InfraNet Pro
NETWORK DOCUMENTATION

TECHNICAL MANUAL

Document your network the way you see it.

Visual mapping of your network infrastructure: floor plan, 19″ rack, L1/L2 topology, SNMP discovery, VLAN management and handover dossier — in a single application, installed on your own premises.

30
DEVICE TYPES

v1·v2c·v3
NATIVE SNMP DISCOVERY

16
THEMATIC CHAPTERS

On-premise · self-hosted | **Manual-first** | Documentation check (drift) | Label printing · **Avery / Dymo** | Bilingual IT / EN

Complete technical manual · One volume
Every feature of the app

Version: 2.12.0 · October 2026
Real app colours · A4-printable

Contents

A clear guide to InfraNet Pro's features — written for network engineers, system integrators and MSPs.

PART I · GETTING STARTED

1 Requirements & installation 4

2 InfraNet Pro at a glance 7

3 The interface 9

PART II · DRAWING THE INFRASTRUCTURE

4 Device types 13

5 The floor plan 15

6 The rack 18

7 Cables, connections and wireless 20

8 Networks, VLANs and colours 25

9 Port states and presence 29

PART III · MAKING THE NETWORK TALK

10 SNMP: discovery and synchronisation 32

11 Automatic topology 37

12 The workflow 39

PART IV · MAINTAINING AND HANDING OVER

13 AI assistant (advisory) 41

14 Documentation check (Drift) 44

15 Dashboard (summary view) 47

16 REST API and automation (Ansible) 53

17 Change history 55

18 Export, labels and Handover dossier 57

PART V · INTEGRATIONS

19 DCIM / IPAM import (NetBox) 61

20 Hardware catalog, PDU and combo ports 66

PART VI · THE FLOOR ABOVE THE PROJECT

21 Sites and links (multi-site) 69

PART VII · APPENDIX

22 Bilingual IT / EN 73

01 Requirements & installation

What you need to run InfraNet Pro and how to get it up and running in a few minutes.

InfraNet Pro is an **on-premise** application: it runs entirely on your own server or computer, with no cloud services, no external databases and no mandatory containers.

What you need

Component	Requirement
System	Windows, Linux or macOS — any computer that stays on
Runtime	**Node.js 16 or later** (LTS recommended)
Browser	An up-to-date Chrome, Edge or Firefox
Network	Reachability to the devices via **SNMP** (UDP 161), only for automatic discovery
Disk space	Minimal: projects are lightweight local files

Where to get Node.js

Download it free from **nodejs.org**, choosing the **LTS** version, or use your distribution's package manager. To check it is there: `node -v` should answer `v16` or higher.

No heavy infrastructure

No database, no cloud service, no Docker required. Each project is a local file written atomically with automatic backup copies: a power cut mid-save will not corrupt your work. *If you prefer containers, a ready-made Docker image is available too.*

Installation in 4 steps

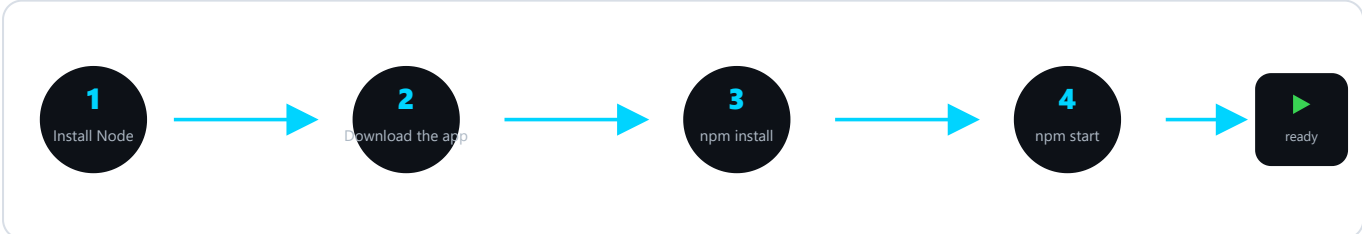


Fig. 1 — From package to first launch: four commands and the app is online.

Option A — with Git (recommended: updating becomes a single command):

```
git clone https://github.com/muttley1973/infranetpro.git
cd infranetpro
```

Option B — download the ZIP (no Git needed): on the GitHub project page press **Code** → **Download ZIP**, extract the folder and open it in a terminal. More immediate, but to update you download the ZIP again.

Then, **in both cases**, install the dependencies once and start:

```
# install dependencies (once only)
npm install

# start the server
npm start

# open your browser at:
http://localhost:8421
```

> **Quick start on Windows**
>
> The folder contains `avvia.bat`: a double-click starts the server without opening a terminal.

Alternatively: run with Docker

If you already use containers — Docker, or a NAS managed with **Portainer** — the project ships a ready-made `Dockerfile` and `docker-compose.yml`: one command builds and starts it, and your data lives in a persistent volume that survives upgrades.

```
# set a fixed session key (once)
cp .env.example .env

# build and start in the background
docker compose up -d --build

# open your browser at:
http://<host-ip>:8421
```

The administrator password generated on first launch appears in the container log (`docker compose logs infranetpro`).

> **Network discovery & access**
>
> The compose file uses **host networking** by default, so the container behaves like a native install and **discovery sees the whole network**. Since it is published on the host's interfaces — login-protected, but still — keep it on a trusted network and use a VPN or reverse proxy for outside access. For an **isolated** instance use `docker-compose.bridge.yml`.

First login and configuration

- **Login.** You sign in with the default administrator account; the first thing to do is **change the password** from the user menu.
- **Configurable port.** The default is `8421`, changeable via the `PORT` environment variable.
- **Access from other PCs.** Point them at the server's IP, e.g. `http://192.168.1.10:8421`.
- **Where data lives.** Projects and floor-plan images are files in the app's projects folder: a backup is a copy of that folder.
- **If a file gets damaged.** The sites, the AI and DCIM settings, the API tokens and the skin list also keep a safety copy (`.bak`) next to the file: if the file can no longer be read, the app uses that copy and notes it in the server log. A file you deleted yourself **stays deleted**, and a broken file never replaces the good copy.

Updating the app

Your data — **projects, users and settings** — lives in files separate from the code, and an update **does not touch it**.

With Git (Option A), two commands in the app folder:

```
git pull
npm install
```

With the ZIP (Option B), download the new version, extract it into a **new folder** and **bring your data across** from the old one:

Keep	What it holds
`projects/`	Your saved networks and floor-plan images
`users.json`	User accounts and passwords
`skins/`	Uploaded panel skins

Then run `npm install` in the new folder and restart. Do **not** copy `node_modules`: it is regenerated.

> **Tip**
> Before updating, copy the app folder: if anything goes wrong you roll back in seconds. The **Git** method stays the most convenient if you update often.

02 InfraNet Pro at a glance

What it is, who it is for, and the principle that holds everything together: *manual-first*.

InfraNet Pro is designed to **draw and keep up to date** the documentation of a network: where devices physically are (the floor plan), how they are mounted in racks, how they are cabled and which VLANs they carry. It is built for people who know the network — engineers, system integrators, MSPs — and need documentation that is always accurate, not a drawing gathering dust in a drawer.

Two views of the same project

🗺 Map (physical)

Where devices are **physically** located: floor plan with rooms, positioned racks, wall sockets, access points. The reality you can touch with your hands.

🕸 Topology (logical)

How devices are **logically connected**: lines between devices derived from LLDP/CDP and switch forwarding tables, with filters for VLAN, trunk and wireless.

These are not two separate drawings: they are the **same graph** read from a physical or logical perspective. You switch between them with the `1` and `2` keys.

The principle: manual-first

> **The network suggests, the human decides**
>
> Data you enter manually is **never overwritten automatically**. SNMP discovery and consistency checking are *advisors*: they propose, flag, highlight — but the racks and cables you draw remain the authority. This is what makes the output trustworthy for a handover.

Many of the app's design choices follow from this: automatic discovery *fills in the gaps* but does not overwrite fields you have filled in; consistency checking shows you the differences but does not delete anything on its own; every alignment with reality is always your explicit decision, with a single click.

> **🔒 The lock: manual-first made visible**
>
> Next to **IP**, **hostname** and a **port's VLAN** there is a small lock. It is not a new feature: it simply makes the protection described above **visible and one-click**. **Lock open** = the field follows the network (the Sync updates it). **Lock closed** = the value is your decision: the Sync leaves it alone and *Verify documentation* warns you if reality diverges. Close it where the value *must* stay fixed (e.g. a static IP, a port's VLAN); leave it open where you want it to follow the network. The **operating status** carries the same lock, but there it tells a different story: nobody measures a device's lifecycle — no device announces over SNMP that it is being decommissioned — so there is no network to defend the field from. A closed lock there means the status is **your own declaration** rather than a value taken from the DCIM you imported from.

IPv6 address. Alongside IPv4, the device panel has an **IPv6** field with the **same padlock** as IPv4. On an SNMP device the **Sync auto-populates** it by reading the device's *own* address (routable IPv6, global/ULA — link-local `fe80::…` is not a management address). Close the padlock to fix it by hand: the Sync leaves it alone and **Verify warns you** if the device reports a different one. A *neighbouring* device's IPv6 can also surface from the deep scan (neighbour discovery).

Model, serial number and firmware. Each of the three inventory fields has the same padlock, but it **closes by itself** as soon as you write the value (and opens again if you empty it). Here the meaning is the opposite of IP and hostname: a closed padlock means **Verify does not contest it**, even if the device says otherwise. It exists for one reason: Verify compares these fields with what the device reads over SNMP, and a model written with the commercial name (instead of the ENTITY-MIB code) was flagged “replaced” on every scan. The Sync does not fill a locked field and “Adopt” does not overwrite it. A **locked serial no longer flags a real swap**: to bring the check back, open the padlock. Values you wrote *before* this version have no padlock: they get one when you rewrite them or click it.

03 The interface

The toolbar, the three work areas and the shortcuts that speed everything up.

The screen has three areas: on the left the element **library** to drag from, in the centre the **floor plan** and **rack** views, on the right the **properties** panel for whatever you select.

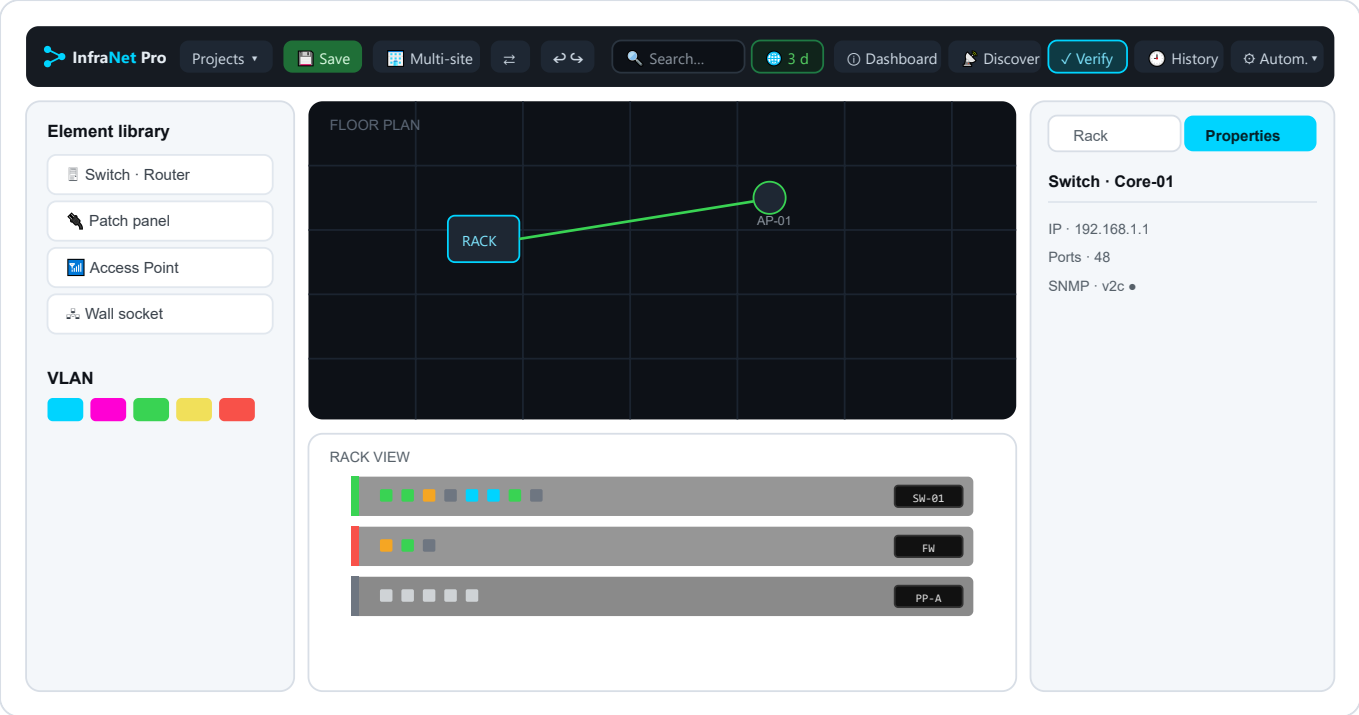


Fig. 2 — The three areas: library (left), dark floor plan + rack on white (centre), properties panel (right). In the toolbar **Verify** is the single primary action, the **freshness chip** (🌐 3 d) next to the search box says how recent the last SNMP read is, and ⓘ **Dashboard** opens the summary view. The old *Sync* button now lives inside **Automation**. **Multi-site** opens the panel for sites, WAN lines and the links between them; the ⇄ icon holds import and export; **History** opens the change log, the Verify history and the restore points.

The status bar

Under the toolbar runs a thin **status bar** in three parts: where you are, what you are looking at, and how the network is doing.

- **Left: the path.** A trail like *InfraNet Pro / Project / Floor plan*. When the open project is the site of an organisation declared in the Multi-site panel, the organisation's name sits between the two.
- **Centre: the active filter and the last auto-link.** When you filter the topology by VLAN, a chip with that VLAN's name appears here (× clears it); after a Sync it also says how many cables auto-link added and how long ago. Most of the time the centre is empty. The recommended next step is no longer pinned to the screen: it lives in the Assistant panel, where advice is asked for rather than imposed (see *Onboarding*).
- **Right: three health numbers.** *Docs* (the share of addressable devices with an IP), *Devices* (the devices that have, or can have, an IP address: wall jacks, patch panels and rooms don't count, and a device with several network cards counts once) and *SNMP* (a dot: green if all respond, amber if some are missing or not polled yet, red only when something is genuinely down).

> **Numbers are computed, never estimated**
>
> The three stats come from the same data you see in the Properties panel: a count, not a forecast. A project just opened and not yet synced shows the SNMP dot **amber**, not red: “not verified yet” is not “broken”.

The Properties panel

The right-hand column holds **everything you know** about the selected element, and it is **contextual**: an appliance gives you IP, model, ports and SNMP; a port gives state, speed and VLAN; a cable gives type, length and label; empty floor gives the background's scale, opacity and grid.

The reason is the heart of InfraNet: objects on the map and in the rack stay **clean and readable** while the **data** is gathered here. Separating the scene from the data is what turns a drawing into documentation: what you type in the panel is your **documentary truth** (*manual-first*).

- **Accordion layout.** Sections (identity, **Network & Access**, the type-specific options) remember whether you left them open or closed.
- **It works with automation.** *Discovery* and *SNMP Sync* fill these same fields and add read-only *live cards*, but they **do not overwrite** what you typed: automation enriches, it doesn't command.
- **It's what the check compares.** *Documentation check* compares exactly these fields against the network; the VLAN filter and colours read them too.

> **Rack or Properties**
>
> Top-right you switch between **Properties** and **Rack** with the `P` and `R` keys: two faces of the same column.

How certainty reads: six signs

In front of any row you ask **one** question: *how much do I trust this?* InfraNet always answers with the same six words — on a cable, on a panel field, on a Discovery row, on a floor-plan tile and in the Dashboard. Learn them once and they hold everywhere.

Sign	What it means	Who has to move
Measured	Read from a device.	Nobody: it is a fact.
Declared	Written by a person. **It does not age**: a decision does not expire.	Nobody.
Derived	Computed by the app from something else: to be confirmed.	You, if you need it certain.
Contradicted	Reality says the opposite of the document.	You: something here needs fixing.
Not declared	Nobody wrote it.	**You**: a declaration of yours is missing.
No reading	No measurement at all.	**A probe**: what is missing is a reading, not your data.

The sign looks at where the value comes from, not at which field it sits in

Brand, model, serial number, firmware, MAC, and a port's description and address are copied by **the Sync** (or the Discovery) when they are empty: as long as you do not touch them, the sign says **Measured**, because a device read them. If you write them, or the device later says something else, the sign goes back to **Declared**. A hand-written value that *matches* what the device says reads as Measured: the device confirms it.

The two absences look identical, and that is deliberate

Not declared and *No reading* carry the same **empty ring**, because both are absences: what tells them apart is the **word**, not one more colour. But telling them apart matters, because they call on two different people — the first asks *you* to write something, the second asks *a probe* to go and look. And neither is a fault: the fault has a name of its own, **Contradicted**.

A high score is not a reading

Wherever a number appears — Discovery's confidence, the strength of an adjacency — it is the **detail** behind the sign next to it, never a second opinion. A device is **Measured** only when something authoritative spoke: it answered SNMP, or a neighbour declared it over LLDP/CDP. Everything else — NetBIOS, SMB, open ports, hostname, MAC, ping — are observations *about* it, not statements *by* it: summed well they make a high number, not a reading. They stay **Derived**.

Six words, four engines — and none of them changed

Behind the signs, separate engines keep working with deliberately different half-lives: a declaration never expires, a measurement does — faster or slower depending on what it measures. What was unified is the **alphabet**, not the arithmetic: the numbers are the ones you had before.

Essential shortcuts

Key	Action	Key	Action
`1` / `2`	Map / Topology view	`Ctrl+S`	Save project
`R` / `P`	Rack / Properties panel	`Ctrl+Z` / `Ctrl+Y`	Undo / Redo (also `Ctrl+Shift+Z`); inside a text field it undoes the typing
`Space` +drag	Pan the map	`Del`	Delete selected element
`right-click`	Start / finish a port connection	`Esc`	Cancel connection / close panel

> **Connecting two ports — use the right mouse button**
>
> Right-click the source port, then the destination one, and the cable is created; `Esc` cancels. For a *cross-rack* connection a banner advises you: navigate to the other rack and right-click the destination port.

