made static first.

Click a device's traffic figures to watch its connections live:



📷 Every connection the device has open, busiest first, refreshed every two seconds, and (with the DNS history switched on) what it looked up and what was blocked. Connection and lookup detail is for admin accounts.

Firewall, with match counters and lint warnings:



Every rule, its counters, and buttons to disable, move or remove it (each one goes through review).

Configuration, with the YAML editor and history:



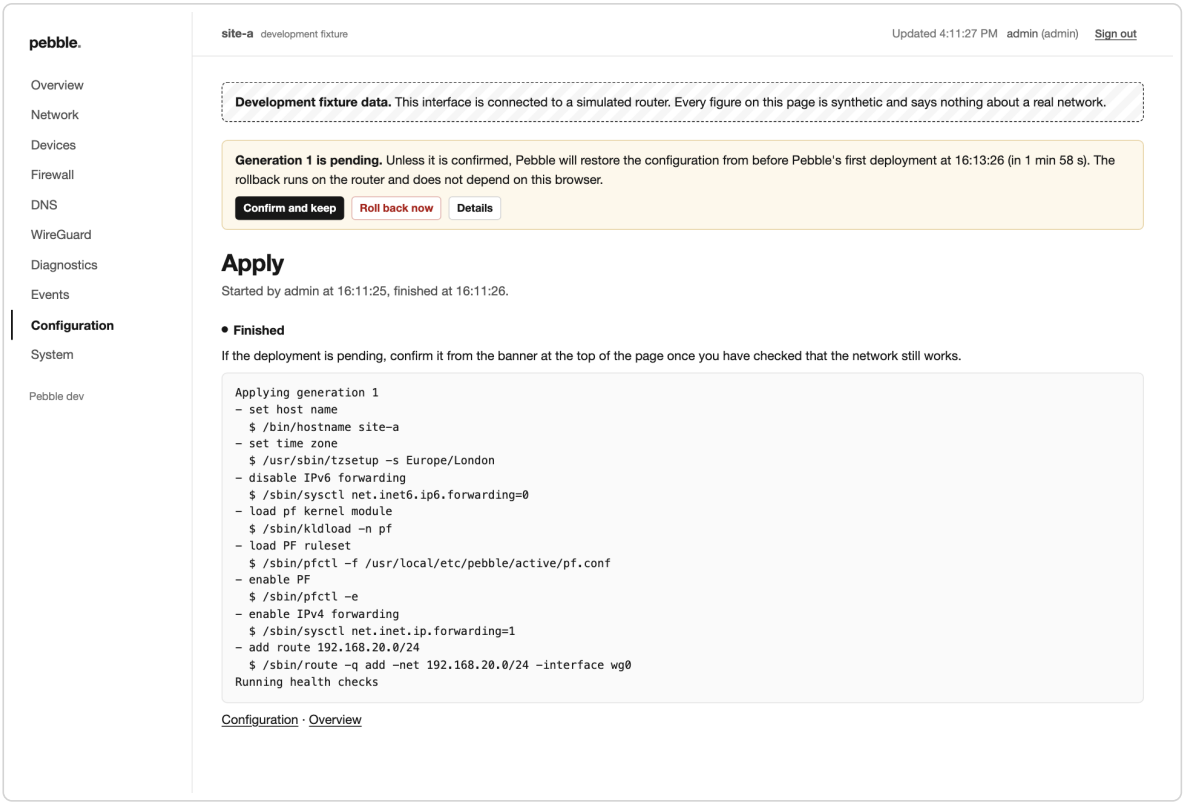
Edit the YAML directly; it still goes through review, plan and protected apply.

Making changes in the browser

Every change, whether from a button or the YAML editor, shows you a **review page** first: the exact lines changing in `pebble.yaml`, every generated file affected, and any warnings. Only then can you press **Apply**. Risky changes show a banner with a countdown and two buttons: **Confirm and keep** or **Roll back now**.

Made a mistake in a form? You're taken straight back to it, with everything you typed still there and the problem written next to the box that caused it. Fix it and press the button again. (There's an **Edit as YAML instead** button too, if you'd rather.)

After you press **Apply**, you watch it happen:



Every step, as it runs.

> **Tip.** The rollback timer runs on the **router**, not in your browser. Close the tab, lose Wi-Fi, drop your laptop in the bath: the router still puts things back if you don't confirm.

