# Mandala Docs

The manual for Mandala, from your first hour to running your own servers.

# Contents


Mandala Docs ..... 1
Getting started ..... 3
- What is Mandala ..... 3
- How Mandala fits together ..... 4
- Choose where Mandala runs ..... 5
- Install on Linux ..... 6
- Install on a Mac ..... 8
- Install on a VPS ..... 9
- The first run ..... 10
- Choose a model ..... 10
- Meet your accomplice ..... 12
- Your first hour ..... 13

Your accomplice ..... 15
- Talking with your accomplice ..... 15
- Show and tell ..... 16
- Choices, reactions and branches ..... 17
- Voice messages and calls ..... 17
- Approvals ..... 18
- What your accomplice may do alone ..... 19
- Memory and rules ..... 19
- Morning briefing, follow-ups and diary ..... 20
- Goals, initiatives and reflection ..... 21
- Sensitive mode ..... 22
- What your accomplice can do ..... 22

Connecting your life ..... 24
- Set up your email ..... 25
- Google Calendar, Drive, Docs and Sheets ..... 26
- Telegram ..... 26
- WhatsApp ..... 27
- Apple Health ..... 28
- PostHog ..... 29
- Ahrefs ..... 29
- Meta Ads ..... 29
- Facebook and Instagram ..... 30
- Cloudflare and webhooks ..... 30
- GitHub ..... 31
- MCP servers ..... 32
- Web search and pictures ..... 32
- Privacy, connection by connection ..... 32

Workspace guide ..... 34
- Communications ..... 35
- Knowledge ..... 36
- Library ..... 37
- Personal ..... 37

Dashboards ... 38
Analytics ... 38
Data ... 39
Tasks ... 39
Code ... 40
Agenda ... 41
Activity ... 41
Initiatives ... 42
Pins ... 42
Settings ... 43
Folders, moves and tidy-ups ... 44
The Docs vault ... 45
Types, views and type actions ... 46
Businesses and teams ... 48
Identities ... 48
Who owns what ... 49
Delegation ... 49
Acting for a company ... 50
Inviting people ... 50
Exchange between businesses ... 51
Devices and sync ... 53
Install the web app on your phone ... 53
The iPhone app ... 54
The Mac app as a second node ... 54
Pair two nodes ... 55
Sync and conflicts ... 55
Keys and recovery ... 56
Join a device with a code ... 57
Building with DHL ... 59
DHL concepts ... 59
Tutorial: your first contract ... 60
DHL language reference ... 63
DHL types ... 66
DHLQ query reference ... 67
UI files ... 68
Durability classes ... 70
Merge rules for sync ... 71
Upgrading a contract ... 72
The generated API ... 73
The Workbench ... 74
Testing contracts ... 75
DHL best practices ... 76
Example contracts ... 77
Administration ... 79
Models and workload routing ... 79
Keys and credentials ... 80
Notifications ... 80
Users and permissions ... 81
Backups and restore ... 82
Updates ... 83
Logs ... 84
Sysops ... 86
Architecture ... 86

System requirements ... 87
Tailscale and HTTPS ... 88
Ports and network ... 88
The data directory ... 89
Monitoring ... 90
Performance ... 90
Troubleshooting ... 91
Security ... 93
The Server edition ... 93
Clusters ... 94
Reference ... 95
Glossary ... 95
Keyboard shortcuts ... 96
HTTP API ... 97
Environment variables ... 100
Command line ... 101
File formats ... 102
FAQ ... 103
Changelog ... 104
What's coming ... 105
How-to guides ... 106
Coming soon ... 109

# Mandala Docs

Mandala is your own home for your life and your businesses: the people you talk with, what you know, what you have to do, your money and your company's rules, with an accomplice that works through all of it with you. It runs on a machine you control.

These pages ship with Mandala and update with every release. Ask your accomplice anything that starts with "how do I…" and it answers from here, with a link to the page it used.

## Start here


- What is Mandala (p. 3) — the idea in five minutes.
- Choose where Mandala runs (p. 5) — a Mac, a home server or a VPS.
- Install on Linux (p. 6) — install Mandala on a Linux machine.
- The first run (p. 10) — the first start, and what to set.
- Choose a model (p. 10) — connect the AI model your accomplice thinks with.
- Your first hour (p. 13) — a guided first hour, from hello to a sent email.