> **Tip**
>
> After clicking a port, press `R` to jump to the connected rack without touching the mouse; `Space` +drag is the quickest way to navigate large floor plans.

04 Device types

Over 30 ready-made types, split between rack appliances and floor plan endpoints.

Every element is dragged from the left-hand library. Types come pre-configured with an icon, port count and sensible attributes: a patch panel starts with 24 ports, a switch with 24, a server with a 2U height. Everything remains customisable.

Rack appliances

Type	Default	Type	Default
Switch	1U · 24 ports	Server	2U · 4 ports
Router	1U · 8 ports	Storage / NAS (SAN/RAID)	2U · 4 ports
Firewall	1U · 4 ports	UPS · PDU	power continuity / distribution
Patch panel	2U · 24 ports	KVM · Console server	out-of-band access
WLAN Controller	AP management	PBX / Phone system	telephony
Blank panel · Cable management	structural elements	Media converter	copper ↔ fibre

Floor plan endpoints

Type	Use	Type	Use
Wall socket	termination point (auto-named WA-01…)	PC / Workstation	user workstation
Access Point	Wi-Fi coverage (auto-named AP-01…)	VoIP phone	telephony, often with cascaded PC
Network printer	endpoint with IP	Smartphone / Tablet	mobile endpoint (Wi-Fi / cellular, BYOD/MDM)
Webcam / CCTV	video surveillance (auto-named CAM-01…)	Desktop NAS	Synology/QNAP, SMB/NFS/iSCSI
IoT device	sensors, embedded controllers	Smart TV / Media player	AV / signage
Projector	meeting room	Badge reader	access control
Room/Area	resizable structural element	Custom endpoint	uncovered cases

Auto-naming

Wall sockets, access points and cameras number themselves as you drag them in: no need to rename fifty sockets one by one.

An empty field means "I don't know"

Device-card fields — a projector's lumens, an NVR's channels, a UPS's VA rating — start out **empty**, with a grey suggested value there only to show you the unit and the order of magnitude. Until you type into one, InfraNet treats that fact as **not declared**: it stays out of the handover dossier, out of the reports, and the AI assistant answers "not documented". Dropdowns open on "— not declared —" for the same reason. Clearing a value you had entered really removes the declaration, instead of reverting to the suggestion.

05 The floor plan

Set the physical scene: import the building plan, position devices, create rooms.

The floor plan is your physical canvas: you load the building plan as a background, scale it to the real dimensions and place devices and rooms on top.

Importing and scaling the background

- You load an image (**PNG, JPG, GIF, WebP, SVG**) from the toolbar; PDF is not supported, convert it first.
- You adjust **scale** and **opacity** from the properties panel, then press **Lock scale** so you cannot move it by accident.
- The **grid** aids alignment: things stop on **the line you see** — the step drawn and the step snapped to are the same number. Hide it and placement is **free to the pixel**, rooms included.

Positioning and connecting

Drag devices from the library onto the plan. A **click** selects, a **double-click** opens the full properties; on a wall socket or an AP it takes you to the rack of the connected cable, with the path highlighted.

What can host virtual machines

A *hypervisor* is not required: the **Virtual machines** section is there on **storage arrays and NAS boxes** — rack and desktop — and on **servers** too. Hosting VMs is something a device *does*, not something it *is*: a Synology or a QNAP runs them from a package (Virtual Machine Manager, Virtualization Station) and is still a storage box, with its capacity, RAID level and protocols where they belong. A device **never changes type** to make room for its virtual machines.

Import a VM into its host

When a virtual machine shows up on the network as its own device: in the properties of a host — *hypervisor*, *home-lab*, storage, NAS or server — the whole **Virtual machines** section is the drop area. **Drag the device onto it** and the VM is absorbed into the host, inheriting name, IP and MAC (a MAC *or* an IP is enough, so it works via SNMP from another subnet too) and leaving the undocumented list. It fires **only if you release inside the section**, and it is undoable with `Ctrl+Z` in a single step: the device goes back to where the drag started.

A machine is documented **once**. If that MAC (or that IP, when no MAC contradicts it) is already a VM of a host — this one or another — the absorption is **refused**: the device goes back where it was and a notice says where the other one already is. If you really moved it, delete the other one first. **Discover** recognises it too: the row reads “Already present”, with the VM and its host in the tooltip, and the import does not create a duplicate next to the VM. Another device that took the same IP with a different MAC stays “New”: that is exactly the conflict the duplicate check has to be able to report.

The virtual machine card

In the host panel the VMs are **a list**: state dot, name, role, **resources** and address, with the resources *measured* over SNMP or, failing that, the ones you declared. The row opens the **full VM card**: identity (name, role, guest OS), **state in the icon next to the title** (green running, red stopped, yellow attention, grey not specified) with a small **switch** to start or stop it, **allocated resources**, **network cards** and **handover** data — owner, criticality, backup, notes. All of this you **declare**: what you leave empty stays empty in the documents too. «Network & access» holds the name the machine answers to and the **Management** row, exactly like a PC.

As in the host panel, the header has the **expand all** and **collapse all** arrows, and the card **remembers which sections you closed** even while you type. A closed section summarises what it holds on the right — role and guest OS, resources, hostname, address and MAC, SNMP outcome, owner and criticality — and where you declared nothing it says “empty” in grey, never an invented value.

vNIC ports: when a VM has more than one network card

In the **vNIC ports** accordion you add as many cards as you need — a virtual firewall has the Internet leg, the LAN and maybe a DMZ — declaring name, IP, VLAN, MAC, IPv6 and **port group / vSwitch** for each. The last one cannot be deleted, only emptied: a VM must always have somewhere to write an address.

It pays to fill them all in: **each VLAN** joins the trunk on the host's uplink, **each MAC** makes that leg recognisable to the Verify pass, **each IP** joins the duplicate audit. In the dossier the machine stays a single row.

One thing you will *not* find: the cable. A virtual card plugs into a vSwitch port group and rides the host's physical uplink, already documented. You are not asked which physical port it leaves from either: with two uplinks in teaming that is decided by the balancing policy, and InfraNet never writes down what it cannot verify.

Reading a VM over SNMP

Many VMs expose an **SNMP agent on their own address**: from the *Integration* section you query them with the **very same fields as a device** and the **Read over SNMP** button (leave the host override empty and the VM's IP is used). What comes back — system name, MACs, uptime, cores, RAM, disks — appears in a **«measured» box stamped with date and time**, kept separate from what you declared: it does not overwrite your fields, a dedicated button copies it across.

The **MAC** is what makes the VM «documented» for the Verify pass, and it matters most for machines imported from another subnet, which never carry one. It is written in automatically **only in the case with no ambiguity**: one card measured, one declared. With several cards the query *lists* the MACs but does not assign them — interfaces read over SNMP do not carry their IP — so the pairing is yours. A MAC you already typed is never overwritten.

Two honesty rules, the same as the presence colours: if the VM **answers**, that is recorded as a measurement; if it **does not answer** the outcome is noted in amber and explained, but the VM is *never* declared stopped — it may have no agent, or be filtered.

The network says whether the VM is there: beside your state, never over it

A VM's state — running or stopped — is the one **you declare**. The **Verify** run sets beside it what the network says right now: it asks the IPv4 addresses of the VM's cards and reads the same signals it uses for devices (ARP, the routers' ARP and ND tables, the switches' MAC tables). The card shows one line: **"Seen on the network"**, with when and how (ping, ARP, TCP); or **"No answer"**, in amber, which *does not prove* it is off — a guest firewall or a suspended machine are just as silent; or **"No longer on the local network"**, the only proven absence: on the server's own network the address does not even answer ARP.

When the network **contradicts** what you declared — stopped but it answers, running but gone from the local network — the icon turns **yellow**, a yellow ring surrounds the switch and the host's list shows an amber triangle beside the dot. VMs do *not* fall into the red categories of the Verify: a stopped machine is not an anomaly.

Follow VM state (`Automations` → *Virtual machines*, off by default): the Verify run writes the state itself — running if it answers, stopped only if it has left the local network — but **only where the state is empty or the run wrote it**. A state you set yourself is never touched, the whole batch is a single **Undo** step and it lands in the History. It runs with the Verify (also the one of the "Full" automatic monitoring), not with the Sync alone.

Storeys and rooms: two levels of containment

A *Room/Area* is a resizable coloured rectangle, with an anti-move lock, for offices, server rooms and departments. A *Storey* is the same shape at a larger scale — same drag, same resize, same lock, same colour and opacity — for the level that contains them: «Ground floor», «Floor 1». Its label sits top-left, where the rooms inside it do not cover it.

The stacking is **declared**, not left to chance: the storey sits below the room, the room below the devices. A container covering what it contains is not an aesthetic detail — it makes the contents **unselectable**. A storey is always top-level: a storey inside a storey would need a third level the drawing does not have. Neither storeys nor rooms enter the device inventory: they are the scene, not the contents.

Navigation	How
Pan the view	Click+drag on empty area, or `Space` +drag anywhere
Zoom	Scroll wheel (centred on cursor) or `−` / `+` buttons · range 5%–500%
Deselect / clear VLAN filter	Click on empty area

On a tablet

The map and the rack work with a finger too. **Two fingers** zoom and pan together (the point under your fingers stays put); **one finger** on empty space pans the view; a tap selects, two taps in a row open the properties, a tap on empty space deselects. A device, a room or a rack icon is **grabbed by pressing it for a moment**: it lifts, and only then does it follow the finger — so zooming never moves anything, and a finger that slides first just scrolls the view. The thin bars between the columns can be dragged. **For now with a mouse**: drawing a cable (right-click) and adding a device from the library (drag and drop).

06 The rack

The realistic 19″ cabinet front: units, ports, available capacity, gateway.

Every project can have several racks (6 to 60U, default 42U), with the front drawn realistically, like a photo of the cabinet.

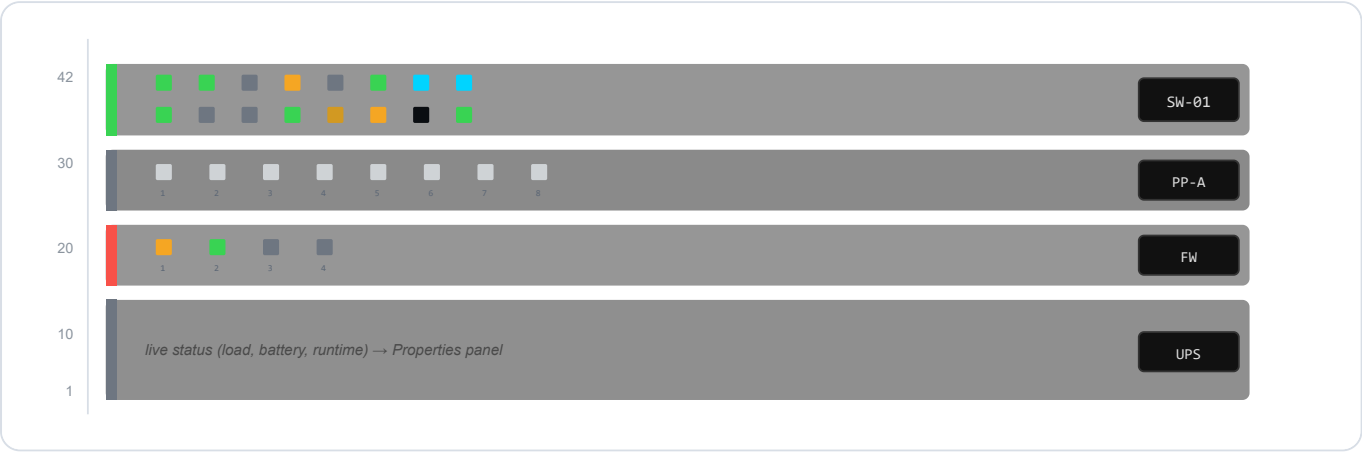


Fig. 3 — Rack front on white viewport. The **vertical stripe** on the left is the SNMP status (green ok · **amber** not polled yet · red error · grey not configured); port LEDs are **small squares** with the **port number** below (green active · red error · grey down or never measured · **amber** no link across three Verifies · **near-black** shut by hand · blue LAG).

Creating a rack and placing it on the floor plan

From the ⋯ menu of the rack panel (the `R` key), **New rack** asks for a name and starts at **42U**. With more than one, the dropdown at the top switches between them: each rack is its own page.

- **Height in units** from **6 to 60U**: shrink it and the app checks that the appliances already mounted still fit, warning you instead of cutting them out.
- **U numbering** from top or bottom: it only changes the **label**, to match the real cabinet.
- **Rename** and **Delete rack** are in the same menu; deleting a rack also removes the appliances it contains.

“Place on floor plan” drops the cabinet on the map as a **marker** snapped to the grid: drag it where it actually stands and a click opens the inside view. That way the map tells you *where* the cabinets are, and cables running **rack to rack** become visible runs on the floor. **“Remove from floor plan”** takes off only the marker.

Populating and configuring

- Drag appliances into the rack — they land in the lowest free unit — or **import a CSV**.
- The **port layout** (count, rows, separate SFP block) is adjusted from the properties panel; the front stays readable from 8 to 48 ports.
- **Apply model** searches a real model and sets **ports, SFP/QSFP and MGMT exactly**, redrawn by the same renderer as native devices. The catalog comes from public (CC0) device-model data; up to **24 SFP per block**.

- **Click** a port to configure it; **right-click** to start a cable to another port, on any rack.

> **Zoom in, and find the whole rack again**
>
> The wheel zooms the rack and the **−** / **+** buttons do it in 10% steps. Zooming in is necessary — on a 48-port switch the numbers are readable no other way — but past a certain point the cabinet is wider than its window and its **sides stay outside it**, taking with them the coloured outline of an absent device, which shrinks to a single line and stops reading as a warning. So the **percentage is a button**: click it and the rack fits back inside the window. And while you are seeing only part of it the percentage turns **amber with a ↔ arrow**, because what is missing is precisely what you cannot see.

Free ports and L3 gateway

🟢 Free ports

Answers "*where do I plug in the new device?*": highlights free ports **in the rack** (green = free, yellow = free on paper but seen active via SNMP) and produces a report exportable as CSV.

🌐 L3 map / Gateway

Promotes the gateway from a mere IP to a **"network → routing device" relationship**: one row per **declared network** — IPv4 and IPv6, VLAN or not — with the device answering that address and the suspicious cases. An **L3** badge marks devices acting as gateways.

07 Cables, connections and wireless

Clear connections at a glance, the cable editor, the complete physical path and Wi-Fi "wave" associations.

A cable is created with the **right mouse button**: right-click the source port, then the destination. The colour follows the VLAN, the **style** says how reliable it is.

The visual language of cables

Appearance	Meaning
Solid coloured line	**Manual** or **discovered** via LLDP/CDP
Animated dashed line	**Inferred** from MAC/ARP/FDB — to be confirmed
Sine wave	**Wireless connection**

Deduced uplink to a “blind” switch

If Sync finds a *documented* switch's MAC learned on another switch's port, it attaches the uplink even when the downstream switch does not answer SNMP: we know *that* it sits behind that port, not *which* of its ports the cable enters. The cable is born **“Inferred · to verify”** — even when the local port is a **LAG member** — and only **one** is attached: confirm it and fix the downstream port and the other members by hand. It is never shown as a confirmed “LAG”, because the downstream aggregation was never observed.

Two parallel cables are not a LAG until something measures it

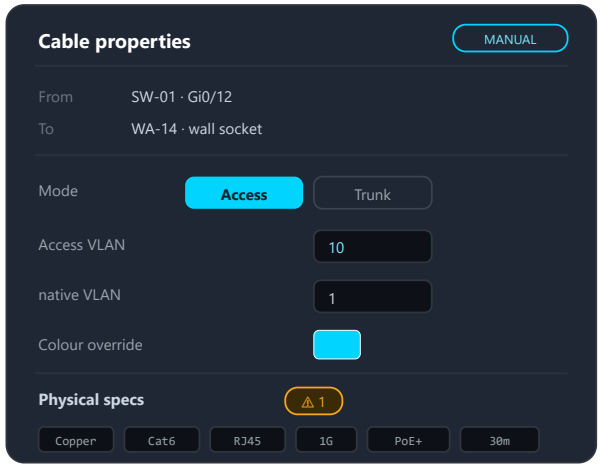
Between two switches, two cables can be a bundle (LACP/EtherChannel) or **spanning-tree redundancy**, where one port forwards and the other is blocked — and a blocked port still has its link up, so no port state separates the two cases. LLDP and CDP say nothing about aggregation. InfraNet therefore shows a LAG only where a device **reports** it (the ifStack / 802.3ad tables the Sync reads) or where you declared it; otherwise the cables stay two, and the coincidence is reported in the auto-link diagnosis line. ⚠ Spanning-tree state — which port is blocked, which bridge is root — is not read: the Overview lists it among the things the app does not judge.

On Cisco, aggregation lives in one table only

IOS declares a `Port-channel` as a *propVirtual* interface rather than an 802.3ad aggregate, and of the tables that speak about LAGs it publishes **one**: `ifStackTable`, where every member port states which aggregate it lives under. InfraNet reads that one — and recognises the aggregator by **name** as well as by type, because on those same switches the VLAN interfaces are *propVirtual* too, so the type alone could not tell them apart. In practice: member ports inherit the **trunk** and its **carried VLANs** from the bundle, which on Cisco is where they are configured rather than on the individual port. ⚠ The **LACP mode** (active/passive) stays your declaration: that value lives in the 802.3ad MIB, which those images do not expose at all.

The cable panel: what you can set

Selecting a cable turns the right-hand panel into its **editor**, and what you write by hand is authoritative (*manual-first*).



- Access or Trunk with one click — the right field appears
- The VLAN you enter here is authoritative (manual-first)
- Override: force a colour on the cable
- Physical specs with validation that explains why
- Double-click the cable → physical path (Fig. 5)

Fig. 4 — The cable editor: endpoints (read-only), **Access/Trunk** mode, VLANs, colour override, physical specs with the ⚠ warning badge.