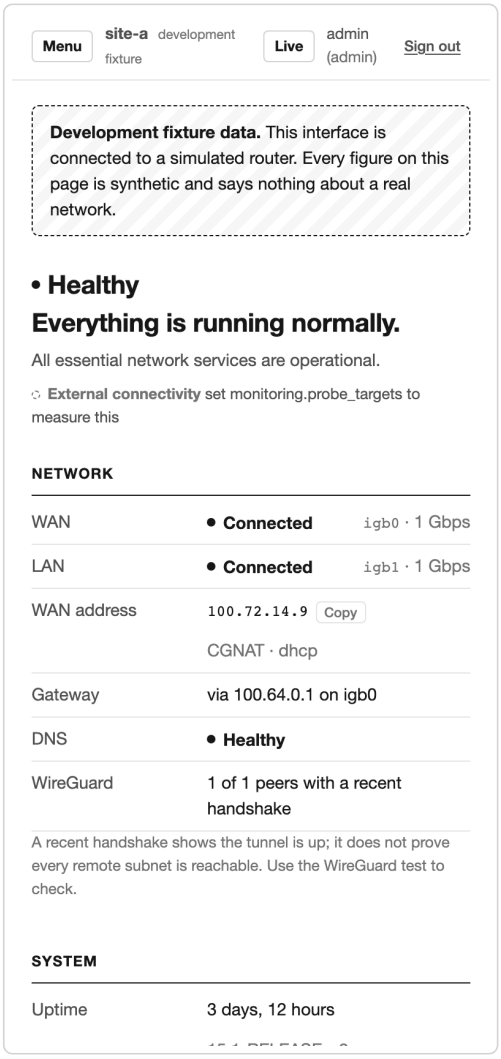
Two kinds of account

Role	Can
`viewer`	Look at everything, test things, preview changes
`admin`	All that, plus apply, confirm and roll back

```
pebble web user add guest --role viewer
pebble web user list
```

```
pebble web user passwd yourname
pebble web user remove guest      # their sessions end at once
```

Tip. It works on your phone too.



The overview on a phone-sized screen.

Done when you can sign in and the Overview says your network is healthy.

CHAPTER 10

Device DNA: noticing when a device changes

TL;DR

- Switch on `device_dna` (and the flight recorder it relies on), then leave it a week to learn.
- It tells you, in plain words, when a device starts behaving unlike itself.
- It **only reports**. It never blocks, isolates or changes anything.

5 minutes to switch on; a week before it's fully awake

Device DNA learns what each device on your network *usually* does: how much it sends and receives, when it's busy, which places it talks to and which ports it uses. When a device does something different (a printer suddenly sending gigabytes at 3 a.m., a smart plug trying every address on your network) it opens an **incident** and tells you what it saw, against what's normal for *that* device.

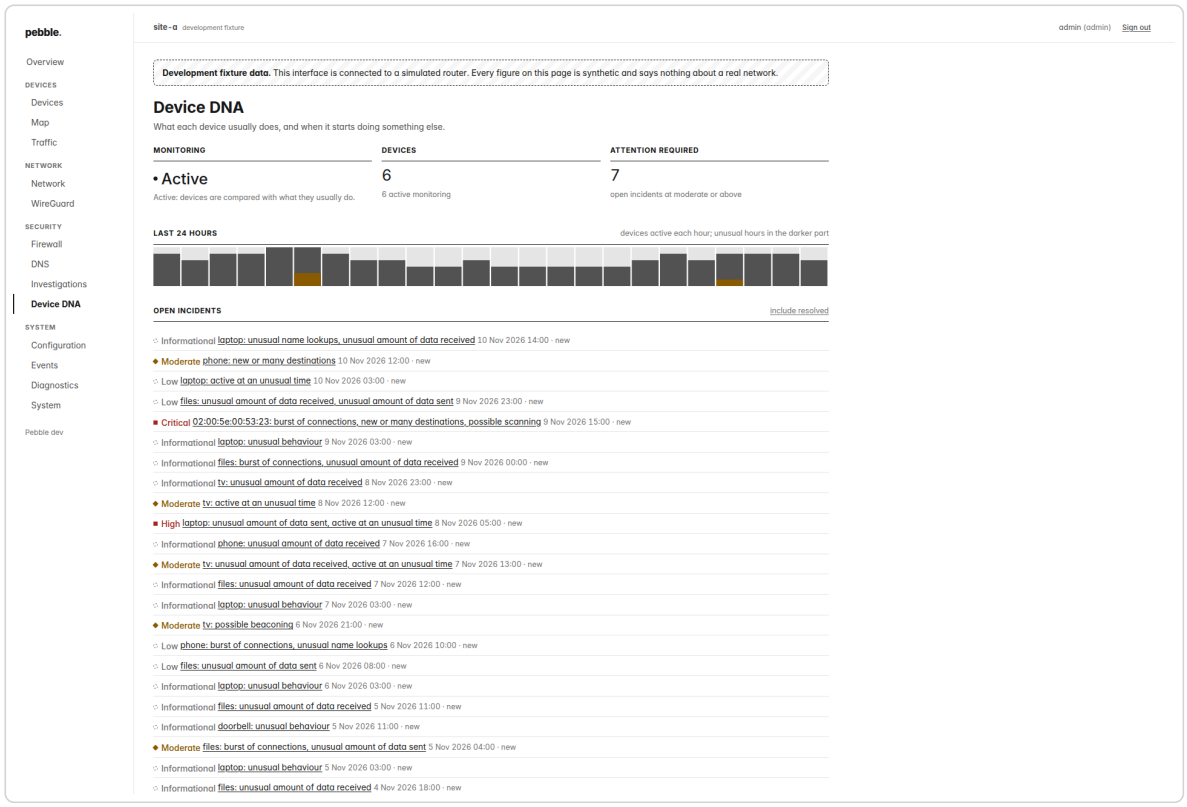
It works from what Pebble already sees once a minute: the firewall's connection table, DNS lookups and DHCP leases. It captures no packets, reads no content, and keeps everything it learns on the router.