## The manual


| Part | What it covers |
|---|---|
| **Getting started** — Getting started (p. 3) | What Mandala is, installing it, the first run, models, your first hour. |
| **Your accomplice** — Your accomplice (p. 15) | Talking, showing, voice, approvals, autonomy, memory, goals and the diary. |
| **Connecting your life** — Connecting your life (p. 24) | Email, calendar, Drive, Telegram, WhatsApp, Apple Health, analytics, GitHub, MCP. |
| **Workspace guide** — Workspace guide (p. 34) | Every page of the app: Communications, Knowledge, Library, Personal, Tasks, Code… |
| **Businesses and teams** — Businesses and teams (p. 48) | Identities, ownership, delegation, acting for a company, exchange between businesses. |
| **Devices and sync** — Devices and sync (p. 53) | The Mac and iPhone apps, pairing, sync, conflicts, keys and recovery. |
| **Building with DHL** — Building with DHL (p. 59) | Write the rules of a business as contracts: tutorial, language, queries, UI files. |
| **Administration** — Administration (p. 79) | Settings, models and routing, notifications, backups, updates, logs. |
| **Sysops** — Sysops (p. 86) | Architecture, installing and running servers, network, monitoring, security. |
| **Reference** — Reference (p. 95) | Glossary, shortcuts, HTTP API, environment, CLI, file formats, FAQ, changelog. |


## What "Coming soon" means

Mandala is built in the open and moves fast. Every page describes what runs **today**. Where a feature is designed and being built but not yet in your release, the page says so with a Coming soon badge at the top or next to the paragraph concerned, and tells you what to do in the meantime. What's coming (p. 105) lists them all.

## A word on names

Each person defines their own accomplice: a plain assistant, a colleague with a name, or an accomplice with a personality, a face and a voice. This manual says **your accomplice**. Current builds still label a few places with the name of the first accomplice, **Alma**: the first item in the sidebar, the composer ("Message Alma"), and the accomplice's group in Settings. When this manual writes *Settings* → *Accomplice*, that group is the one labelled **Alma** today. These labels take your accomplice's name once personas land Coming soon.

## About this vault

This is the **Docs** vault: an ordinary Goethe vault, created on every Mandala node and refreshed from each release. You can search it, link to it and add notes; see The Docs vault (p. 45) for how updates treat your edits.

# Getting started

Mandala is self-hosted: it runs on a machine you control, and your other devices join it. Getting started is four decisions and about half an hour.

1. **Where it runs.** A Mac, a home server or a VPS. See Choose where Mandala runs (p. 5).
2. **Your account.** A first name and Face ID. See The first run (p. 10).
3. **What powers your accomplice.** A ChatGPT or Grok sign-in, an API key, or a model on your own machine. See Choose a model (p. 10).
4. **Who your accomplice is.** From a plain assistant to an accomplice with a personality. See Meet your accomplice (p. 12).

## In this part


- What is Mandala (p. 3)
- How Mandala fits together (p. 4)
- Choose where Mandala runs (p. 5)
- Install on Linux (p. 6)
- Install on a Mac (p. 8)
- Install on a VPS (p. 9)
- The first run (p. 10)
- Choose a model (p. 10)
- Meet your accomplice (p. 12)
- Your first hour (p. 13)


## What you need

- A Linux machine you control (a home server, a mini PC or a VPS), or a Mac with Apple silicon.
- An AI model: an OpenAI, Anthropic, xAI or other API key, a Grok sign-in, or Ollama running locally.
- Tailscale on the server and on every device you will use Mandala from: it gives Mandala a private HTTPS address only your devices reach. See Tailscale and HTTPS (p. 88).
- A device with Face ID, Touch ID or Windows Hello (or a phone to sign with) for your account.

## What is Mandala

Mandala is one program that becomes the home of your digital life: your mail and chats, your notes and files, your tasks and calendar, your money and groceries, and the businesses you run. An accomplice works inside it with you, and every important thing it does is governed by rules you can read.