- **From / To** — the two endpoints, read-only.
- **Status** — the badges that answer *what is this cable*: the mode (**TRUNK** or **ACCESS**), how it came to be (manual, discovered, part of a LAG), over which protocol, and its proof state.
- **VLAN** — from 2.10.1 this is **one section**, collapsible like every other in the panel, holding everything about the VLAN in this order:
  - which VLAN *applies* to this cable, and how we know;
  - the **Access or Trunk** mode — the right field appears depending on the choice;
  - the editable **native VLAN** (PVID): the field carries your *declaration*, and what applies today stays as a grey hint;
  - the **carried VLANs** (`10,20,100-200`), also shown as coloured pills: on a trunk no VLAN wins, so all of them are shown;
  - the **Colour override** — forces a colour on that cable, overriding the VLAN's.
- **Physical specs** — Medium, Category, Connector, Speed, PoE, Length.
- **Wireless connection** — the 📶 toggle turns the cable into a radio association; the physical specs disappear.

The physical cable path

A real connection is often a **chain**: switch, patch panel, wall socket, endpoint. A **double-click on a cable in topology** highlights *the entire path*, not just the clicked segment, and collects the segments in one panel where you select and edit them one by one.

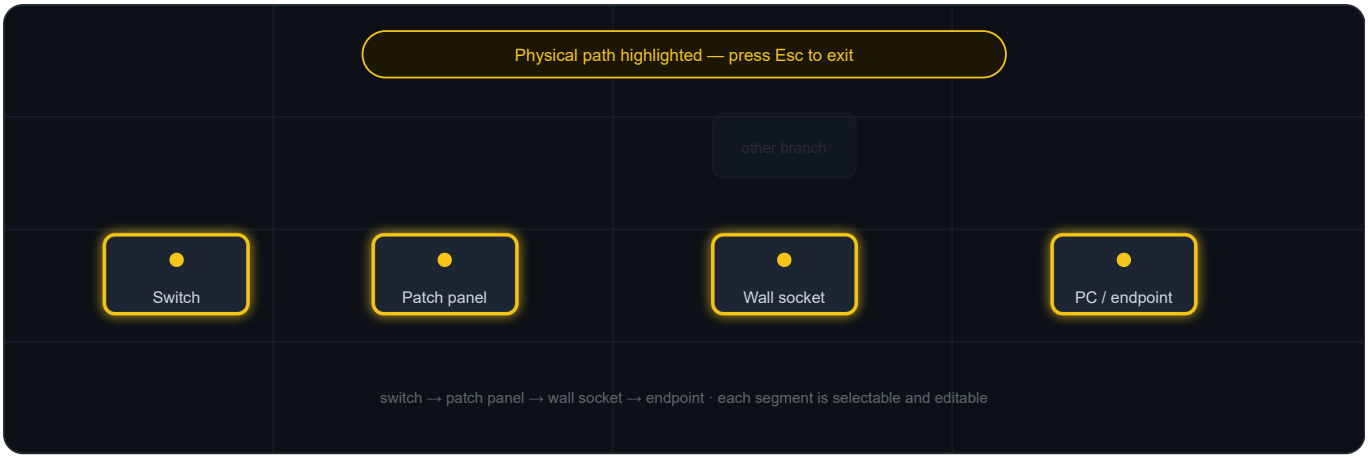


Fig. 5 — **Physical path** highlighting: endpoints in yellow, trace in cyan, unrelated branches dimmed. Press **Esc** to exit.

The socket, the panel and the port description

With the **in-wall run** documented once (socket ↔ panel port, which never changes), the socket no longer has to be maintained by hand: it follows from the switch port. And the devices that never move — printers, access points, phones — act as **anchors**: knowing which socket they sit behind, the rack patch cord that serves them can be derived on its own, even in a rack patched with no rule at all.

> **A convention: put the socket in the port description, not who is plugged in**
>
> A patch panel is passive: no measurement can tell which wall socket a cable comes from. **Who** is plugged in, on the other hand, the app measures on its own — it is the MAC that speaks on that port. So spend the switch port description on the one thing nobody can measure: **the panel port or the socket**, for example `PP1-14` or `PP1-14 | 203`. It is also the most stable fact: it changes only when someone re-patches, which is exactly when they are in the rack with the console open. A description naming a person or a PC goes stale the first time someone moves desk; one naming the socket stays true for years.
>
> ⚠ InfraNet **reads** that description on every read (Check, Sync, monitoring) and, if the port "Description" is empty, **writes it there by itself** in the document: the device is never touched. A description you typed is never overwritten, not even when the device says something else; one that is still a copy of what the device said follows the device when it changes; and a device that merely repeats the interface name does not count as having a description. The "Port descriptions" report — it opens from the button at the foot of the "Cables" row of the Dashboard — counts the ports that write it in the "jack | text" form (you choose the separator, in the report) and says whether the jack it names **exists** in the project: as a wall jack, or as a patch-panel port named "panel-number". It does not check that it is the *right* jack for that port, and a difference **does not appear as a discrepancy in the Check**: this is a working convention, not an automatic check. If the project documents no jacks at all, the report says so instead of calling every jack "unknown". In the report a description equal to the one read from the device shows as "measured", a different one as "declared".
>
> Mind the device limits: some (Cisco, Arista) cut the text at 64 characters, and the cut text is what gets adopted; Aruba CX refuses a longer description altogether. That is why the jack goes **before** the separator: it is the part that always arrives.

Smart validation (warnings, not blocks)

The app checks that medium, category, connector, speed, PoE and length are consistent, and **explains why** when something does not add up:

- 10G on Cat5e → Cat6A required; on **Cat6** 802.3an/TSB-155 allows 10G **up to 55 m**, so within that length there is no warning;
- copper run over 100 m → outside TIA-568, use fibre;
- **fibre beyond the reach of its optical class** at the declared speed: on multimode the distance *drops* as the bitrate rises — OM3 carries 300 m at 10G but 100 m at 40G. Class, speed and length must all be declared, or the app says nothing;
- PoE on fibre, and medium↔connector inconsistencies (RJ45 on fibre).

These remain **warnings**: documenting stays free, but you know immediately what and why.

Wireless: the radio port

> **"Wave" associations**
>
> Wi-Fi devices expose an **antenna badge** 📶: the *radio port*, a virtual connector that hosts many clients without occupying physical ports. Drag a connection from the client to the badge and it is created as wireless. An AP broadcasting several SSIDs on different VLANs lands each client on the VLAN of its SSID, and its wired uplink becomes a trunk carrying all of them.

> **AP mode — capability, not role**
>
> SSID fields appear only on devices that *are* access points by type. A PC used as a **hotspot** can turn on **"AP mode"** in *Network & Access* and broadcast an SSID **without changing its type**: it stays a PC, with the SSID editor unlocked. Opt-in and reversible — turning it off removes any orphaned BSS.

The cable's proof-state

After a **Documentation check** every cable inherits the *reachability* of its endpoints, and the panel says it in the single notation (ch. 3). The **Status** row now holds two separate things: *what* the link is (Access, Trunk, LAG member) and, next to it, *how far it is trusted*.

Sign	Where it comes from
Measured	A **strong** adjacency: a neighbour protocol (LLDP/CDP) declared that neighbour.
Derived	A **weak** inference from forwarding tables (FDB/MAC/ARP): plausible, not confirmed.
Declared	A **manual** cable. It stays declared even when the device goes quiet: “it's down” is news about the *node*, not about the cabling.
Contradicted	Reality says otherwise: the port announces a different neighbour, the endpoint changed its IP or hardware identity, or the port is **shut by hand** under a cable declared live — two declarations in conflict.
No reading	The *ghost*: an inferred cable whose evidence has **vanished**. It stays drawn, never deleted — and what is missing is a reading, not a declaration of yours.

Where there is one, a **quiet chip** sits next to the sign with the detail — the protocol and the strength of the adjacency, for instance `LLDP · 92%`. It is that sign's detail, not a second opinion: these used to be two badges that could disagree.

> **The ghost is visible, not hidden**
>
> On the map an inferred cable is **dashed**, one that lost its evidence **faded**, a declared one solid. In the *Dashboard* the "Cables" tile sums up "N ghost · N inferred · N declared". Signs appear only *after* a check: on a network never verified, no ghosts are invented.

> **The same sign in the Dashboard too**
>
> In the Dashboard's "Cables" list every cable carries **the same pill** you see here: same words, same colour, same engine. Up to 2.11.2 that list had a vocabulary of its own — the same cable said "Strong" over there and "Measured" over here.

08 Networks, VLANs and colours

One colour per VLAN: cables colour themselves automatically and the network is readable at a glance.

You give each VLAN a number, a name and a colour: cables colour themselves based on the VLAN they carry, and the network becomes readable at a glance.

Default palette

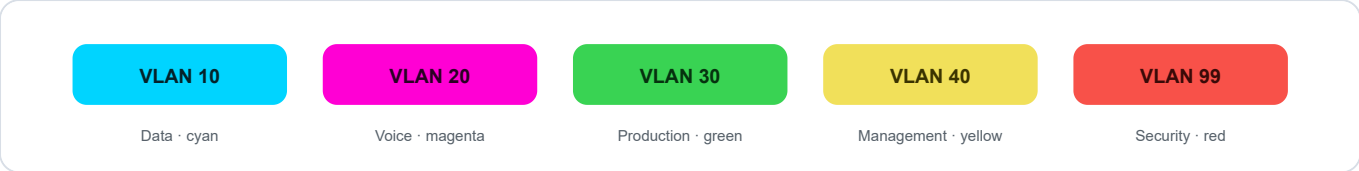


Fig. 6 — The five default VLAN colours. New VLANs receive a distinct colour automatically; every colour can be changed.

Access, Trunk and L3

Mode	Meaning
Access	The port belongs to a single VLAN (untagged traffic): the case of a PC.
Trunk	The port carries several 802.1Q-tagged VLANs, e.g. `10,20,100-200`: the case of inter-switch uplinks.
L3	The port *routes*: it does not switch, so it is in no VLAN at all — not even 1. The case of a point-to-point between two routers, or of an interface with an address of its own.

The app **propagates** VLANs along cables and through pass-through elements (patch panels, wall sockets, media converters), so a run *switch* → *patch* → *socket* → *device* reflects the same trunk. The mode is a property of the active interface (the switchport).

The third one, **L3**, is the one you *declare* when a port does not switch. It matters because otherwise a cable between two routers, in a hand-drawn project, would come out coloured as `VLAN 1`: not a missing answer, but a wrong one. Choosing it, the VLAN field gives way to the **routed network** — optional, picked from the ones you have already declared under “Networks” — and the port table writes `L3` instead of a number. Your word outranks the measurement: if the polled device reports that the port switches, your declaration stands and the panel tells you the two disagree. Choosing it clears that port's VLAN and carried VLANs, because an interface that routes has neither.

The VLAN filter

In topology, clicking a VLAN badge in the legend **isolates** that VLAN: everything else dims or disappears and you see only where it goes. A double-click opens the member list.

An **L3** badge may sit beside the numbers, and it behaves the same way: it isolates the *routed* links, the ones in no VLAN at all — not even 1 — because at that end nothing switches. It appears only when the drawing actually contains one, and hovering it gives you the full explanation. It is an alternative to the others: lighting it turns off the active VLAN, since asking for VLAN 30 and for the links that are in no VLAN at all is not a question. The badge appears both for links the app has *measured* as routed and for those you have *declared* so with L3 mode, so it is useful in a project you have never polled.

Declared networks

The app keeps VLAN and network apart: a **VLAN** is a number, a name and a colour; a **network** is a prefix (e.g. `10.0.10.0/24` or `2001:db8:0:14::/64`), a gateway, a DNS. A VLAN can carry more than one network, and a network need not have a VLAN at all.

- **Dual-stack.** The same VLAN can hold its IPv4 *and* its IPv6, each with its own gateway.
- **Secondary address.** Two prefixes on the same interface: legitimate as long as they do not overlap.
- **Networks with no VLAN.** A point-to-point link between routers, a transit network or out-of-band management have real addresses and no VLAN.

Where you declare them: one place

Networks are written **from the "Networks" section** and nowhere else, because a network has **at most one VLAN** and that field lives on the network.

- **In "Networks" you add, change and delete.** The field at the foot declares new networks; click a row and in the detail pick its VLAN, or "none". The **×** deletes it from the document.
- **The VLAN card shows and takes you there.** It lists the networks that reference it and takes you to their row. It changes nothing, except the *gateway device*: that really is an attribute of the VLAN.

You can type **several networks separated by commas**: they land in one step, with one Undo to take them back. Anything not recognised **stays in the field, in red**: it does not vanish and it is not guessed at.

The "Networks" section: the addressing plan

"Networks" is **the first section** of the project context, because the addressing plan is the document, and it is **a single list**: every declared network, with a VLAN or without, **sorted by address**. The columns are `NETWORK`, `VLAN`, `USAGE`; overlapping pairs are marked in red with the conflict spelled out underneath. Each row opens its own detail; the field for adding one sits at the foot.

> **The bar next to the fraction**
>
> Every row carries a **miniature occupancy bar**, with the same colours and amber threshold as the detail. Six fractions with different denominators do not compare at a glance, six bars do: the question the list answers is "where am I running out of room". On IPv6 there is **no bar**.

> **Why by address and not by VLAN**
>
> Two networks that tread on each other are neighbours in the *address space*: sorting by VLAN hides exactly what you need to see.

The **gateway address** finds its network by **containment**: if it does not fall inside the prefix the app does not guess — it flags it in red and says which network it actually falls in. The **DNS** stays an ordinary field.

Two gateways that are not the same thing

Two related fields appear in the panel, each answering a different question.

- **Gateway address** (in the network detail) is `192.168.20.1`: what clients set as their default gateway. It is *per network* — a dual-stack VLAN has two.
- **Routed by** (on the VLAN card) is *which device* routes it: the box you log into when that VLAN stops passing. It is *per VLAN* — there is one SVI, even with two addresses.

The app tries to connect them by itself: if a documented device holds that address, "Routed by" says so and asks you to confirm. Often it cannot, because the device carries its *management* address while the SVI answers on another: then you pick it by hand, and that choice wins and is never overwritten.

> **Where the warning appears**
>
> "No documented device has this IP" reads **next to the address**, in the network detail: that is where you fix it, not on the VLAN card. The word *documented* is not filler: the warning looks at the project, not at the network — it is not saying that nobody answered. Hovering it shows the only two things that can have happened: either the device is missing from the project, or the address needs fixing.

> **If you import from NetBox**
>
> In NetBox a prefix's VLAN is **optional** and most prefixes have none. They all come in now, and the import preview says how many they are and which VLANs carry more than one network.

Address occupancy (IPAM)

If you've loaded the **DHCP leases** (see *Documentation check*), the card shows the **real address occupancy**: addresses used out of total capacity, a bar that turns **amber** when the subnet is nearly full, and the split between **documented**, **DHCP-only** and **free**.

> **On IPv6 there is no percentage**
>
> A `/64` holds 2^{64} addresses: the app counts the ones it has **seen** and stops there. Two spellings of the same address (`2001:DB8:0:20:0:0:0:1` and `2001:db8:0:20::1`) are *one* address: the comparison is on the canonical form, not on the text.

> **From leases to the map in one click**
>
> When there are addresses *seen by DHCP but not yet documented*, the card shows **"N undocumented → Adopt"**: one click takes those devices (MAC, IP and hostname) into the Adopt window, so they enter the map already documented.

A network is a thing of its own, not a field of the VLAN

Up to 2.8 the subnet was an attribute of the VLAN: one VLAN, one address. From 2.9 the **network is first-class** and the VLAN is a label it may carry — because a VLAN can carry two networks and a great many networks have no VLAN at all. You can add several at once, fix an address in place, delete a row or clear the plan; both destructive actions ask first.

> **Project format.** The file moves to *schema 2* and is migrated in place on open: older projects still open.

The L3 map is therefore one row per network, not per VLAN. **IPv6 can be declared**, not only measured: a bare address normalises to its /64. In the REST inventory networks come out as `prefixes`, with VLAN (`null` when there is none), name, gateway, DNS and provenance.

> **A declared network is recognised by its real shape.** Verify and the Dashboard each read a CIDR their own way, and both IPv4-only: a declared IPv6 network never earned its badge, and "declared if it sits in a /24 or wider" judged a /22 against 24 instead of its own size. One reader answers now, and a row is declared when a declared network contains it and is no narrower.

09 Port states and presence

LEDs, the SNMP stripe and the dimming of devices that have disappeared from the network.

Status is read from colours everywhere: port LEDs in the rack, device borders in the floor plan and in exports — the same colours as the app.

A port either passes traffic or it does not

There are **three** states — active, inactive, error. There is no "idle" any more: the word promised "up but quiet" while devices wrote quite different things into it, and nothing read it to decide anything. What a device actually says about a port carrying no traffic (*testing*, *dormant*) stays a **measurement**: it appears in words in the panel's SNMP bar, in the device's own wording rather than translated, and it expires like every other measurement. The amber in the legend is a different thing — **no link across three Verifies**, an ambiguous symptom (device off? dead NIC? SFP unseated?) that somebody has to go and look at.

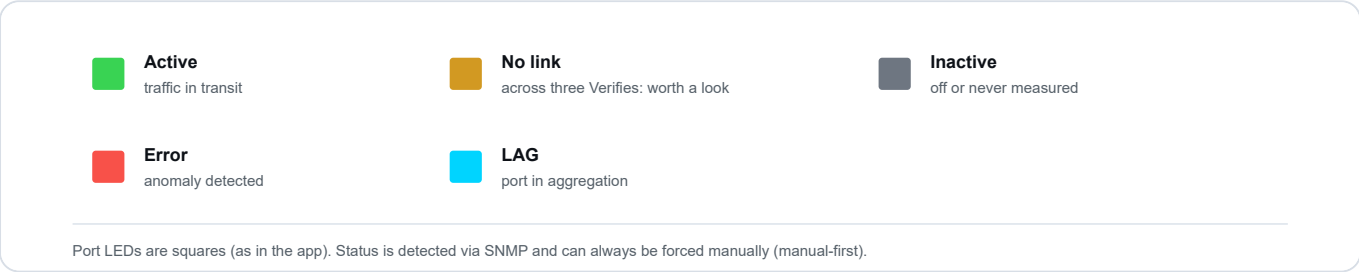


Fig. 7 — Port colour legend, identical in rack, floor plan and export.