Switch it on

```
monitoring:
  enabled: true        # Device DNA measures nothing without it
dns:
  query_log:
    enabled: true      # optional, but it can then see lookups too
    retention: 24h
device_dna:
  enabled: true
```

Then the golden loop: `pebble plan`, `pebble apply`, `pebble confirm`. Check it's running:

```
pebble dna status      # running? how many devices? anything open?
pebble dna devices     # every device, and how far it has learned
```



The Device DNA overview: is it running, how many devices it knows, what needs attention, and the last day hour by hour (unusual hours in amber). (Simulated router.)

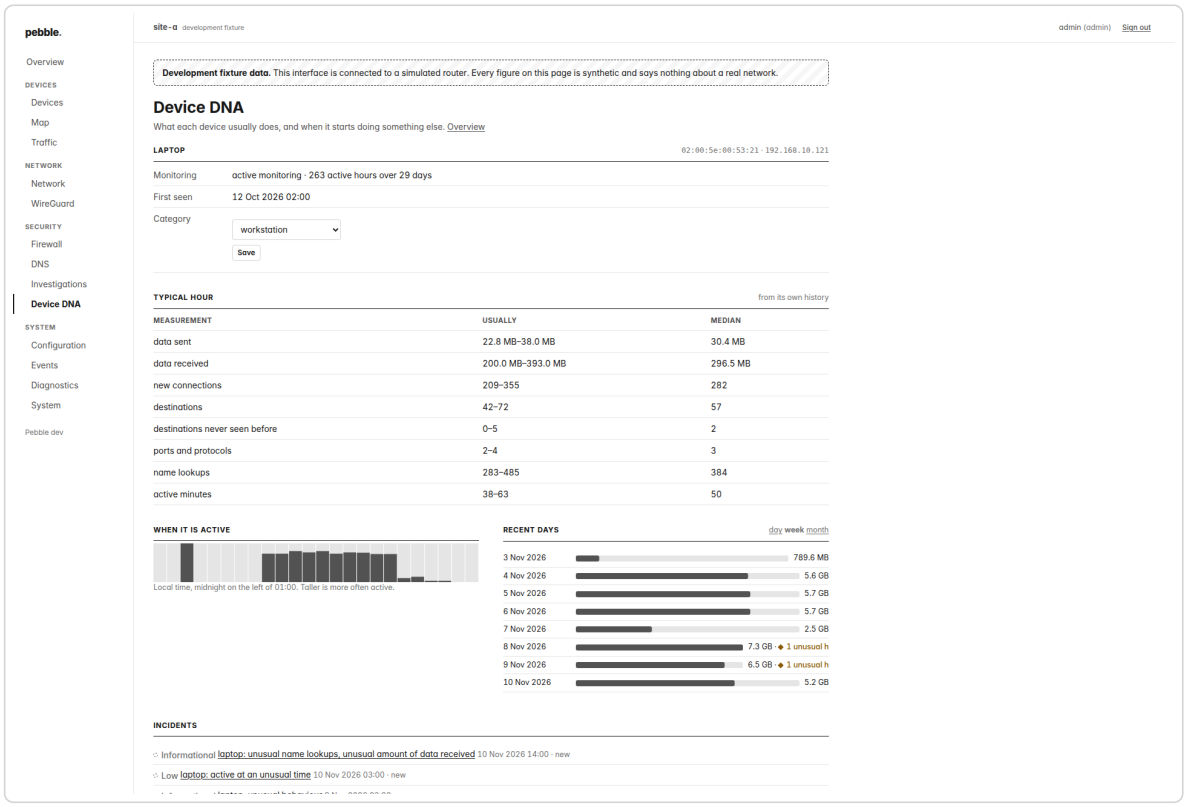
> **Tip.** A device you'd rather it ignored (a lab machine that does strange things on purpose)? Add its name or hardware address to `device_dna.exclude`.

It needs time to learn

Each device goes through these stages on its own clock:

Stage	Means
collecting	First seen less than a day ago. Watching only.
learning	Seen for a day or more, but busy for fewer than 12 hours.
provisional	At least 12 busy hours. It reports, but never above *moderate*.
active	A week or more, and at least 48 busy hours. Fully awake.
insufficient data	Around for a week, but too rarely busy to judge.

Something odd during learning isn't learned: an hour that looks unusual is held back, so a device can't quietly teach Pebble that its bad habit is normal.



One device's DNA: its typical hour, when it is usually active, and its recent days. Every range here comes from that device's own history.