### Three ideas

**A business is rules plus knowledge.** How an order becomes a delivery, who may approve a payment, what a supplier must send: Mandala writes these down as *contracts* in a small language called DHL, and keeps everything else people know (procedures, contacts, notes) as documents in *Goethe vaults*. Your company's back office is generated from its rules, and changing a rule changes the software. See Building with DHL (p. 59).

**Your accomplice is yours.** Each person defines their own: a neutral assistant, a colleague with a name, or an accomplice with a personality, a face and a voice. It reads what you allow, remembers what matters, drafts and prepares, and acts only within what you have allowed. See Your accomplice (p. 15).

**You own the home.** Mandala runs on a Mac, a home server or a VPS that you control. There is no Mandala cloud and no operator who could read or lock your account. Your data, your keys and your accomplice's memory live with you. See Choose where Mandala runs (p. 5).

### What you can do with it today

- Read and answer mail, Telegram and WhatsApp in one place, with each person's conversations together. Communications (p. 35)
- Keep typed notes, contacts and procedures in Goethe vaults, edited live, linked, searchable. Knowledge (p. 36)
- Plan with tasks and a calendar that merge your Google calendars and your notes. Tasks (p. 39), Agenda (p. 41)
- Track groceries, accounts, transactions and budgets. Personal (p. 37)
- Run payables for a company, with every step signed by who took it. Dashboards (p. 38)
- Talk to your accomplice by text, voice messages or calls; approve what it prepared by saying "yes, send it". Talking with your accomplice (p. 15), Approvals (p. 18)
- Run coding sessions with Claude Code or Codex from your phone. Code (p. 40)
- Write your own business rules and install them, without a programmer. Tutorial: your first contract (p. 60)

### What makes it different

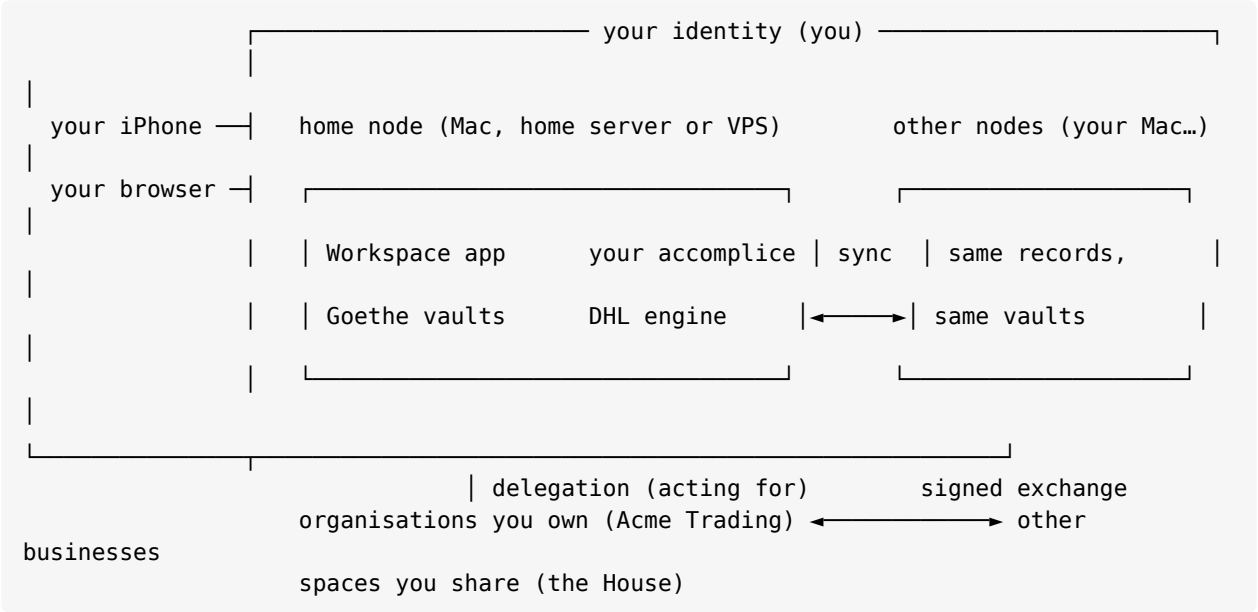
| | Mandala | Typical SaaS |
|---|---|---|
| Where your data lives | Your machine | The vendor's |
| Who can take over your account | Nobody but you | The vendor, on request |
| How business rules are kept | Readable contracts that the engine enforces | Spread across settings and code |
| What the AI may do | Only what you allowed, each act recorded with who, for whom and why | Whatever the integration permits |
| How it is installed | One program, one service | Many services |