On a passive hop, "active" means occupied

A patch panel or a wall outlet never sees traffic go by: there green says **there is a patch cord in it**. The state sets itself with the cable; the *Status* field stays editable so you can state what the app cannot see. Speed and VLAN are not offered: those are decided upstream.

The address belongs to the port, not to the device

A port's card carries an **Interface address**, for the devices that hold **one address per port** — routers, L3 switches, firewalls; the address you *administer* the device on stays on the device card. It also stops you seeing double: a router announcing itself over LLDP with another interface's address looks like a second device.

Not offered on passive ports

A patch panel or a wall outlet has no address: it is copper passing through, not an interface terminating traffic, so the field does not appear. ⚠ What you type is stored **as typed**: if it does not look like an IPv4 address the panel says so but does not refuse it — this is your statement, not a form.

The SNMP stripe ≠ presence

> **Two different signals**
>
> The **vertical stripe** on the left of a device says whether the last SNMP poll succeeded (green) or not (red/grey). That is different from *network presence*: an endpoint that does not speak SNMP is always grey on the SNMP side, even while powered on and working.

> **On the floor plan: orange ring ≠ red halo**
>
> A device that **will not answer the SNMP query** wears an **orange ring**; one the Verify has **proven absent** wears a **red halo** — "it is there but will not answer me" is not "it is gone". The orange has three usual causes (wrong credentials, an ACL excluding the server, a stopped agent) and hovering reminds you of them. ⚠ It does **not** appear on a device the Verify could not reach: there SNMP fails for that very reason.

Absent devices, dimmed on the map

When **Verify documentation** finds **no signal** — no SNMP, ping, ARP or trace in switch tables — the device is **greyed out** on the floor plan and in the rack: still clickable, but at a glance "it is on paper, not on the network". It lights up again when it comes back. Devices that are **Not verifiable**, on a subnet the check could not reach, are **not** greyed: their absence was never confirmed. The same goes for a device whose IP is not written validly (an octet above 255, say): nobody probed it, so it stays **not verified**, never "tried and absent". The dimming rests on the full multi-signal sweep, so a plain Sync never dims anything.

> **And now the dimming has a word**
>
> A coloured halo with no text forces you to ask what it means. Hovering a dimmed device — on the floor plan or in its rack slot — the tooltip says *which* of the situations it is, in the words of ch. 3: **Contradicted** if the probe looked at its subnet and did not find it, or if it answers while you declared it out of service; **Declared** if it does not answer but you expected that, because the silence is the one you declared; **No reading** if the probe never reached that far — which is not news about the device.

Shut by hand, or merely without link?

A switch exposes *two* states per port: whether there is link (`ifOperStatus`) and whether someone turned it off (`ifAdminStatus`). The first is a symptom — device off, dead NIC, SFP pulled; the second is a **decision**, written on the device by a person. On a real switch eight ports read "down" with no way to tell the two someone had shut from the six simply empty. From 2.9.2 the two are separated, and from 2.10.1 they are separated by **colour** too, because they are not the same kind of fact:

- **Near-black** — the port is in `admin shutdown`. A decision, and whoever made it knows: it stays a quiet hole in the chassis.
- **Amber** — no link across three consecutive Verifies: measured but ambiguous (device off? dead NIC? SFP pulled?), with the threshold there so a reboot does not raise an alarm. Somebody has to walk over and look, which is why it asks for attention. It is not the amber of "Ready", which is a declaration: two states cannot share a hue.

- **Light grey** — the port has been down for fewer than three Verifies, or nothing is known. Neither means “shut by hand”.

The same colours reach the **PDF dossier** and the **draw.io** export: paper has no tooltip to ask, and that is where a wrong colour lasts longest. In the port's **Properties** the “SNMP” line says it in words.

> **Your word stays yours.** If you declared the port's status yourself, the LED shows what you chose: the measurement does not overwrite it. Where a declaration and a measurement disagree is the Verify, not a 7-pixel square. And the reading **expires**: if the switch stops answering the port goes back to grey — an assertion this strong must not outlive its evidence.

On cabling: a **cable you declared yourself** running to a shut port reads **Contradicted** (ch. 3) — two declarations in conflict — with a row of its own in the Verify, counted in the Dashboard's «Cables» tile — your word and the word written on the device contradict each other, which usually means the kit was decommissioned and the document has not caught up. An *inferred* cable on that port does **not** become a ghost: through a shut port the link never forms, but that “no link” is the consequence of a decision, not an observation about the cabling.

What counts as a measurement

The amber “no link” is earned over three Verifies, and none of them may be a pretend one:

- **Verifies run while the port was shut** do not count: we are looking at a choice, not at a cable. Reopening it puts the port back to **light grey** and the amber has to be earned again.
- **Readings that never arrived** do not count either: if the device answers “I cannot tell”, or the port does not appear in the answer at all, the previous measurement is **forgotten** instead of standing in for it. A port we know nothing about does not age towards “no link”.

> **If you open a project saved earlier**
>
> The link measurement is a new field: a document saved by an earlier version does not carry it, and until you run a **Sync** no port accumulates. That is deliberate — about those ports we know nothing yet.

10 SNMP: discovery and synchronisation

Letting the network speak for itself — with SNMP v1, v2c and v3, even without credentials on the first pass.

InfraNet Pro speaks **SNMP v1, v2c and v3** natively. Give a device a driver and an address and the app reads interfaces, VLANs, port status, MACs and — where exposed — printer toner, CPU/RAM, UPS and power continuity.

Three actions that look similar but answer different questions

	🔍 Discover	⟳ Sync	✓ Verify
Question	"What is on the network that I have not drawn?"	"How are the devices I drew doing right now?"	"Does my drawing match reality?"
Starts from	a range/subnet	already configured IPs	already configured IPs
Finds new devices?	**Yes**	No (at most adds cables)	No (flags them, does not create them)
When	first mapping / new devices	quick status and topology refresh	audit "is the doc still accurate?"

In the toolbar they are not three equal buttons

The three *questions* stay distinct, but the toolbar has one primary action: **Verify**. One click re-reads the SNMP data (**it includes Sync**), rebuilds the topology and *then* compares against reality. The **bare Sync** lives in the `Automation` menu; **Discover** keeps its own button. A **freshness chip** next to the search box always says how recent the last read is.

The most common confusion: Discover ≠ Sync

Discover starts from a *range* and looks for new hosts. **Sync** starts from the *IPs you already have*, refreshes them and derives cables from forwarding tables (FDB) and LLDP/CDP: you do not need Discover to get connections.

Discovery: scanning a subnet

1. You give the subnet (e.g. `192.168.1.0/24`) and the SNMP credentials.
2. The app scans IP by IP, **classifies vendor and type** from the device profile and presents the candidates.
3. You pick which ones to import: they enter the rack with basic data already filled in.

What you can enter. A subnet (`192.168.1.0/24`, from `/22` to `/30`: at most 1024 hosts per scan), a range (`192.168.1.1-254`) or a single address. An address written with leading zeros (`010.0.0.5`) is read as `10.0.0.5`, and that is the address the app probes. Multicast, broadcast and `0.0.0.0/8` are not hosts: they are refused, and the message says why.

Scan options. Under the fields sit the **Speed** and a **"Search deeper"** box of four *opt-in* options: they find more at the cost of time, and stay separate because some add noise on the network.

- **Speed** — the probe pace toward *unknown* hosts: **Fast** (parallel), **Balanced** (the default) or **Careful (monitored networks)** — a serialised sweep in **random order** with random pauses, nmap's *polite* profile: documenting the network **does not trip its IDS/IPS**, but is slower. Polling of known devices stays parallel. **Fast and Balanced** also resolve the **NetBIOS names** of Windows PCs; **Careful** does not, to leave no NetBIOS footprint.
- **Recognise devices better** — probes the most common TCP ports (HTTP banner, Google Cast, RTSP) to refine type and vendor.
- **Also find quiet devices** — listens to mDNS/Bonjour and SSDP/UPnP announcements on the local segment: it recognises phones, TVs, printers, smart appliances and IP cameras (via ONVIF) that **ignore ping/SNMP**, with a *measured*, vendor-neutral type and model. Same subnet only.
- **Include hosts that ignore ping** — SNMP-probes *every* address in the range, useful behind firewalls that block ping but let SNMP through. Slower.
- **Follow the links between switches** — follows the announced LLDP/CDP *neighbours*, widening discovery beyond the given range.

When the scan ends the panel switches to the **results phase**: the form disappears, only the **table** of found devices stays, and **"‹ New search"** takes you back to the form with the range already filled in.

SNMP v3 detection works even **without credentials** on the first pass — it recognises the device and says authentication is needed — and handles mixed-version environments in one scan.

v3 credentials and protocols. They go in *Properties* → *Integration*: user, authentication protocol and passphrase, privacy protocol and passphrase, security level. Authentication can be `MD5`, `SHA-1` or the `SHA-2` family (224, 256, 384, 512); privacy `DES`, `AES-128` or `AES-256`. AES-256 is not standardised and comes in **two variants**, *Blumenthal* and *Reeder*, which derive the key differently: a device speaks only one. With MD5, SHA-1 or SHA-224 authentication the wrong one does not raise an error but a **timeout** — a device that looks switched off — while with SHA-256 or longer the two coincide. A protocol the app does not know is never replaced by another: the read stops with an error that names the field and the values accepted.

Vendor recognition. The manufacturer is resolved from two local, refreshable datasets: **IEEE OUI** (from the MAC) and **IANA PEN** (from `sysObjectID`). Together they cover tens of thousands of vendors, so a new model is recognised *without* updating the app: refresh them with `npm run update-oui` and `npm run update-pen`.

> **The sign next to the percentage: from the signals, not the score**
>
> In the results table the confidence stays a **percentage**, but the **sign** beside it (ch. 3) does not come from there. It reads **Measured** only when something authoritative spoke — the device answered SNMP, or a neighbour declared it over LLDP/CDP — and **Derived** in every other case. The score is a vote that *adds up* heterogeneous signals (NetBIOS, SMB, open ports, hostname, MAC, ping) and gets high with neither SNMP nor LLDP: calling that "measured" would pass a well-summed pile of clues off as a reading. The tooltip always gives you both. And a barely recognised row is **no longer painted red**: not being sure about something is not a fault.

Automatic monitoring (two depths)

From the **Network automations** popover a single switch turns on background monitoring at a chosen **depth**. **Light** refreshes *only* the SNMP data at short intervals (5–30 min) **without** touching the topology — the cheap "is it up right now?" path. **Full** runs a silent **Verify** at hourly-to-daily intervals: it already includes the SNMP refresh and adds the comparison against reality plus history. The two depths do *not* run together — a Verify subsumes the poll — and an in-flight guard prevents overlapping runs. Honest limit: it runs while the browser tab stays **open** — also when the tab is in the background or behind another window. Chrome can, however, freeze or discard a tab left in the background for a long time (its memory saving): the monitoring then stops, and nothing here can prevent it.

A running round is visible. The scheduled round is silent — it opens nothing — but the `Auto` badge in the header shows a spinning icon in place of the countdown for as long as the round lasts (from a few tens of seconds to a few minutes, depending on how many devices there are). While the round reads the devices — the longest part — **Save** is off, and a click on **Verify**, on `Refresh now` or a right-click on **Topology** answers with a notice (*automatic monitoring is reading the network*) instead of doing nothing; the Topology is drawn from the latest data read. A round that has started is not redone: you wait for it to end.

A device that does not answer does not stop the round. Every SNMP read has a deadline, wider than the longest time the program allows that device (with the factory settings, from 70 seconds to about two minutes; it grows if you raise the SNMP timeout). Past that time the read is abandoned: the device shows *in error*, with the reason, and the round goes on with the others. The same goes for the reachability check (ping and TCP ports) of the **Verify**: if it does not answer within a time proportional to the number of addresses, the Verify goes on *without* the multi-signal presence and declares nobody absent just because it could not look.

> **UPS and power continuity units**
>
> UPS and ATS units have no interfaces to map: Sync queries dedicated parameters and fills a "Live status" block — mains/battery feed, percentage and remaining runtime, load, voltages. UPS OIDs are standard (RFC 1628, vendor-neutral).
>
> **For transfer switches no equivalent standard exists**: we use the *APC PowerNet* profile, the only publicly documented one. An ATS from another make answers the session but not those OIDs, and the card **says so** — "Not readable with this profile" — instead of staying blank as though you had never queried it. No values are invented.

Sync finds wireless clients too

Beyond cables, Sync recognises **wireless associations** and draws them as **waves**, not cables. It derives them from two *vendor-neutral* SNMP signals: the **FDB** (a MAC learned on a radio interface) and the **L3 neighbour table** (ARP/ND), the latter universal and covering devices that don't expose the FDB. It works on the **all-in-one** box, whose FDB already holds the wireless clients — *provided it exposes the radio as an interface of its own*, see the note below. The client gets a station radio attached to the AP's radio; the **BSS/SSID** is chosen by *VLAN match*, and when ambiguous you pick it. The L3 signal fires only for devices that **broadcast an SSID**, so a station is never mistaken for an AP. As always *manual-first*: a hand-drawn wave is never touched, and the client must already be a node — Sync *links*, it does not invent the device.

Honest note. A device exposing SNMP on the LAN is required: typical **ISP all-in-one** boxes are out of reach. Works on enterprise AP/WLC, MikroTik, UniFi, OpenWrt and PC hotspots.

Autosave

Still from the **Automations** popover, **Autosave** (opt-in, off by default) persists fresh data by itself after any real change — an edit, a Sync, a Verify — a few seconds after activity settles; just browsing never saves. Periodic verification is not a separate switch: it lives in **Automatic monitoring** at depth *Full*. Each Verify appends a history row and can keep restorable snapshots — see ch. 17.

Two things saving now tells you

If you **edit while a save is in flight**, the Save button stays in its «unsaved» state: that edit could not have been inside a request that had already left, and switching the signal off would have said otherwise. With autosave enabled the difference mattered even more — the last edit of the day could stay in memory and never reach the disk.

And if **another session has written in the meantime**, the save stops instead of going over it: it shows you **who wrote and when**, and offers to overwrite. Before, the last one in won silently, and whoever lost their work was told everything had gone fine. Two people documenting two different racks the same afternoon is not a laboratory scenario; with the *sites*, which are one per installation, any two sessions are enough.

If a project is reopened from its safety copy

Every save keeps the previous version beside the file. If one day the project file cannot be read — it is truncated, or on Windows an antivirus holds it for a moment — InfraNet opens **that copy** instead of telling you the project is gone. It is **older** content, and it now says so when the project opens, *before* you save over it: from that point on the recovered version would be the good one, and nothing would mention the previous one again. The warning tells the two cases apart — the file is *missing*, or it is there and *will not open* — and blocks nothing: it is your call.

Silent to this key

With **SNMPv2c a wrong community raises no error**: the agent silently drops the request and the probe times out. From outside that is indistinguishable from a device with no SNMP at all — and it is the most common way to file an entire estate as "unmanaged" after merely knocking with the wrong key. Discover says it now: the row carries a grey **"SNMP"** badge with a muted speaker (the tooltip says why and what to do), and their count sits at the top of the table, under the **"Silent to this key"** filter, which is what makes someone re-run the scan with the other community.

> **When it appears, and when it does not**
>
> Only where something **authoritative** licensed the expectation of an answer: a neighbouring device **declared it over LLDP/CDP**, or the project already **documents it with an SNMP driver** and the host is alive. It never appears on a PC or a printer without SNMP: those are correct as they are, and a badge on every quiet host would be noise — and noise on a signal erases it. A documented device that is *switched off* stays absent, not silent: those send a person to two different places. ⚠ A neighbour announced by a switch that does not answer our key **used not to appear at all** — not a wrong row, an absence, which reads as "not there". It now shows, with who announced it, from which port and under what name.

When the VLAN column stays empty

After a Sync you may find the **VLAN empty** on ports that clearly have one. The reading did not fail: the reading is saying *"this device did not declare it to me"*. The standard exposes it in `dot1qPvid`, but not every switch publishes it — on some it stays at 1 whatever the port's real VLAN is. Where one exists, the app also tries the vendor's own path; where neither answers, **the field stays absent**.

That choice is deliberate, and it changes what you see: this case used to come out as `1`, which is indistinguishable from a *measured* 1. From there on every reader treated it as a value read from the device, and the VLAN you had **declared** — or the one the app had propagated along the cable — could no longer prevail over a measurement that had never existed. An empty cell is information; an invented 1 is damage.

> **A "1" is not always a measurement**
>
> A device that knows **only** VLAN 1 is not switching VLANs: its interface is simply *untagged*, and what it says carries no authority over a VLAN it does not know. One that knows others too, when it says 1, is *choosing* it — and then it counts. That is why a controller or a switch living in VLAN 99 can no longer paint the cable of a network that declares 99 with VLAN 1. ⚠ **What to do about it: declare** that port's VLAN in the cable panel (ch. 8). Your word is not overwritten by the Sync, and if one day the device starts declaring something that does not match, the panel tells you instead of choosing for you.

11 Automatic topology

The L1/L2 cross-check: the app draws discovered neighbours and proposes cables to create.

The Topology view reads the neighbours declared by devices (LLDP/CDP) and forwarding tables, and draws them as lines on the map. It is the **mirror** between the cabling you have documented and what the network reports about itself.