When something looks odd

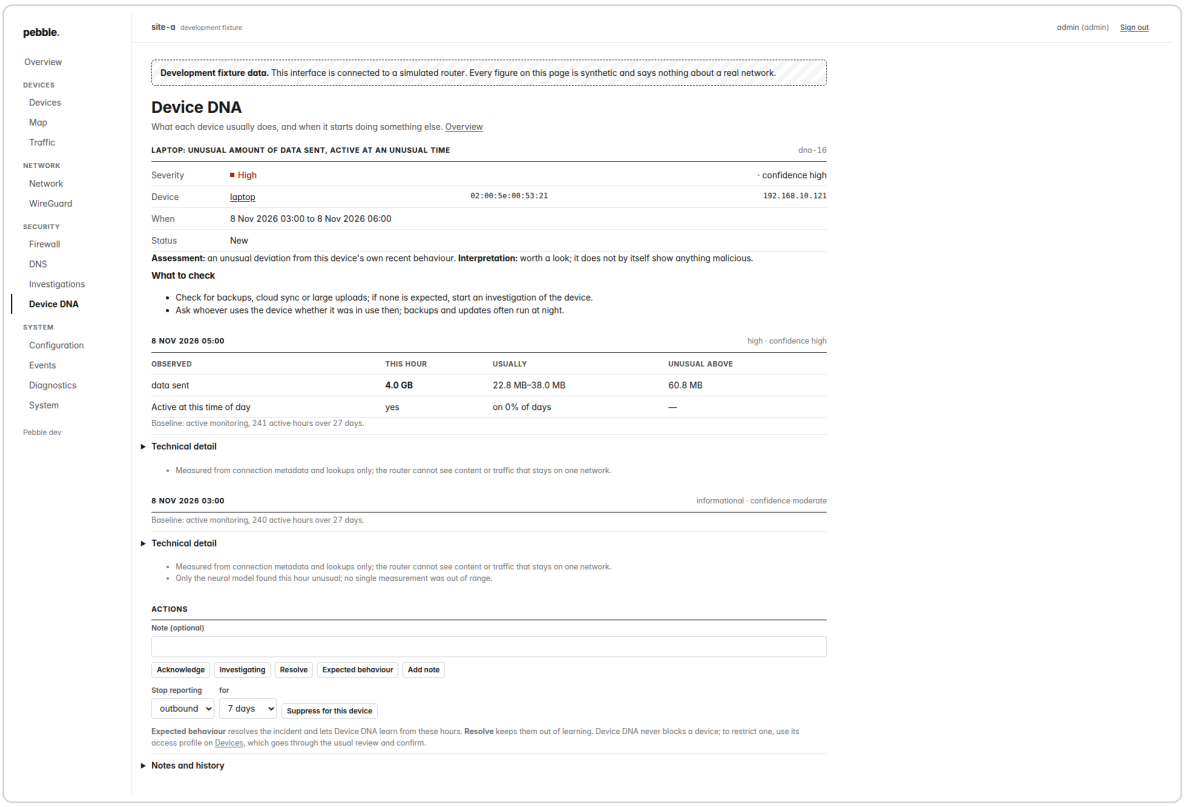
Open **Security** → **Device DNA** in the web interface, or:

```
pebble dna incidents          # what's open
pebble dna show dna-12        # one incident, hour by hour
```

Each incident says what was observed, what's usual for that device, which measurements were out of range, a **severity** (informational, low, moderate, high, critical) and how **confident** Pebble is. Then you decide:

You say	What happens
acknowledge	"I've seen it." Records who.
investigate	Marks it as being looked into.
resolve	Closed. Those hours stay out of learning.
expected	"That's fine, it's the new backup." Closed, and the behaviour becomes normal for that device.
suppress	Stop reporting one kind of thing for this device, for 1 to 90 days.
note	Add a note for future you.

```
pebble dna act dna-12 expected --note "new backup client"
pebble dna act dna-12 suppress --kind beaconing --days 7
```



An incident: what was observed against what's usual, what to check, and the buttons that close it. None of them blocks anything.