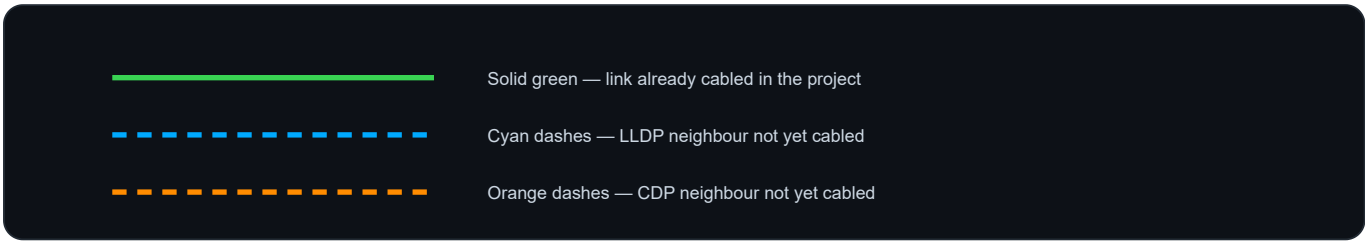


Fig. 8 — Topology lines. Clicking a dashed line opens the detail and the **"Create cable"** command, which matches ports by interface name.

Filters active only here

- **TRUNK + ACCESS** — cycles through three states on each click: *together*, *access only*, *trunk only*. Excluded lines **disappear** rather than fading: a nearly transparent line is still there to read and to reach across with the mouse. As with the medium, the pill carries the number it is hiding. A run carrying both trunk and access stays visible under either filter, because it is true under both; one discovered over LLDP and never documented has no declared mode, so it drops out of both and is counted separately — calling it “access” would be an invention.
- **ENDPOINT** — hides the last segment towards terminals: the view stops at wall sockets and the backbone (useful for reading the structure without the noise of PCs). Terminal means *it has an IP address*, so a VoIP phone counts too, even though it carries the PC downstream. Sockets, patch panels and media converters do not — they have no IP, they are copper, and they stay to mark the edge of the filtered view.
- **WIRED + WLAN** — cycles through three states on each click: *cable and waves together*, *wired only*, *WLAN only*. It is there so you can look at one medium at a time: an access point with twelve wireless clients is unreadable among the forty cables running beneath it. While a filter is on, the pill carries the number of links it is hiding.

> **When the port is unknown**
>
> Two devices that announce each other **are attached**, even when the port name they exchange matches no port in the document. The line is drawn all the same, with a **?** where the port would be: that is the honest shape of what was measured, rather than picking a port at random for the sake of drawing something. Where a cable of yours already joins the two, the pair reads as confirmed and the precise link wins.

Why a neighbour did not become a cable

At the end of a Sync the summary line counts the announced neighbours that produced no link, **split by reason**, because they ask for different things: “device not in the document” is closed by scanning it or adding it; “local port unresolved” means the protocol named a port that matches none of its own.

Create cable from suggestion

Let LLDP find the neighbours, then click "Create cable" on the dashed line: cable the network on the map without drawing every connection by hand.

12 The workflow

Not a straight line, but a *maintenance loop*: discover, document, verify.

You import the floor plan once. Then the *discover → document → verify* cycle runs over time: that is what keeps the documentation alive instead of letting it age.

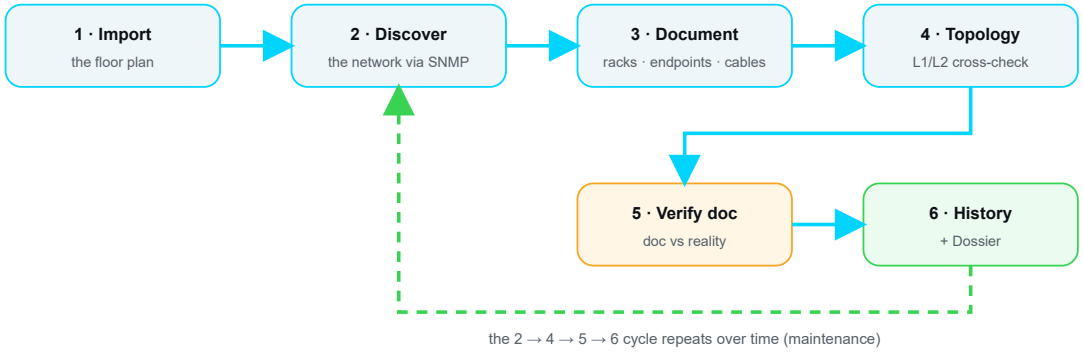


Fig. 9 — The maintenance loop. Phases 1–3 are the initial setup; **2 → 4 → 5 → 6** is the recurring cycle: after History/Dossier you return to **Discover** for the next round.

Two ways to start — and a principle that ties them together

The cycle above is the **maintenance** one. At the very start, though, there are **two paths**, both valid, depending on how the project is born:

- **Planned network (*declare-first*)** — you know your addressing plan. Then: *new project → import the floor plan → **declare all the networks** (VLANs and subnets in the panel) → scan the assigned subnets.* The plan first, then the check against reality. This is the clean path.
- **Inherited network (*discover-first*)** — you don't know the subnets up front: discover what's on the network first, and **declare afterwards** based on what you find.

> **The principle: what's declared is law**
>
> What you write in the **VLAN/subnet** panel is the **project truth**. If you declare a `/16` or a `/28`, the app measures capacity and compares reality **against that prefix** — the scan *confirms* it, it does not redefine it (it's *manual-first* applied to the addressing plan). And whatever it finds **outside** any declared subnet it doesn't hide: it flags it as **"undeclared"**, so you know what's missing from the plan. Declare-first is **recommended, not mandatory**: with no declarations the app still works (it assumes a `/24`), but tells you clearly. See the Dashboard (chapter 15).

Where manual-first lives

- **In discovery** — fields you fill in manually stay yours: SNMP fills the gaps, it does not overwrite them.
- **In topology** — it is a mirror: it shows where cabling and reality agree or diverge, but it does not touch your cables.
- **In verification** — "Update doc" is an explicit action you take; the tool flags, it does not destroy.

Where Verify lands

The outcome of a **Verify** is no longer a flash that vanishes: it **lands in the Dashboard**, in the **"Conformance"** column, and stays there as state — with the differences ready to decide (see chapter 14 and chapter 15). The *loop* never really "ends": the Dashboard tells you when it is worth starting again.

13 AI assistant (advisory)

Query your documented network in plain language — your data stays yours.

The **AI assistant** is an *advisory* chat — a copilot, not an autopilot — that answers questions about **your documented network** in plain language: presence, VLANs, free IPs, **ports**, **SNMP health** and **topology**. It lives as a **third «Assistant» tab** in the right panel (shortcut `A`) and a toolbar button.

> **The principle**
>
> **"InfraNet computes, the AI narrates."** The numbers are computed by InfraNet and passed ready-made; the AI puts them into words and **cites the data**. We never ask the LLM to do network arithmetic — it would get it wrong.

Bring your own key, your choice of provider

A single hook to an **OpenAI-compatible** endpoint: **local by default** (Ollama, no key, data never leaves the machine) *or* any cloud model you choose. Configure it under `Users and access` → `AI assistant`. InfraNet **neither hosts nor pays for** inference and **bundles no model**.

How to configure — local or cloud

Turn on **Enable**, paste **endpoint**, **model** and **key** (write-only: after saving you only ever see "configured"), then **Save**. The header chip shows where inference runs: 🔒 **Local** (green) or ☁ **Cloud** (amber).

Provider	Endpoint	Model (example)	Key
Local — Ollama (default, privacy)	`http://localhost:11434/v1`	`llama3.2:3b`	*empty*
Cloud — OpenAI	`https://api.openai.com/v1`	`gpt-4o-mini`	`sk-…`
Cloud — Anthropic (OpenAI-compatible endpoint)	`https://api.anthropic.com/v1`	`claude-haiku-4-5`	`sk-ant-…`

Local: install **Ollama** and pull a model — no key, nothing leaves the machine. **Cloud:** create the key in your provider's console; quality goes up but **the sanitized data leaves toward the provider**, so use `Show what leaves` first. Any OpenAI-compatible endpoint works (Azure OpenAI, OpenRouter, vLLM, LM Studio): only URL, model and key change.

> 🔒 **Data security — by construction**
>
> Two guarantees, not promises: **(1) the key lives only on the server** and **never returns to the browser**; **(2) only the allow list leaves** — the context is built from the same *allowlist* as the REST API, and the **SNMP community and credentials are not in it**, so they *physically* cannot leave. `Show what leaves` shows the **exact sanitized JSON** *before* you enable anything, and a guard test **fails the build** if a secret ever reached the context.

> **No hallucination**
>
> A hard rule in the prompt: **never invent** names, IPs, MACs, VLANs or ports. If a fact isn't there, the assistant answers "*not in the documentation*" and suggests how to find it. It is **advisory**: it proposes, **you confirm**; any Ansible playbook is a **draft** to review, never applied.

Choose what leaves and what it does

- **Data scope** — what leaves the machine: `Inventory` · `Ports` · `SNMP health` · `Topology` · `Drift`. Turn some off for **privacy** and to cut **cost**.
- **Capabilities** — what it may do: `Q&A` · `Diagnostics` · `Find gaps` · `Suggestions` · `Ansible draft`.

Clickable citations and anti-invention check

Below each answer you see the **sources it used**: cited devices and VLANs become **clickable chips that jump to the node on the map**. A downstream check compares the IPs and MACs the model names against the real data: an address that **isn't in your data** earns a warning chip ⚠ **"reference not found"** — a possible hallucination is *flagged*, not let through. The assistant is also fed **live facts** (the Verify result and IPAM occupancy), so it reasons over current reality, not just the saved file.

Find gaps, suggest, draft Ansible

For "*what's missing*" the assistant uses the **gaps InfraNet already computed**, and for "*which IP do I use*" the **next free address**. Ask for automation and the playbook arrives as a **draft card** with a **"DRAFT — not applied"** banner and a **Copy** button: InfraNet executes nothing.

Every `Verify` row also has an **"Explain"** button that opens the assistant and **seeds a question about that case**, so the *Verify → understand → act* loop stays inside the app.

Onboarding: the «next step» and the spotlight

On first run the Assistant doesn't start from a blank page: a **«next step» chip** tells you what to do *now* based on your project state — empty network → `Discover`; documented but unverified → `Verify`; VLANs without a subnet or gateway; undocumented or absent devices. **«Show me»** lights a **blinking spotlight** on the *real* toolbar button until you click it; **«Ask»** seeds a how-to question. The chip appears **even before a model is configured**, so it can guide Discover and Verify at zero cost.

For *«how do I X in InfraNet»* questions the assistant is grounded on the **real command surface** — the buttons with their tooltips, **derived from the UI itself** — so it cites the **real button label** and never invents menus.

> **Try it**
>
> "Who's on VLAN 20?" · "Which IPs are free?" · "What's on port 5 of the switch?" · "Why is this device absent?" · "What's the printer's toner level?". Local default: install **Ollama**, pull a model, endpoint `http://localhost:11434/v1`, leave the key empty.

> **Administrators only, by choice.** The chat spends the administrator's API key and sends the project's topology to an external provider. Opening it to viewers would take a rate limit and a spending budget first: it is a decision, not a setting left behind.

What the assistant cannot see, it says

The context sent to the model has caps. When what gets cut is a **relation** — the topology, a device's ports, a PDU's outlets, a host's VMs, an access point's SSIDs — the data is accompanied by **how much was shown and how much there was**, and the assistant is told not to answer about what lies beyond. Cutting a *list* is honest: the reader knows less, and everything they know is true. Cutting a *relation* is not: one missing link is enough to make two connected halves look separate, and from the inside you cannot tell.

> **Passive cabling**
> In a well documented network, wall ports and patch panels are **half the devices**, and their record is near-empty by design: no IP, no MAC, no VLAN. Above a threshold they come out **summarised** — how many and of what type — while their **names stay in the topology**, which is where a path reads hop by hop. Measured on the reference network: the context drops by **56%** and of twenty exercise questions it loses **one** — which patch panel position a given cable uses. The assistant knows that and says so, instead of guessing.

14 Documentation check (Drift)

Stop hoping the doc is right: the app tells you exactly where it is not.

Documentation ages: a port moved during a fault, a VLAN changed on the fly, a cable unplugged and never updated. That divergence is called **drift**. The **Verify** button queries the real devices and **compares what is documented with what is there**, showing only the differences.

> **In one sentence**
>
> It stops asking you "*I hope the doc is right*" and starts telling you "*here are the 4 points where it does not match reality*".

> **The result is STATE, not a flash**
>
> Verify used to open an overlay that vanished at the first click. Now the outcome **lands in the Dashboard**, in the **"Conformance"** column (chapter 15), and **stays there as state**: it survives save/reopen, and every category of difference becomes a **navigable row** with its own actions. This chapter explains *what* Verify measures; chapter 15, *where* you read it.

> **What about virtual machines?**
>
> Verify looks at **VMs** too, but does not count them among the red categories: it writes what the network says on their card, beside the state you declared (chapter 5).

The seven report categories

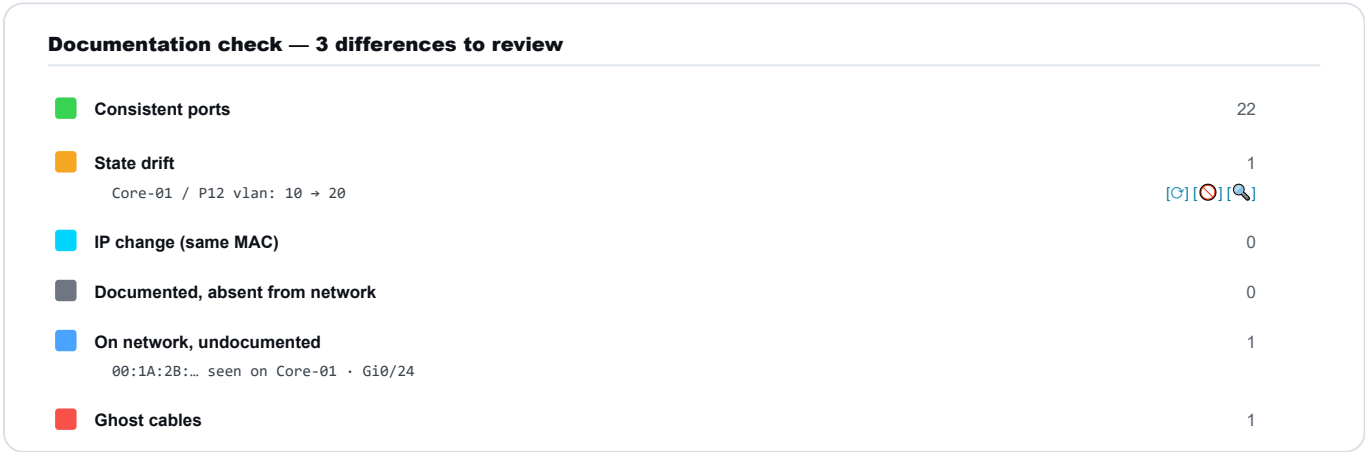


Fig. 10 — The differences, ordered by category. Each row carries its three actions — ⟳ **Update doc**, 🚫 **Ignore**, 🔍 **Investigate** — and now lives as a **navigable row** in the Dashboard's "Conformance" column (chapter 15).

Category	Meaning
🟩 **Consistent ports**	Doc and reality match (count only).
🟧 **State drift**	Documented port active but down, or different VLAN/speed.
🟪 **Hardware identity**	The serial or model reported over SNMP doesn't match the documented one: the device may have been **replaced** (a different firmware is informational only). Accept the new identity in one click, or ignore. A field you wrote by hand is **locked** (closed padlock) and is not contested.
🟦 **IP change (same MAC)**	Device alive at a different IP (DHCP lease changed) → update in one click.
⬛ **Documented, absent**	No response to any signal (SNMP, ping, ARP, TCP) *and* its subnet was reached → greyed out on the map.
🔵 **On network, undocumented**	A MAC appears on the switch but is not on the map → you can "add to map".
🟥 **Ghost cables**	Documented cable with port down across several consecutive checks (threshold: 3).

LAN networks — coverage and presence per subnet

Among the rows of the **"Conformance"** column is **"LAN networks"**: from the documented devices **and the DHCP leases** it derives every **/24** and classifies it — 🟢 **covered** (a reachable SNMP switch), 🟠 **blocked** (the switch is there but does not answer) or ⚪ **open** — each with a **"Discover network"** button that opens *Discover* pre-filled. **After a check** each /24 also shows a **presence badge**, and the **non-verifiable devices** are **nested under their own network** rather than in a separate category: they are two readings of the *same fact*. Each /24 carries a **plan-provenance** tag: **"declared /N"** if it falls inside a subnet you declared, or **"undeclared"** if it is just the assumed `/24`.

Multi-signal presence + BYOD noise

Presence is not judged by SNMP alone: every documented IP is also queried with **ping, ARP and TCP**, so PCs, IoT devices and UPS units do not end up among the "absent". Phones and laptops on the guest Wi-Fi are **collected in a collapsed grey row**; expanding it, each says **why it is hidden** in plain words, and a «Show user/BYOD» toggle brings them back. Finally, **red «absent» requires proof a live host cannot suppress** — a **local ARP-miss** or a **switch access port down for several syncs**: mere silence or an unreached subnet yields **grey «not verifiable»**, never a false absent, and a plain **Sync** never paints anything red. A device behind a router with SNMP stays **green even across subnets**, because the router's ARP or **IPv6 Neighbor Discovery** cache proves it alive on its VLAN.

Presence colours on the floor plan (floor nodes)

After a Sync or a Verify, **floor nodes** are coloured to convey presence; **rack devices** are not, because there the status is already the **SNMP LED**. 🟢 **Green** = present, from any positive signal, a router's ARP or ND cache included, so green works **across subnets**. 🔴 **Red** = *proven* absent, from signals a live host cannot fake. ⚪ **Grey** = not verifiable: ambiguous silence, the honest "don't know". With **«Released lease = likely disconnected»** on, a grey device whose lease is released is annotated as such but **stays grey**: a hint, not a proof.

Management VLAN: the opposite of a guest VLAN

You can mark a VLAN as **"management"**. An *unknown* device showing up there is never hidden as BYOD but treated as **infrastructure** and flagged with a red **"On management VLAN"** badge — an intruder on the management network is exactly what you want to spot at once.

Automatic IP renewal (opt-in)

You can enable automatic IP renewal from DHCP: a device's identity is its **MAC**, the IP is an attribute that rotates. IPs set manually are *never* touched, and every renewal is logged in History.

DHCP leases as a source (cross-VLAN)

Behind a firewall that routes between VLANs the local ARP table is blind, while **DHCP leases** cover *every* VLAN. From `Automations` → `DHCP` → `DHCP leases` → `Load` you **paste or load** the lease table (format auto-detected: ISC dhcpd, dnsmasq, Kea CSV, generic CSV). Leases feed the same engine: **cross-VLAN IP change**, per-MAC presence and **undocumented devices** with their hostname. Multiple servers **accumulate as persisted sources** (merged per MAC, newest lease wins).

An identity map, not a liveness probe

A lease table says *who has which IP*, not *who is powered on*: a documented device **missing** from the leases lands in **Not verifiable**, never among the absent — it may have a static IP. **Expired** leases are ignored, and a **manually-pinned** IP is never silently overwritten: the row asks for confirmation. Loading from file or paste is built-in; **live** pull via vendor APIs is an optional driver pack.

The result does not wait for Save

Who is there and who is not is **stored the moment it is measured**, without pressing Save: that reading is not an edit you made, it is a **measurement of the network**. Reopen the project and devices that are switched off are still red — even with autosave off, and even when the check was run by the **automatic monitor** while you were away.

The document stays yours

Your unsaved edits stay unsaved and the project file is not rewritten behind your back: presence lives beside the history, not inside the document. If a fresher measurement exists, that one wins — old data never drags newer data backwards.

15 Dashboard (summary view)

Three questions, one screen: what's missing, what no longer matches, how much room is left.

The **Dashboard** is a **read-only** view that **never touches the document**. It answers three questions in three columns: **LAN** (does it describe everything?), **Compliance** (does it still match reality?), **Expansion** (how much can I grow?). Every tile carries its number and a **verdict in words**.

In one sentence

It is not a metrics cockpit but the answer to "is my document complete, current and with headroom?", with **where it comes from** next to every number.

Who counts in "Answering over SNMP"

The **driver** in **Properties → Integration** decides: if it is set, the device is counted; without it, the device is **not on this tile**: neither counted nor listed, because there is nothing to do about it from here. This holds for *any type*: what counts is whether you poll it, not what it is.

One ratio, a small figure, and the names

The tile carries **one ratio**: how many **answer** among those you poll (*"12 of 13"*). Next to it, small and grey, the **number** of devices you do not poll (*"20 not observed"*): it is not an alarm and opens no list, but it keeps a green "1 of 1" from staying silent about the other ninety-nine that nobody looks at. It is absent on the dashed row: the verdict already says "no SNMP access configured".

When something is wrong, the verdict writes the **names** straight away — at most three, then "+N".

- **No answer** (red) — the last read *failed* (*"no answer: OfficeJet8715"*): fix it with credentials, by powering the device on, or by no longer polling it.
- **Never queried** (amber) — the access is configured but nobody has read yet (*"never queried: SW-Floor2"*): not a fault, a job to do. One **Verify** is enough.
- **Green** — everything you poll *actually answered*. With *zero* access configured the row is dashed, not green — and so it is when the accesses exist but no read has ever happened: *"configured, never queried"* is not a zero, it is a missing measurement.

The count covers **every** device with SNMP Integration — printers, NAS, cameras, UPS included. For those without, the **Check** still verifies their *presence* (FDB, ARP, ping).

What's new: the Check lives here

The middle column **"Compliance"** is **the home of the Check**: launched from the toolbar, the Dashboard opens here by itself and leaves the result as a **state** that stays, not a window that disappears.

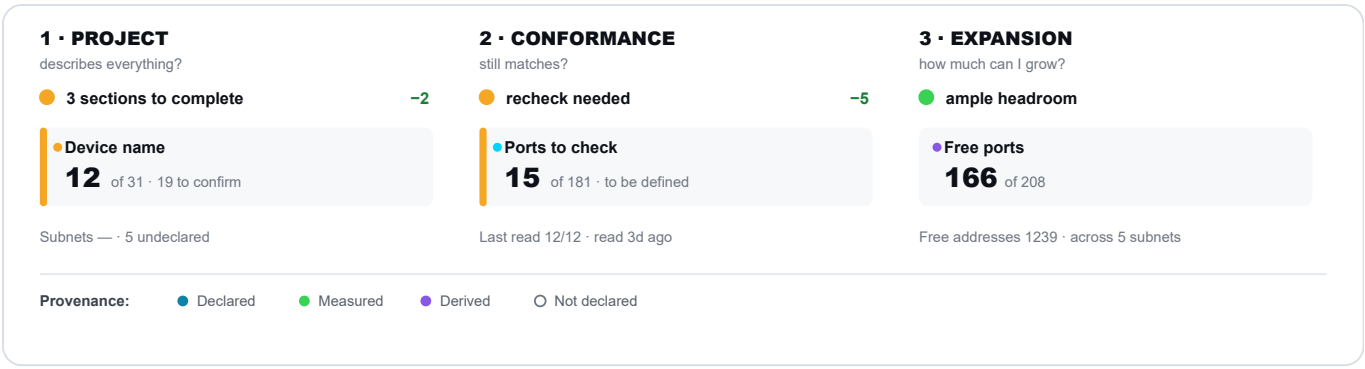


Fig. 11 — The three columns on *Network+Lab*: health dot, delta since the last read (−2, −5), a bar on the most urgent item, provenance on every number. After a Check, "Compliance" shows the outcome (Fig. 12).

> **One pill, everywhere in the Dashboard**
>
> In the lists a click opens, every label has the **same shape**: where an entry came from (*Measured*, *Derived*, *Declared*) and the **reason** it is on the list (*no gateway*, *unverifiable*) are read with the same sign — what tells them apart is the **word** and the colour, not two different drawings. And the certainty words are the **six from chapter 3**: there is no second alphabet to learn here, and the absence carries the **hollow ring** here too.

The "Compliance" column: where the Check lands

After a **Check**, the **when strip** at the top carries two dates — the last *Sync* and the last *Check* — and below it appears **one row per kind of difference**: ports that changed state, undocumented devices, ghost cables, hardware identity, changed addresses. Each opens *in place* with the **three actions Update doc**, **Ignore**, **Investigate**.

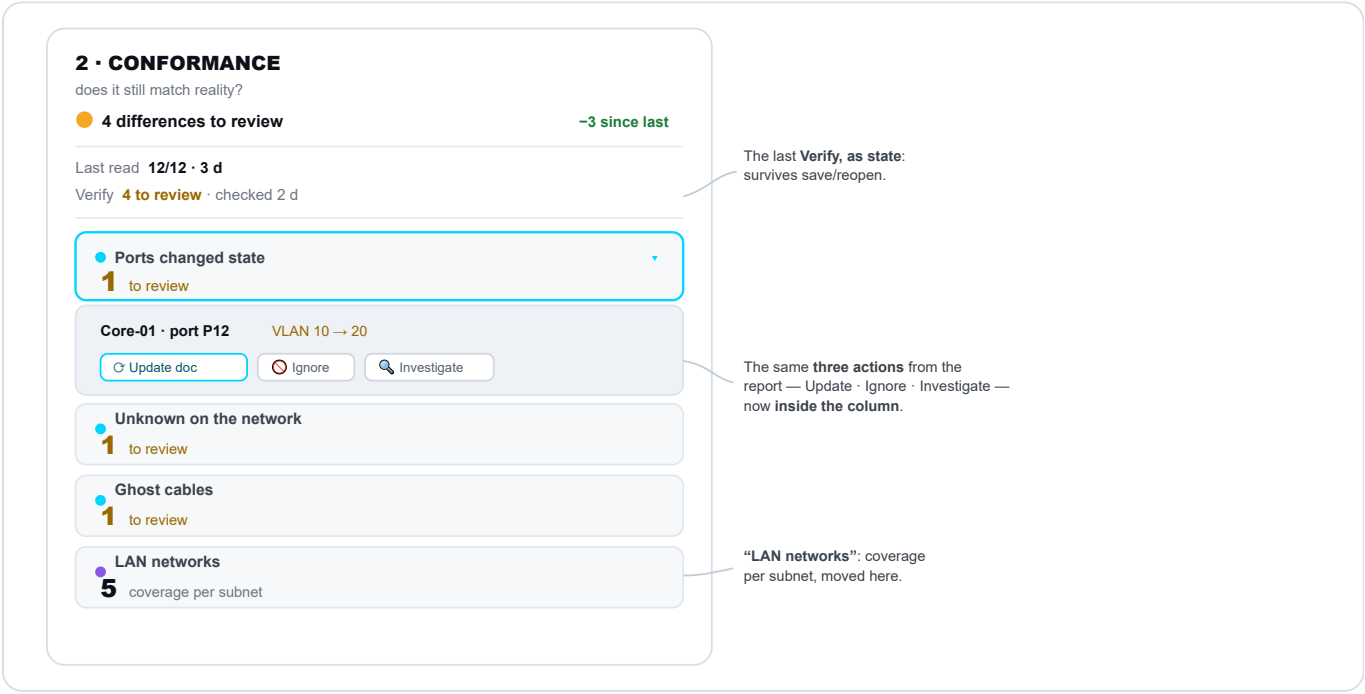


Fig. 12 — "Compliance" after a Check: two dates on top, one row per kind of difference; the open row shows the three actions **inside** the column, the dot shows **provenance** (measured · inferred).

The provenance of every number

The Dashboard **never presents an estimate as a fact**: a dot says where each number comes from, in the six words of ch. 3. Missing data comes out **dashed** — "not declared" when your writing is missing, "no reading" when a measurement is — never as a zero.

Provenance	Meaning
Measured	Read from the device at the last SNMP Sync/Check.
Declared	You wrote it in the document.
Derived	Derived from other signals (e.g. free addresses with an assumed `/24`).
Not declared	Nobody wrote it: dashed tile, never a zero.
No reading	Nobody *read* it: a measurement is missing, not your data. Same empty ring — what tells them apart is the word.

Subnets follow the DECLARED prefix

The **addressing plan is law** (ch. 12): a subnet declared in the VLAN panel — a `/16` as much as a `/28` — counts **with its real prefix**, not with a convenient `/24`: **"Free addresses"** counts the true capacity and **"Subnets"** puts the **declared ones first**. Networks in use but not declared come out marked **"not declared"** (a `/24` assumed as a fallback) with the verdict **"N to declare"**; devices outside a declared subnet that is too **small** surface the same way, neither hidden nor counted twice.

The glance: one verdict per column

At the top of each column sits a **health dot** (🟢 fine · 🟡 minor gaps · 🔴 serious problem) with a sober sentence. **Red is reserved** for a network that was *never read*; a project read recently but with discrepancies stays **amber**. Next to it, if a previous read exists, the **delta "since the last read"** (**−N** · **+N**).

The item to start from

In every column the **most urgent** item gets a **coloured bar on the left**, amber or red by severity; if the column is fine there is no bar — highlighting "166 free ports" would say something false.

Looking does not dirty the document — with one deliberate exception

Opening the Dashboard and clicking around **never marks the project as modified**: the chosen lens and the "since last read" comparison live in the **browser** (localStorage). **The one exception**: the **three actions** of the "Compliance" drill-down, which touch the document and mark it modified — *deciding* a difference is a change.

Every row opens in place

A click on a row **lists what is behind it**, *inside the column*: **cables** are listed *all* of them, both ends resolved by name and split into tabs by **how we know** — **measured**, **declared**, **derived**, and after a Verify also **contradicted** (the network denies the cable) and **not read** (an inference whose evidence evaporated); **LAG aggregations** in the three tabs of their origin, each named after the devices that *own the ports* of the group; **LLDP/CDP neighbours** give a name to a bare chassis MAC; **free addresses** give the free ones *out of the usable* per subnet; **free ports** count **switch** capacity — patch panels, appliances (NVR, KVM, console server, NAS, firewall…) and **wall outlets** stay **apart** ("+N non-switch") — in two tabs, **In rack** and **Outside rack**. In **"Compliance"** the rows open the **cases to decide**: the only place you *act* from.

…and the entries take you there

The detail entries are **live**: a click leaves the Dashboard and takes you to where the thing is seen or edited. An inferred **cable** lights up its **physical path** across the floor plan and the rack; a **subnet**, a **gateway** or a **VLAN** opens the panel where it is **declared**, already on the right entry; an **LLDP/CDP neighbour** that matches a cable **lights up that cable**.

The detail reports live here

Where a tile has a verdict, its **full report** opens from the drill-down: the **free-port table** (**CSV** export) from *Expansion*, the **L3 → gateway map**, exportable too, from the *Gateway* tile. These were entries of a separate "Reports" menu; the toolbar keeps only **Change history**, a log over time rather than a snapshot of state.

The six lenses: from summary to recoverability, security and health

At the top a **lens selector** — `Summary` · `Recoverability` · `Security` · `Health` — changes *what* the view shows. **Summary** is the three columns just described; the others are **full-width**, opt-in lenses.

④ Recoverability (DR) — "if it dies tonight, can you bring it back?"

The lens counts the **managed devices** and gives a verdict **"X of Y recoverable"**; three pieces are needed:

- **A fresh configuration backup** — the pointer to *where* the backup lives (never the file), updated within the last **30 days**.
- **Known hardware identity** — make/model (declared or via SNMP) *or* the serial; a missing **firmware** is only a warning. If the declared serial does not match the measured one, the device is marked "replaced?" and does **not** count — unless you wrote the serial yourself: a locked field is not contested.
- **Location** — the rack it goes back into.

Alongside, three *advisory* signals that do not lower the verdict: **presence** (recent or stale), **redundancy** (a declared **HA** twin: single points of failure surface by difference), **lifecycle**. Each row lists the devices missing that piece, with the IP next to the name.

> **Lifecycle — "when it dies, can you still replace it?"**
>
> In **Properties**, two dates: **Warranty until** and **End of life (EOL)** — **declared and nothing else**, no OID provides them: an empty field means "I don't know", never "unlimited". The row summarises from the most severe: **out of production** (past EOL), **out of warranty**, **expiring** within **90 days**; the denominator is *who declares a date*, not the fleet, and the verdict says how many carry none. It is *advisory*: a device out of production with a fresh backup stays **recoverable** — EOL says you cannot *buy it again*, not that you cannot bring it back up.

⑤ Security & Services — "how exposed is management?"

Only **measured or declared** signals, never invented ones:

- **Encrypted SNMP access** — how many devices speak **v3** and how many still **v1/v2c** (community in the clear); the drill-down lists the weak ones.
- **Default community** — how many use a *guessable* community (public/private). The **raw value** *never* leaves the engine; only an **administrator** sees in the drill-down *which* known default it is (public/private/empty), never a *custom* community.
- **Management VLAN** — how many VLANs you marked as management; the drill-down lists them **by name**, more than one included.
- **Wireless VLAN coherence** — how many **SSIDs** announce a VLAN that their access point's *uplink* does not carry; the drill-down lists them with the AP that broadcasts them, **zero = coherent**.

> **The honest verdict**
>
> **Red** with a reachable guessable community; **amber** for SNMP in the clear, no management VLAN or an SSID outside the trunk; **green** when management is hardened; **grey** until you configure *even one* SNMP access — zero measurements does not mean zero exposure.

⑥ Health — "what is wrong ***right now***?"

The only lens on the **present**: it adds no polling, it **composes the telemetry already returned** during the Check:

- **Resources (RAM · disks)** — memory in use and full volumes, read via HOST-RESOURCES.
- **Continuity (UPS)** — on battery, runtime left, charge, output load.
- **Consumables** — toner and ink running out (Printer-MIB).
- **CPU load** — the busiest among those that reported it: a *number*, not a verdict.

The **denominator of every row is whoever actually answered** with that telemetry: a NAS that does not speak Printer-MIB is not in the count, and the row stays **dashed**.

> **Yesterday's health is not today's**
>
> Without *even one* read the verdict is **grey**, never green; a green **decays to amber** when the **oldest** read passes **6 hours**. Running the **Check** again brings it back to the present.

> **“What I am not looking at” — the declared perimeter**
>
> At the bottom of every lens a sober row names the dimensions this summary does **not** judge (the full text is in the *tooltip*): WAN/circuits, L3 routing, spanning tree, firewall/ACL, AAA·syslog·NTP, backup restore testing, temperature and interface errors, trunk symmetry, 802.1X/NAC. The list is **alive**: a dimension that gets modelled disappears by itself — as just happened to *power/UPS*, *device health* and *warranty/EOL*.

> **A verdict has an age too**
>
> After **seven days** with no Sync or Check, a green verdict in **Compliance** and **Expansion** — the columns that rest on a measurement — turns amber: *“read N days ago — re-check”*.
> The gate does not touch **LAN** (a document does not age) nor **Recoverability**, which already has its own clock (the 30 days on the backup).

Inside “Compliance”: IPAM conflicts, declared status and VLAN names from the network

Two IPAM checks and one declared field complete the middle column:

- **Declared status** — the **operational status** you write in **Properties** (*planned · staged · in stock · in service · failed · decommissioning · out of service*), held up against the measurement. The big number is the **contradictions**: a device that **answers** while you have it down as out of service — there it is the documentation that has gone stale, and whenever there is at least one it takes the column's headline. Beside it, how many **absences the declaration explained**: those are the reds we did not light, and they deserve saying out loud. If *nobody* declares a status the row stays **dashed** rather than a green zero: that is a gap, not a pass.
- **IPAM conflicts** — the **same IP** on two documented devices (VMs included) or two **VLANs with overlapping subnets**; zero = green, otherwise the drill-down lists the devices and the shared address, or the overlapping subnets.
- **VLAN names** — the names *read via SNMP* compared with the declared ones: the Dashboard fills the gaps (a VLAN in use with no name) and flags the **conflicts** (your name ≠ the network's, or two switches disagreeing). The **declared value is never rewritten**: you decide, from the VLAN panel.

> **Status changes how silence reads, never the measurement**
>
> A device declared **planned** that does not answer is **not a fault**: it is not installed yet, and it stops being red on the floor plan. The Verify still records the absence exactly as before — only *how it reads* changes. An empty field means **not declared**: a project that does not use this field behaves in every respect as it did.

16 REST API and automation (Ansible)

Let other tools read your documentation — read-only, with a token.

InfraNet can become a **programmable source of truth**: dashboards, wikis and automation read the network inventory through a **read-only REST API**. It stays *manual-first* — you write the truth, machines consume it.