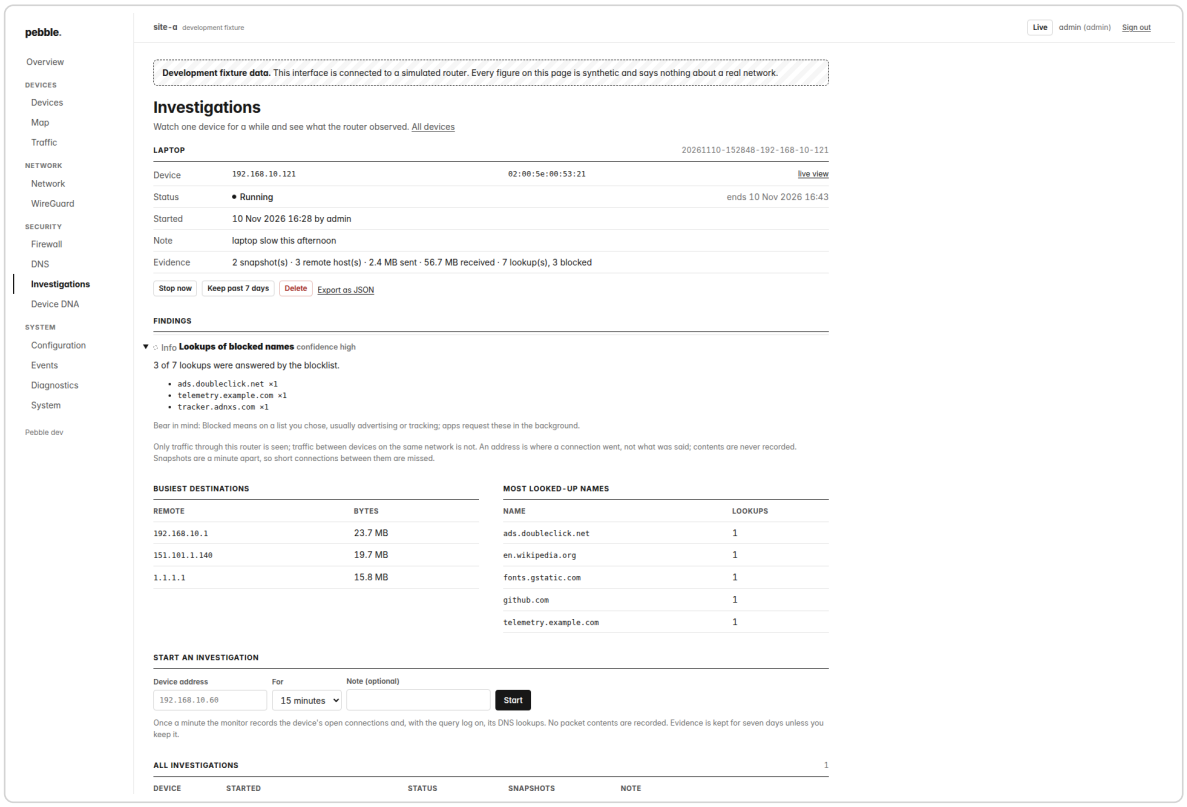
> ⚠ **Careful.** Device DNA is a smoke alarm, not a sprinkler. It **never** blocks, isolates or quarantines a device. If you decide a device shouldn't be on the Internet, that's your call, made the usual way (see *Stop a device reaching the Internet* in Chapter 6), through the golden loop.

Look closer at one device (investigations)

Want to watch one device carefully for a while? Start an investigation. It records the device's connections and lookups once a minute (no packet contents), and keeps them for a week so you can read them calmly.

```
pebble investigate start --for 30m --note "tablet is slow" 192.168.10.60
pebble investigate list
pebble investigate report <id>      # what the Inspector noticed
```

The same is under **Security** → **Investigations**.



An investigation a few minutes in: what the Inspector noticed, the busiest destinations and the most looked-up names. No packet contents, ever.

Get told by email

Put the mail account's password in a file only root can read:

```
install -m 600 /dev/null /usr/local/etc/pebble/keys/smtp.key
vi /usr/local/etc/pebble/keys/smtp.key      # the password, on the first line
```

Then:

```
device_dna:
  alerts:
    min_severity: moderate      # quieter: high
    quiet_hours: 22:00-07:00    # only critical gets through at night
    email:
      host: smtp.example.org
      port: 587
      tls: starttls             # or implicit, on port 465
      username: pebble
      password_file: /usr/local/etc/pebble/keys/smtp.key
      from: pebble@example.org
      to: [you@example.org]
```

Apply and confirm, then:

```
pebble dna test-email
```

Emails always use TLS: a mail server that won't encrypt gets nothing. By default an email says only the severity, the kind of behaviour and the incident number; add `details: true` if you want

device names and addresses in it too. `summary: weekly` (the default) also sends a short report on Monday mornings.

Get told on your phone

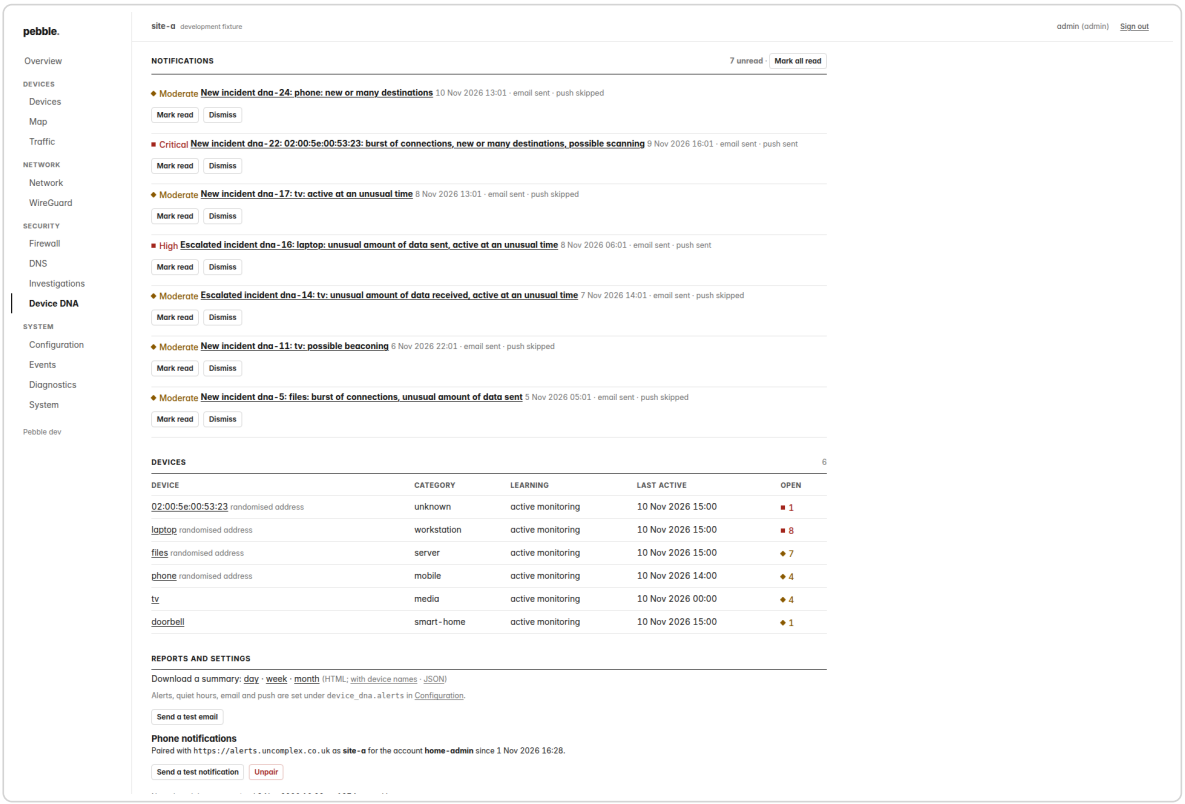
Phone notifications go through a small **relay** service, which passes them on to your phone or browser. Sign in to the relay's web app on your phone, add it to your home screen, choose **Routers** and get a pairing code. Then, on the router:

```
pebble dna pair --name home ABCDE-FGHJK
pebble dna test-push
```

(Or paste the code under **Phone notifications** on the Device DNA page.)

Pairing gives the router its own signing key. The router only ever calls *out* to the relay: no port is opened, and the relay can't reach the router. For each alert the relay learns a random number, the time and the severity. Never an address, a device name or what the device did (unless you set `push.share_category: true`, which adds the kind of behaviour). Acknowledge an alert on your phone and it's marked read on the router within five minutes.

```
pebble dna unpair         # forget the relay
pebble dna rotate-key     # new signing key, same pairing
```



Notifications with each channel's state, the devices and how far each has learned, and the phone pairing at the bottom.

> **Tip.** Not paired yet? Leave `push` out of `pebble.yaml`. Alerts are still listed on the Device DNA page, with an unread count.

Reports

```
pebble dna report --period week                # the whole network
pebble dna report --period month --device 02:00:5e:00:53:10
pebble dna report --period week --html > week.html
```

Devices seen, new devices, incidents, data through the router compared with last time, ports used for the first time, and (honestly) what Device DNA couldn't see.

> **Honesty box.** Device DNA has been tested on a *synthetic* network of 30 simulated devices over four weeks (`pebble dna evaluate` runs it). There it caught about 83–87% of injected problems, with about 1% of normal hours wrongly flagged, and every *high* or *critical* flag was a real problem. Real networks are messier. Expect some *low* incidents that turn out to be nothing, and use **expected** to teach it. Encrypted DNS and traffic between two devices on the same network are invisible to it.

Forget everything

```
pebble dna clear --device 02:00:5e:00:53:10   # one device starts learning again
pebble dna clear --all                        # everything it has learned
```

Switching it off (`enabled: false`) stops all measuring at once; what it learned is deleted after the retention (30 days by default).

Done when `pebble dna status` says it's running, and `pebble dna test-email` (or `test-push`) reached you.

CHAPTER 11

Backups, recovery and upgrades

TL;DR

- `pebble backup` now and then.
- If a change goes wrong, Pebble usually fixes it before you notice.
- Upgrade: new binary → `pebble bootstrap --skip-packages` → `pebble plan`.

2 minutes

Back up

```
pebble backup                    # saved to /var/db/pebble/backups/
pebble backup --include-keys     # also WireGuard keys (keep this file safe!)
```

Copy the file somewhere that isn't the router.

Restore

```
pebble restore pebble-backup.tar.gz --dry-run   # see what it would do
pebble restore pebble-backup.tar.gz             # puts pebble.yaml back
pebble apply && pebble confirm                  # and deploy it
```

Upgrade Pebble

```
pebble update check                       # tells you about updates; installs
    nothing
./pebble-new bootstrap --skip-packages    # installs the new binary and service
    scripts
pebble plan                               # see what the new version would change
```

Tip. `pebble bootstrap` is safe to run as often as you like. It only does what's missing.

Tip. Coming from version 0.2 or earlier? Nothing to convert: the new version reads your `pebble.yaml`, history and settings exactly as they are. Keep the old binary for a day or two (`cp /usr/local/sbin/pebble /usr/local/sbin/pebble.old`); going back is copying it over again and restarting `pebble_agent` and `pebble_web`.