Access tokens

An external tool authenticates with a **token**, created from the user menu → `Users and access` → `API tokens`:

- pick a **label** (e.g. `ansible`) and press `Generate token`;
- it is shown **only once**: copy it right away;
- the list keeps **prefix** and last-used date; you can **revoke** it anytime.

> **Admins only, read-only**
>
> Token management is restricted to admins, and tokens grant **read-only** access.

What the API exposes

Responses are **sanitised**: type, IP, MAC, VLAN, rack and brand, but **never** secrets such as SNMP communities.

Endpoint	What it returns
`/api/v1/projects`	the list of projects
`/api/v1/projects/{id}`	the full inventory: VLANs, racks, devices
`/api/v1/projects/{id}/devices`	just the device list
`/api/v1/projects/{id}/ansible-inventory`	the dynamic inventory for Ansible
`/api/v1/openapi.json`	the full API description (OpenAPI)

```
# project 8 inventory, passing the token in the header
curl -H "Authorization: Bearer inp_…" http://<host-ip>:8421/api/v1/projects/8
```

Dynamic inventory for Ansible

The main use case: InfraNet is the **source of truth**, Ansible the **executor**. `integrations/ansible/` ships a ready-made script that turns a project into a **dynamic inventory**: every device with an IP becomes a host, grouped by `type_*`, `vlan_*`, `rack_*`, `brand_*` and `backup_missing`.

```
# three variables and you're set
export INFRANET_URL=http://<host-ip>:8421
export INFRANET_TOKEN=inp_…
export INFRANET_PROJECT=8

# view the groups, then target a group
ansible-inventory -i infranet_inventory.py --graph
ansible type_switch -i infranet_inventory.py -m ping
```

No hand-kept inventory

Update the documentation in InfraNet and the Ansible inventory follows: one source to keep current, not two.

Configuration backup — the pointer, not the file

In Properties → `Configuration backup` you record **where** a device's backup lives and when the last one was taken: InfraNet notes the *where*, **never the file**, which holds secrets. The inventory then carries per host the **network OS** and that pointer, and gathers into `backup_missing` the devices **without** a recorded backup: a playbook finds them on its own. The path accepts no credentials.

17 Change history

Who changed what, and when. An append-only log that survives Undo.

On a documented network, multiple people work over time. Sooner or later the question comes: **"who changed this, and when?"**. **History** is the chronological log of project changes — an *append-only* diary: entries accumulate, they are never deleted.

> **In one sentence**
>
> It turns the map from a "snapshot of today" into a "film of how we got here".

What is recorded

Only **structural events**, not every minor tweak: device added/removed/renamed, cable created/removed, port VLAN changed, Sync and Verify run (with outcome), documentation aligned, project renamed. Every entry carries **date and time, user, action, object and detail**.

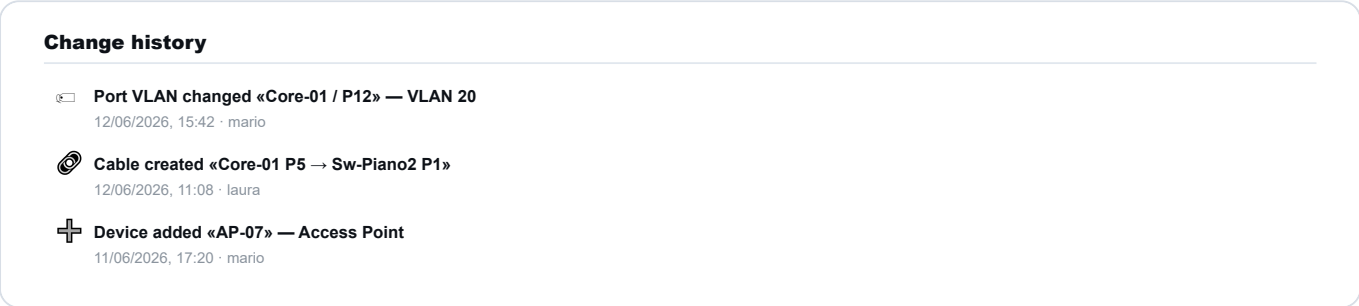


Fig. 13 — The log, most recent at the top. Filter by device/user/action and **export as CSV** for handovers and audits.

- **Cannot be undone.** Unlike everything else, History survives Undo/Redo: a log that can be erased would not be reliable.
- **Export as CSV** to attach documentary proof (handovers, audits, proof of work for MSPs).
- Only meaningful if **each person uses their own credentials**: with a single "admin" account, the "who" column loses its value.

> **What a Sync line says**
>
> "**18/18 SNMP devices read** · no read errors · **25 without SNMP, not read**". The Sync queries only the devices that have an SNMP integration: the line says how many it read and how many real devices it leaves out (rooms and cabling passives do not count). It does not say the rest of the network answers — the Check judges that. The lines of automatic rounds reach the disk **at once**, without waiting for a save; a change to the document (a cable, a renewed IP) stays tied to the save.

Verifications over time and restore points

The same window now has **three tabs**. **Changes** is the log above. **Verifications** is the *timeline*: every Verify (manual or scheduled) leaves a lightweight row with date and divergences, so you can read the network's health over time. **Restore** keeps **full snapshots** of the state: you can *create a point* whenever you like and **restore** it later — before doing so the app automatically saves a safety point. A snapshot is also taken on Save and before risky operations (import, bulk adopt). The whole history lives **outside the project file**, so the data doesn't bloat, behind an interface already ready for a database.

The trend: is this network getting better known?

The **Checks** tab no longer opens on just the list of snapshots: it opens on their **reading**. A bar says *where you are* — the last check split into **consistent**, **mismatched** and **not verifiable** devices — and four sparklines say *where you are going*, that is the shape of the series: a plain "from → to" throws that away, and two equal endpoints often hide a curve that moved in between.

> **Why it may refuse a verdict**
>
> If the share of the network that **cannot be looked at** has grown, the verdict is "not comparable": an improvement in the other curves could come from that rather than from the network. In the same spirit a manual check and an automatic poll are never compared (they are two instruments), the series **breaks** if the network changes size — two different estates do not compare — and a *blind* check, one that could measure nothing, does not drag the curve down: it leaves the trend and joins the coverage. Below three samples there is no trend, and it says so: the confirmed share **matures with use**, it is not there on the first run. And when the trend is flat it gives the reason — which category is holding the mismatches up — because "flat" on its own is not something you can act on.

18 Export, labels and Dossier

Getting documentation out of the app: SVG floor plan, cable labels, handover dossier.

Floor plan and backup

- **SVG** — the printable floor plan, with VLAN-coloured cables and device icons. With a VLAN filter active, the SVG exports only that VLAN.
- **JSON** — the project in a **versioned format**, for moving it between installations. It is **secret-free**: SNMP communities and backup references are **stripped** on export — a backup of the network, not of passwords. On import InfraNet accepts older saved files too.
- **CSV** — bulk import of tens of devices from a spreadsheet or CMDB.

> **SVG = recommended format for the floor plan**
>
> The SVG export keeps the floor plan **vector**: you can print it at any size — plotter included — **without losing sharpness**, unlike a raster image that pixelates.

Export the rack to draw.io

From **Import/Export → Export to draw.io** you get a `.drawio` file with **one page per rack**. It is not a screenshot: frame, devices and ports are **native, editable shapes** inside draw.io's own rack container, each device snapped to its U slot. Move a server, rename a port, add a note — it all stays editable.

- The **device names** sit **outside the rack**, leaving the device face entirely for the interfaces.
- The **SNMP status stripe** is not exported: it is a *live* indicator, whereas the file is a *static snapshot*.
- A device with a **skin** carries its artwork over as a vector image with the real ports overlaid.

> **One cable layer per VLAN, with a table**
>
> The export includes the **intra-rack cables** as native connectors, split across **separate layers — one per VLAN**, each **named after the VLAN**, coloured by VLAN and routed in dedicated lanes so they don't overlap. In draw.io open **Edit → Layers** and turn on the VLAN you care about: alongside its cables a **table** lists the connections, one cable per row.

> **Isolate a cable: click the row to highlight it**
>
> In diagrams.net's **View** mode, **clicking a table row highlights that cable**: the line thickens, *stays* highlighted and the view scrolls onto it. Clicking the header clears it.

> **A4 format, or A3 if it doesn't fit**
>
> Each page is exported as **A4 portrait**; if the content doesn't fit — a very tall rack, a VLAN with many cables — the page switches to **A3** by itself.

What about Visio?

Visio does not open `.drawio` files. If you need it, open the file in diagrams.net and use **File → Export as → VSDX**: the result is flattened, because draw.io's native rack shapes have no Visio equivalent.

Printing cable labels

From **Import/Export → Export labels** you generate cable labels ready to print on standard market media:

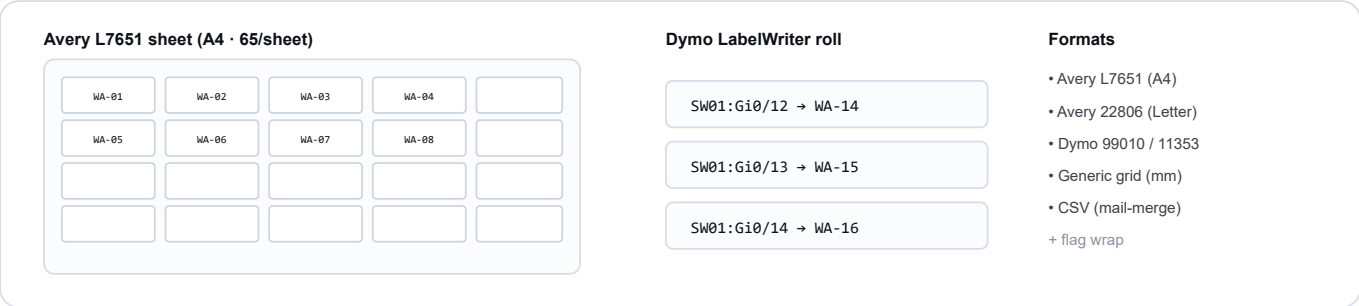


Fig. 14 — Labels on **Avery** sheets (L7651 A4, 22806 Letter), **Dymo** rolls (99010, 11353), a generic grid configurable in mm, or CSV for mail-merge. Preview is live.

- Choose the **fields** to print (by default only the label ID) and the **flag wrap** option, which repeats the ID in both halves so it reads from either side.
- Geometries are nominal: do a **test print** against the label sheet to check alignment; any offset can be corrected with the "generic grid".

Handover dossier

The dossier lives inside "Export PDF"

Import/Export → Export PDF opens a single dialog, and at the top of it sits the **Handover dossier**: the **cover page** with the key figures up front and the **change history** at the back, its checkbox **always on and locked** — every PDF is a dossier, so there is no second menu entry to look for. Below it, the **sections all start switched off** and you switch on what the recipient needs: floor plan, **Dashboard**, rack fronts, cable inventory, as-built path trace, port assignment, VLAN/IPAM, **virtual machines**, **power (PDUs)**, asset register, **WAN (the site map)** and the rest. **Select all** gives the complete dossier in one click. For an MSP it is a *billable deliverable*; for a company, *insurance against staff turnover*.

Power: one restore sheet per PDU

The **Power** chapter opens with a totals line, then gives **each PDU a sheet** with everything needed to put it back: identity (brand, model, serial, firmware, asset tag, warranty), position, electrical rating, management (mode, ports, IP and MAC), the **backup pointer**, what it is **powered from**, and the **outlets and loads** list — for every outlet its state, the device it powers, that device's power port, and whether the link was *imported* or set *by hand*. A sheet never splits across two pages: tear it off and you have that PDU whole. An undeclared field prints a dash, *never* a zero; and a PDU that declares an outlet count without listing the outlets says so, instead of passing a row of zeros off as a measurement.

Notes sit next to their device

The **note** you wrote on a device is printed **under its own row** in the **Asset register**, not in a chapter of its own: a separate chapter forced the reader to match names back to devices before the note meant anything. Notes written on *structural cabling* are not printed: the register lists IT assets, not the building's wiring.

The Dashboard up front

Right after the floor plan, one page with the **three questions** and their verdicts — the same ones you see on screen, with the same **provenance dot** next to every number. It is the *executive summary*: the reader gets the judgement first and the data backing it later. It lists no devices, so it fits on **one page** and actually gets read. At the bottom, the **"What I'm not looking at"** line travels with the dossier — putting in writing what was *not* assessed protects whoever hands it over as much as it is honest towards whoever receives it.

Virtual machines in the dossier

VMs get a **chapter of their own**: one row per machine, grouped by host, with role or service, state, VLAN, IP, allocated resources, owner and criticality. On the cover they have **their own counter**: they are *never* added to the device count, because a host with ten VMs is still one installed appliance. Whatever you left blank prints as «-»: the dossier invents nothing.

The "Recoverability (DR)" section — the runbook to rebuild from

An optional page, **one row per managed device**, built for the day of the disaster: **where** each config backup lives (pointer and method, *never* the file or the credentials), the **date and time** of the last one, the **serial and firmware** to procure and reflash, the **lifecycle** (declared warranty and end-of-life dates, summed up by the worst state) and the **rack** it goes back into — under a header verdict **"X/Y recoverable · N without a backup · N without a location · N with an unresolved identity · N end-of-life"**, the last four shown only when greater than zero. The thresholds are the *same* as lens ④: the dossier you hand over cannot say something different from the app. Kept *off-site*, this page lets you rebuild the LAN even with InfraNet down, and carries the same secret-free allowlist as the asset register.

WAN: the site map, and what it takes to rebuild it

The **WAN** chapter is the floor *above* the project: the sites declared in the **Multi-site** panel (chapter 21), drawn as you see them on screen — **vector, on a white ground**, so it stays readable printed and zoomed — followed by **one card per WAN line and per link**. Each card holds what you need at 3am: the **provider** and the **circuit id** you dictate on the phone, the **committed rate** (*never* the port speed), the **SLA** reference, the **WAN interface** and the **public addresses**; for a link, its kind, its state with *who says so*, VRF/service/overlay, the IKE version, the **device at each end**, the **peer address** (the *other* end's, as seen from here) and the **reachable networks** — which on an IPsec link *are* the encryption domain: bring the tunnel back up without them and nothing passes through it. Unlike the PDU sheets, the four fields you telephone with stay printed **even when empty, with a dash**: a line with no circuit id is something to see *now*, not on the night you need it. The header **counts the chapter's own gaps** (lines with no id or no provider, links with no declared networks, sites with no line). With no site declared the page says so, rather than disappearing.

For a leaner dossier, leave off the sections you do not need — even all of them: the cover page and the change history always print. Tip: run a *Sync* just before exporting, so ports, VLANs and status are up to date.

19 DCIM / IPAM import (NetBox)

Build a new InfraNet project from an existing NetBox — read-only, and with no surprises.

If your network is already described in **NetBox**, you don't have to redraw it: InfraNet **imports** it and creates a ready-made project with racks, devices, VLANs and cabling. The import is **free** and **read-only** — it reads, and never writes back.

> **Where to find it**
>
> Menu **Import/Export** → **DCIM/IPAM sync**. Write-back to NetBox is a **paid module**: in the free build the "Export" tab stays locked.

1 · Connection

Enter your NetBox **base URL** and an **API token**, then press **Test connection**: the button turns **green** if it answers, **red** if it doesn't, with the reason. The token is kept **server-side only**, never comes back to the browser and never lands in the saved project — the same handling as the SNMP community.

2 · Import in three steps

Step	What you choose
① **Scope**	The **site** to import, with counts next to it. You import **one site at a time**.
② **Entities**	What to bring in: **Devices + ports + cables**, **IPAM** (VLANs and prefixes), **Racks**.
③ **Preview**	The **estimate**, the **list of decisions** and per-row deselect → **Create project**.

On confirm a **progress** screen appears, then a **result** with the counts and an **Open project** button.

> **One site = one project**
>
> The site names the project, racks keep their NetBox names, and a device's *Location* becomes a note. **Step one opens on choosing the site** and "Next" stays disabled until you pick one: a site at a time keeps rack names unambiguous and lets the project record which slice of the DCIM it came from, which is what makes the later comparison exact.
>
> "**Import the whole DCIM**" is still there but it is a detour: beyond one site the rack names collide and the comparison loses its target.

3 · What it rebuilds, and how

- **Rooms from locations** — NetBox *locations* (the floor, the server room, the offices) become **rooms** on the plan, with their racks and devices drawn inside them. NetBox does not say *where* they are or *how big*, though: those numbers are our choice and the import says so — the size comes from the contents, two racks make a small room and twelve a large one, and from there you move and resize them as you like. A location with **nothing in it** does not become a room, and a device **with no location** stays out, drawn below the rooms: dropping it inside a rectangle would claim a membership the DCIM never declared. **Nested locations** are no longer flattened into a name: the parent becomes a **storey** and the child a **room** inside it, keeping its own name — inside a box already labelled “Floor 1”, a room called “Floor 1 · Server room” says the same thing twice. **Two levels are drawn, not N**: with Building → Floor → Room the storey is the *immediate* parent, and whatever sits above stays in its name (“Building A · Floor 1”), because faking three levels with two is the same lie one storey higher. A location holding devices of **its own** as well as sub-locations becomes a storey, and those devices sit inside it next to the rooms rather than in a room NetBox never declared. A parent the read did not bring back — out of scope, truncated — does not become a storey at all: that would be a rectangle with an id inside it.
- **Racks on the floor plan** — each imported rack is a clickable floor icon: **inside its room** when the DCIM declares one, otherwise on a non-overlapping grid. The first one opens already populated.
- **Front/rear split** — a cabinet with devices on *both* faces becomes **two** InfraNet racks (the rear as "*name* · retro"); cables crossing the faces become links between them.
- **Patch-panel cabling** — front/rear-port terminations are rebuilt as a **native pass-through chain** (switch → panel-A → panel-B → server). Power cables and WAN circuits don't become topology links: a PDU's connection lives in the *outlet*.
- **PDUs, UPSs and power** — InfraNet imports their **outlets** (up to 48) with **status** and **powered device**, the power **inlets** and the **management** ports. A UPS's outlets are the only thing that says **who stays on when the power goes**. The **ATS** is deliberately left out — its point is its *two inputs*.
- **Stacks** — a *virtual chassis* becomes a **stack** again: chassis name, member position, master. Members stay **separate devices** but now know they belong to the same stack.
- **Catalogue** — the model is matched against the built-in catalogue (native port count and front panel); if it isn't there, the imported interface count is used.
- **Platform and description** — the declared *platform* picks the network OS for the **Ansible inventory**; the *description* lands in the **notes**.
- **Addresses on ports** — an address belongs to the *socket*, not the device: every address declared on an interface now reaches the **port address** field. A port holds **one** and wants it IPv4; what does not fit is **counted and declared** rather than dropped.
- **Reserved addresses** — an addressing plan also declares addresses on no device at all. They don't become devices, but **free they are not**: they land on the *network* and count towards its occupancy, so "next free IP" stops offering an address someone already booked.
- **VLAN roles** — a DCIM gives its VLANs a *role*, and InfraNet has four declarations you fill in by hand: **management**, **voice**, **guest** and **native**. The role's name is chosen by whoever fills the archive in, so the import **does not guess**: you match it *once per role* in the preview.