Done when a backup file lives somewhere other than the router.

CHAPTER 12

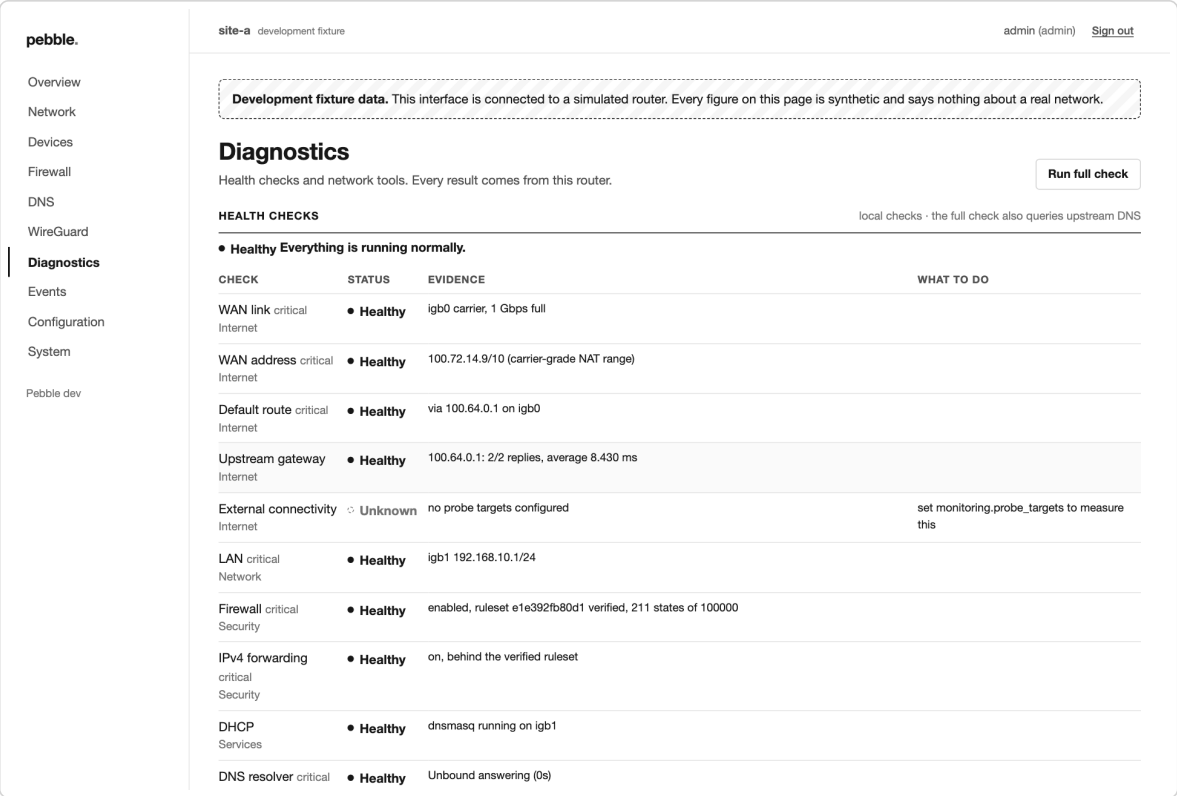
Something's wrong

TL;DR

Breathe. Run `pebble diagnose`. It usually tells you what's wrong *and* what to do. Then find your symptom below.

The Internet is down

1. `pebble diagnose`: read the first thing marked **unavailable**.
2. `pebble interfaces`: is the WAN port **up**, with a link and an address? On PPPoE the address is on `pppoe0`: `service pebble pppstatus`.
3. `pebble explain outage --last 1h`: what happened, in order.
4. Just made a change? `pebble rollback`.



Diagnostics in the web interface: every check, its evidence, and what to do about it.

I've locked myself out over SSH

Wait. If the change is unconfirmed, it rolls back by itself within the rollback window (2 minutes by default). Then reconnect and look at what you changed. If you'd already confirmed it, use the local console and run `pebble rollback`.

Websites don't load, but `ping 1.1.1.1` works

That's DNS.

```
pebble dns test example.com       # what happened to this name, and why
pebble dns status
```

If a site you need is being blocked, add it to `dns.blocking.allowlist`.

A device didn't get an address

```
pebble dhcp status
pebble dhcp leases
```

Check the DHCP range isn't full, and that a reservation isn't pinning the device to a different address.

The tunnel's up but I can't reach the other site

Run `pebble vpn test site-b`. The usual culprits, in order:

1. A missing firewall rule on **one** side.
2. A missing **route back** on the other router.
3. `allowed_ips` that doesn't include the remote LAN.

A device has no Internet through its VPN

1. `pebble vpn status`: the VPN exit needs a recent handshake. None? Check the endpoint, the provider's public key and the private key file, all from the **same** provider file.
2. `pebble firewall test --in lan --src <device> --dst 1.1.1.1 --proto tcp --port 443` should name a `pebble-vpn-` rule. If not, the device isn't on its reserved address yet: reconnect it.

No Internet while the tunnel is down is the kill switch doing its job.

IPv6 doesn't work

1. `pebble diagnose`: **IPv6 default route** missing? The ISP isn't offering IPv6 on your line (some need it switching on). **LAN IPv6 prefix** missing? No prefix was handed out: try `delegation_length: 48` or `64`, whatever your ISP documents.
2. `service pebble dhcp6status`: is dhcpcd running?
3. One device has no IPv6 but the rest do? If it has an `egress` setting, that's on purpose.

Your devices still have IPv4 meanwhile, so nothing else breaks.

"Permission denied", or the web interface won't start

Something has changed who owns Pebble's files (a `chmod -R`, a restore from elsewhere). Let Pebble put them back:

```
pebble fix-permissions --dry-run   # what's wrong
pebble fix-permissions             # put it right
```

It also runs on every apply and every time the Pebble services start, so a restart often fixes it on its own.

"Is my firewall even loaded?"

Run `pebble firewall status`. Pebble only enables forwarding once *its own* ruleset is loaded in PF. If it isn't, the router stops forwarding rather than running without a firewall.

Done when the thing works again. (Then take a backup.)

CHAPTER 13

Cheat sheet

Job	Command
Look before changing	`pebble plan --diff`
Apply	`pebble apply`
Keep it	`pebble confirm`
Undo	`pebble rollback`
Go back to version N	`pebble rollback --generation N`
Check the file only	`pebble validate`
Quick status	`pebble status`
Full health check	`pebble diagnose`
Live dashboard	`pebble top`
What happened?	`pebble events --last 24h`
Why did it break?	`pebble explain outage --last 1h`
Network ports	`pebble interfaces`
Routing table	`pebble routes`
Which route?	`pebble route explain 192.168.20.10`
Firewall counters	`pebble firewall status`
Firewall in plain English	`pebble firewall explain`
Would it pass?	`pebble firewall test --in lan --src ... --dst ... --proto tcp --port 443`
Block a device's Internet	reservation `egress: none` (or Devices → Blocked)
Send a device via a VPN	reservation `egress: wg1` (or Devices → VPN wg1)
Who's using the bandwidth?	web interface → **Devices** → traffic figures
DHCP leases	`pebble dhcp leases`
Test a DNS name	`pebble dns test example.com`
Refresh blocklists	`pebble dns blocklist update`
Refresh IP lists	`pebble firewall iplist update`
VPN peers	`pebble vpn peers`
Test a tunnel	`pebble vpn test site-b`
New WireGuard key	`pebble vpn genkey wg0`
New tunnel, or a VPN provider's `.conf`	web interface → **WireGuard**
Web account	`pebble web user add name --role admin`
Web certificate fingerprint	`pebble web cert`
Device DNA running?	`pebble dna status`
Open incidents	`pebble dna incidents`
That's fine, learn it	`pebble dna act dna-12 expected`
Test email or phone alerts	`pebble dna test-email`, `pebble dna test-push`
Pair with the phone relay	`pebble dna pair CODE`
Weekly report	`pebble dna report --period week`
Watch one device closely	`pebble investigate start --for 30m 192.168.10.60`
Back up	`pebble backup`
Fix file owners and modes	`pebble fix-permissions`
PPPoE session up?	`service pebble pppstatus`
Redial PPPoE	`service pebble pppup`
IPv6 status	`service pebble dhcp6status`
Import from OPNsense	`pebble import opnsense config.xml --map`

Most commands take `--json` for scripts, and `-h` for help.

CHAPTER 14

Glossary (jargon, decoded)

Word	Plain English
WAN	The Internet-facing port
LAN	Your network's port
NAT	Lets all your devices share one Internet address
PPPoE	Logging in to your ISP with a user name and password over Ethernet
Prefix delegation	Your ISP giving the router a block of IPv6 addresses to share out
SLAAC	Devices making up their own IPv6 address from the prefix the router announces
ULA	A private IPv6 range (starting `fd`) that never changes, for use inside your home
PF	FreeBSD's firewall
DHCP	Hands out addresses to devices automatically
Reservation	"This device always gets this address"
DNS	Turns names like `example.com` into addresses
DNSSEC	Checks DNS answers haven't been tampered with
Generation	A numbered, saved version of your router's configuration
Rollback	Going back to an earlier generation
Peer	The router at the other end of a WireGuard tunnel
Handshake	WireGuard's "hello, it's me" between peers
VPN exit	A WireGuard tunnel to a VPN provider that only chosen devices use
Kill switch	If the VPN drops, its devices get no Internet instead of leaking out the normal way
Egress	"Way out": how a device reaches the Internet
Agent	The optional background helper behind the web interface and flight recorder
Flight recorder	Pebble's log of important network events
Device DNA	Pebble learning each device's usual behaviour and noticing changes
Baseline	What's normal for one device, learned from its own history
Incident	One episode of a device behaving unusually, with the evidence
Relay	The service that passes alerts on to your phone; it learns almost nothing
Investigation	A closer look at one device for a fixed time

Open Pebble. Understand the network.
Make a change confidently. Close Pebble.

The router carries on quietly doing its job.