- **Wi-Fi** — an 802.11 interface is an **antenna**, not a port: it comes in as a **radio** with its band, and underneath the **SSIDs** it broadcasts, each with its VLAN and encryption. The **channel** comes in only when comparable: a DCIM declares a *wide* channel while InfraNet asks for the primary 20 MHz one, and that block covers four without saying which — so the band comes in and the channel is left for you. The cap is eight radios per device, and the ninth is declared.
- **Virtual machines** — every VM lands **inside its host** with name, role, operating system, state, resources and network adapters; its address joins the address audit like a device's, and its VLAN feeds the derived *trunk* on the host's uplink. NetBox can name the **physical machine** or only the **cluster**: with a single imported device of that cluster the host is unambiguous and the VM lands there; with two or more it **stays out** with a row explaining why — picking one at random is not documenting. A device the DCIM puts VMs on **keeps the type the DCIM gives it**: a storage box with virtual machines stays a storage box. The type only changes for something that could not host them at all — a switch, a firewall — where the data would otherwise have nowhere to live, and then a row says so.
- **Device kinds** — the NetBox role picks the InfraNet type, and not only for infrastructure: cameras, VoIP phones, PCs, displays, projectors, door controllers, sensors, tablets and KVMs each have their own. An unknown role stays **generic**, but follows *where it sits*: the floor generic when there is no rack.

> **VM memory and disk change units**
>
> NetBox counts memory and disk in **megabytes**, InfraNet shows **gigabytes**: the import divides by 1024 and **says so** in a row, rather than hiding the conversion inside a number. Worth opening one VM to check the values add up.

> **If the virtual machines do not show up**
>
> The host panel is empty when **no VM in the DCIM is attached to a device you imported** — in NetBox the VM's "device" field is optional and many archives only record the cluster. The panel always says how many exist and how many came in, because an empty list with no explanation reads as a defect in the application.

> **What changed in the DCIM since you imported**
>
> The preview carries a **"Compare with the open project"** button. It imports nothing and **writes nothing**: it lines up the differences — this device has another name, this network is new, this VLAN is gone — with the document's value and the DCIM's side by side.
>
> Three things it deliberately does not do. It **does not accuse you of your own work**: a device you added by hand has no DCIM id, so it never shows up as "gone" — hence no "apply all". It **does not compare measurements**, only declarations. And it **does not assign blame**: with no timestamp per field nobody knows whether you corrected the name or NetBox did.
>
> The comparison **narrows itself** to the slice of DCIM the project was born from, and the line at the top declares it. A project imported before 2.9.2 has no such origin: it falls back on the scope you pick in step 1, and that is stated too.

You match the VLAN roles yourself, once

One row per **role**, with the VLANs it touches and a dropdown: *do not declare* (the default), management, voice, guest, native. There is **deliberately** no automatic match: "Access - Wireless" is not the guest network, and a rule looking for the word "wireless" would have declared the whole corporate network a guest network without asking. The three lists take *every* VLAN of the role; the **native** VLAN is a single value, so a role touching more than one is declared rather than resolved at random. A role that sits only on *networks* is declared too.

Wi-Fi encryption is a reading, not a fact

NetBox says "personal" or "enterprise" and **never says the generation**. Something had to be written: **WPA2** goes in as the common case, and a row says so. If those networks run WPA3, correct them in the radio panel — from that moment the value is yours. **WEP**, which the model has no word for, is left **empty**: better nothing than an encryption that is close but wrong.

4 · The preview is a list of decisions

The third step does not list warnings, it lists **decisions**. At the top sits the **estimate** — "I will import 72 devices · 50 cables · 7 VLANs · 24 racks · 4 stacks" — with **what will not come in** as chips beside it; then the rows, ordered by what they cost you.

Section	What it holds
To decide	The real choices, with the alternative and its consequence, and the **default** already applied. One row is *one decision*, never one device: thirteen switches with the same case are a single row. Rows whose default already answers well fold to one line; only the ones that **block creation** carry a tag saying so.
What will not come in	What stays out *because of how the model is built*: console ports and power feeds on anything that is not a PDU, power cables and WAN circuits. With the **count** and the **names**, not just a number.
No choice needed	Declared limits that lose nothing: VLANs that merge, models outside the catalogue.

What the DCIM declares is on screen. At the bottom of an imported device's **Properties** sits the **Declared by the DCIM** block: owner (*tenant*), status, role, platform. It is **read-only** on purpose — that is NetBox speaking — and shows **only** what the DCIM actually declared: a printed blank would read as data.

Touch nothing and you get what you would have got before: the defaults **are** the historic behaviour. Changing a decision and pressing **Recalculate** is instant, because the NetBox read is reused instead of repeated — which is why the panel says **what time** it was read, with **Read NetBox again** next to it.

Why names, not just numbers

A preview that says "58 cables not imported" leaves you wondering *which*. Every row opens the list of the devices involved: the decision is yours, but on data with a name on it.

Manual-first and non-destructive

The import always creates a **new project**: it never touches or overwrites an existing one. It writes nothing to NetBox — write-back is a separate operation, in the paid module, and even then additive only (create or update by natural key, **never delete**).

20 Hardware catalog, PDU and combo ports

Give every device its real ports — power outlets included — without inventing them.

A well-drawn device has **the ports it actually has**. InfraNet starts from a **NetBox-compatible hardware catalog**, treats **PDUs** as first-class citizens and never double-counts **combo ports**. Everything stays **manual-first**.

1 · Apply a model (NetBox-compatible catalog)

In **Properties** → **Ports** you pick the **real model** from a catalog of thousands of devices (NetBox's CC0 library). The device inherits its exact ports — **copper, SFP/SFP+, QSFP, management** — in the right number and order, instead of a generic count.

> **Vendor-neutral**
>
> The catalog describes ports in a **vendor-independent** language: applying a model invents nothing and reorders nothing. It is refreshed with one command (`npm run update-device-types`) without touching your projects.

Where the catalog comes from, and how to rebuild it

The catalog ships inside the application: there is nothing to download on first run. It derives from **NetBox's device-type library**, which is **CC0 — public domain**: no licence obligation travels with it. It is not a full copy but a **selection**: the devices InfraNet can draw stay in — network, power, NAS, access points, console servers — while accessories and peripherals (phones, printers, rack kits, blanking panels) stay out, because they do not belong in the document.

The snapshot of the source is recorded with its **exact commit**. That serves two purposes: knowing how old the catalog you are using is, and being able to **rebuild the very same one** a year from now, when the upstream library has moved on.

```
# update to the latest public library (needs git and network)
$ npm run update-device-types

# rebuild EXACTLY the version recorded in the manifest
$ npm run update-device-types -- --ref=<commit>

# offline, from a local checkout of the library
$ npm run update-device-types -- --input=C:\path\to\devicetype-library
```

Without the catalog the application still works: the **Apply a model** field simply does not appear, and you declare ports by hand as always. A control that cannot do anything is not left there pretending.

2 · PDU — the rack's power outlets

InfraNet draws the **PDU** with its **outlets** — up to **48** — inside the rack frame, which grows with the PDU's height (1U, 2U and beyond).

In **Properties** → **Power**, populated by the NetBox import or filled in by hand, you say outlet by outlet **which device** it powers and on **which inlet**. Outlets are not network ports: you don't run a data cable to them and they don't count towards spare ports.

Outlet groups: what stays on when the power goes

A power strip answers "what is fed". A **UPS** has to answer a different question, the only one it is bought for: **what stays on in the dark, and what can I sacrifice to last longer**. That is a property of the **group**, not of the single outlet, and in **Properties** → **Outlet groups** you declare it with two answers only: whether the group **can be switched on its own** or stays always on, and whether the **battery holds it** or it is merely surge-filtered. Each group takes a colour, which the rack repeats as a **thin band** above its outlets: the fill colour keeps saying the state. The groups reach the handover dossier too, next to every outlet.

When you apply a catalog model, the groups the maker already wrote into the outlet names ("Group 1 Outlet 3", "Primary Group", "Segment2-1") arrive **already filled in**. What the name does not say stays for you to declare: an outlet's position is not evidence. And your word is never overwritten — no update renames the groups you declared.

> **It is a declaration, not a measurement**
>
> The SNMP standard for UPSs (RFC 1628) **has no outlet groups at all**: every brand keeps them in its own private dictionary. That is why you write the group, and InfraNet does not pretend to have read it off the device.

Outlet state	Meaning
Active	Outlet with a device connected and powered.
Inactive	Free or switched-off outlet — the default of an undocumented one.
Faulty	Outlet flagged as failed.

> **Your word wins**
>
> If you import the PDU from NetBox the imported connections stay the **source**; your edits are **separate overrides**, **resettable** back to the imported value. The auxiliary ports — **Ethernet management, serial, sensor, USB, expansion** — are kept apart, and only the Ethernet management port acts as a network port.

3 · Combo ports

A **combo** port is one physical position shared between copper and an SFP slot, where you use **one at a time**: InfraNet models it as **one** position, not two.

Why it matters

Counting them as separate ports would inflate the real capacity. Treating them as shared slots keeps the drawing, the spare-port count and the topology aligned.

21 Sites and links (multi-site)

The floor above the project: the company's sites, their WAN lines and the links that hold them together.

A project documents **one building**. This is the floor *above* it: how the buildings talk to each other. It opens from the **Multi-site** button beside the project picker, and it has five tabs — **Map**, **Sites**, **WAN lines**, **Links**, **Coherence**.

> **Written by hand, and not as a fallback**
>
> No device is queried here: this is where you **declare**. The two-to-five-site company this layer is built for has no DCIM to draw from, and the day it does (the NetBox import, further on) it fills *these very fields* rather than different ones. Reading is open to everyone; only the **administrator** writes, because documenting is not administering.

> **One organisation per installation**
>
> The panel shows **the same sites whichever project is open**: the organisation belongs to the InfraNet *installation*, not to a single project. That is the model, not an oversight — there is one company, and a copy inside every project would be the same fact written in five places.

1 · The sites

Each site has a **name**, a **role** — *Hub*, *Spoke* or *Standalone*, and it is the role that decides the shape of the map — an **address**, its **networks** and, above all, the **project** that documents it. The project is **a reference, never a copy**: the site points at the building's document, which stays the only place where that building is described.

> **«Take the networks from the project»**
>
> Instead of retyping what the site's project already says, one button brings them up. It takes the **declared** networks — the rows of the Networks panel — and **never** the /24s inferred from device addresses: the first are a document, and copying a document into another one is honest; the second are a derivation, and promoting them to a company-level declaration would be inventing with the face of a choice. And it **only adds**, never replaces: a hand-typed row and one taken from the project yesterday look too alike for a mis-click to be allowed to delete somebody's work. It reports two numbers — how many it added and how many the project declares — because «added 0 of 4» and «added 0 of 0» are different answers.

2 · The WAN lines

One row per line leaving a building, with what you need **when it is down**: **provider** and **circuit ID** — the one you dictate on the phone — the **service type** (FTTH, FTTC, FWA, xDSL, SDSL, dedicated fibre, dark fibre, carrier ethernet, MPLS, mobile, satellite), the **committed rate**, the kind of **addressing** (static, DHCP, PPPoE), the **public addresses** and the **WAN interface** it lands on.

The committed rate is not the port speed

Two different things, alike enough to do damage: the port runs at 1 Gb/s, the contract guarantees 30 Mb/s. If you do not know the rate, the field stays **empty**: a number taken from the port would write down a guarantee nobody sold.

Public addresses are a list

Not a single one. A business line comes with a routed block, IPv6 rides the same line and an HA pair exposes several: a single field would be false by the next day.

«Read the WAN lines from the DCIM»

If the site was born from a **NetBox import** (chapter 19), one button reads the circuits terminating at that site and **adds** them — never replaces, for the same reason as the networks. What comes across is what NetBox can answer for: provider, circuit ID, service type and committed rate. Only **active circuits** are offered: a planned or decommissioned line, once imported, would be indistinguishable from one in service. Everything it could not place — a circuit going to a site this organisation does not hold, a line already there, a truncated read — it **tells you**, instead of dropping it.

3 · Links: two questions, not one

A link between two sites answers **two** different questions, and InfraNet keeps them in two separate fields: **Transport** («what it travels on») and **Tunnel** («what runs on top»).

Field	What it says	Values
Transport	What it travels on	Internet · MPLS · VPLS · VPWS · VXLAN · EVPN · Direct link · Other
Tunnel	What runs on top	None · IPsec · GRE · WireGuard · OpenVPN · L2TP · SD-WAN · Other

Why two fields and not one

An **IPsec inside an MPLS** is one single link, and it happens often. With one field you had to pick one of the two and the other disappeared from the document. With two axes you write down what is actually there — and «Internet + IPsec», «MPLS + None», «MPLS + IPsec» become three rows you can tell apart at a glance. Either axis may stay **unstated**: «I don't know» is already information, and with *Other* you can write what you call it yourself.

The link then carries what it takes to bring it back up:

- **Networks reachable at each site** — one declaration per end. On an IPsec link they *are* the encryption domain: bring the tunnel back up without them and nothing passes through it. Through a hub it is legitimate for a link to carry a third site's networks as well.
- **WAN lines carrying it** — true of every nature, and it is the recovery question: «the Milan fibre is down, what falls with it?». The panel only offers the lines of the two sites at the ends.

- **The device at each end** — picked from the devices of that site's project, or typed by hand: a carrier's CE is often not a documented node, and forcing a choice from a list would have made the commonest case undocumentable. What you pick shows as linked and says when it stops resolving; what you type stays a declaration.
- **Provider, circuit ID and name** — «GRE-LAB» is the name, «GRE» is the nature: different questions, different fields.
- **State** — *Up* or *Down*, with **who says so**. Until somebody says it, it stays «not stated», which is not the same thing as «it works».

4 · The map

Sites are **boxes** holding their own WAN lines, links are the lines between the boxes. Two links between the same two sites — the real case: an MPLS primary and an IPsec backup — **fan apart** instead of drawing one over the other.

> **The same organisation always draws the same map**
>
> The geometry is **deterministic**: no physics simulation settling on its own. A map that rearranges itself on every open cannot be compared with yesterday's or printed twice alike. The shape follows the **declared role**: a hub goes to the centre, and with zero hubs or two it falls back to a ring rather than picking one for you.

A site's box says **how much is inside it** — «42 devices · 1 rack» — so the weight of a site is visible from the floor above, and a site whose project is not linked yet reads as such at a glance. A **click on a site opens its project**: from inside, the path at the top reads *InfraNet Pro / organisation / project / view*, and the organisation's name is the way back up.

5 · Coherence

The last tab checks the document **against itself**, on the declared data alone, and keeps apart two things that are not alike at all:

- **Incoherences** — something is *wrong*: a link carrying a network no site claims, the same subnet declared at two sites, an end pointing at a site that does not exist, a spoke touching no hub, a link declaring a WAN line that belongs to neither of its two sites — a Turin line cannot carry a Milan-to-Rome link —, an address declared public that is not one.
- **Gaps** — nothing is broken, but a question cannot be answered: a line with no public address, a link that does not say which networks it reaches or which line it runs on.

> **«Could not be checked» is a third answer**
>
> Every check that could not run leaves **its own name and its reason**. An empty list must mean «I looked and there is nothing», never also «I did not look»: same screen, opposite situations.

The link that does not say which line it runs on

On the first real organisation, **five links out of eight** did not say — and nothing reported it. It is the gap that costs most, because it is exactly the question you ask on the night of the outage. A *direct link* between two of your own buildings is the one exception: there the emptiness is the right answer, because that link *is* the line.

The «public» address that is not public

A line's *Public addresses* field takes what you write, and there is one case where the wrong thing is also the most natural to paste: behind **CGNAT** the address the router shows on its WAN port is a `100.64.x.x`, and nobody reaches you there from outside. Same for a private address, a loopback, a `0.0.0.0`. The row carries the address *as you wrote it* and what it is — «100.64.8.2 · CGNAT» — because «this is not public» and «this is the carrier's NAT» send you to two different fields. **Documentation** addresses (203.0.113.x and friends) are not reported: they are exactly the ones the hint under the field offers you as an example.

6 · And on paper

All of this goes into the **WAN chapter** of the handover dossier (chapter 18): the map redrawn for print, and one **recovery card** per line and per link. In its header the chapter **counts its own holes**, instead of hiding them.

22 Bilingual IT / EN

Interface in Italian or English, with networking terms always kept in their original form.

The interface is **bilingual Italian/English**. The selector is in the user menu; the choice is remembered between sessions. You switch between languages without reloading and without losing your work.

> **The technical glossary is not translated**
>
> Networking terms — **VLAN, SNMP, LLDP/CDP, SFP, trunk, uplink** — and vendor names remain unchanged in both languages: translating them would only cause confusion. What is translated is the interface (menus, panels, messages), not the language of the trade.

Bilingual coverage is verified automatically: a check flags any string translated on only one side, so both languages stay aligned as the app grows.

Support the project

InfraNet Pro is **free and open source**. If it saves you time, you can buy the development a coffee: every contribution helps fund new features, fixes and support.

> ☕ **Buy me a coffee**
>
> Support page: **ko-fi.com/infranetpro**