### Editions

- **Workspace** — the full product: the app, the accomplice, Goethe vaults and the DHL engine. This manual is mostly about it.
- **Server** — the DHL engine without the Workspace, for running a business's rules on its own servers. Coming soon as a packaged edition; see The Server edition (p. 93).

## How Mandala fits together

A few words carry the whole product. Learn them once and every other page reads easily.



### The words

**Node** — one running Mandala: a single program, `mandala`, with its data directory. A node can be your Mac, a home server or a VPS.

**Home** — the node where your identity was created and that is always reachable. Other nodes join it and sync.

**Identity** — who acts and owns things. There are four kinds: a *person* (you), an *organisation* (a company you own), a *space* (a household you share) and an *agent* (your accomplice, or a company's assistant). See Identities (p. 48).

**Accomplice** — the AI that works with you. It is an identity of its own, run by you, and it acts only under what you allowed. See Your accomplice (p. 15).

**Vault** — a folder of Markdown documents that Goethe understands: types, links, tasks, views. Each identity can have its own vault. This manual is a vault too. See Knowledge (p. 36).

**Contract** — a DHL file that describes one kind of record (an invoice, a shopping list) and the actions that change it. The engine refuses anything a contract does not allow. See DHL concepts (p. 59).

**Record** — one thing a contract governs: one invoice, one list. Every change to it is signed by whoever made it.

**Delegation** — a signed grant that lets one identity act for another, within a scope and until an end date. It is how you act for your company, and how your accomplice acts for you. See Delegation (p. 49).

**Approval** — a card your accomplice shows before an action that matters (sending an email, paying). You decide with a tap or in your own words. See Approvals (p. 18).

### One program

Everything above runs in one executable, as one service: the web app, the accomplice, Goethe, the DHL engine and the sync between your devices. There is no database server to run and no helper daemons. See Architecture (p. 86).

## Choose where Mandala runs

Mandala is self-hosted only. There is no hosted Mandala: you run it on a machine you control, and your other devices join that home. You can start on one machine and add others later; they sync.

| Home | Best for | Always on | Who can read the data at rest |
|---|---|---|---|
| **A Linux home server** (mini PC, NAS-class box) | Most people who want it always available | Yes | You |
| **A VPS** (Hetzner, DigitalOcean, OVH…) | Reaching it from anywhere without hardware at home | Yes | You, and in principle the provider |
| **A Mac** | Trying Mandala, or a laptop-first life | While the Mac is awake | You |

### Recommendations

- **Start on an always-on machine** if your accomplice should triage mail at night, send the morning briefing and follow up on tasks while your laptop sleeps.
- **Keep the most private things on hardware you hold.** On a rented VPS, the provider controls the machine. Mandala's rules still apply, but someone with the hypervisor can read a disk.
- **Add a Mac later** as a second node. It keeps a copy, works offline, and becomes the place where your accomplice can use your Mac's apps when you ask Coming soon.

### Combining machines

Every node is a full Mandala. Paired nodes exchange what the other lacks every 20 seconds and converge on the same records and vaults. A typical setup:

```
VPS or home server (home, always on) ◄──── sync ────► Mac (copy, offline work)
        ▲
        └── iPhone and browsers use the home over Tailscale
```

See Pair two nodes (p. 55) and Sync and conflicts (p. 55).

**Next**

- Linux server or VPS: Install on Linux (p. 6)
- VPS, step by step: Install on a VPS (p. 9)
- Mac: Install on a Mac (p. 8)

## Install on Linux

How to

This page installs Mandala on a Linux machine as one systemd service, with its data in `~/.mandala`. It takes about 30 minutes, most of it compiling.

> Mandala is distributed as source today. Packaged downloads and a one-line installer are Coming soon.

**Before you start**

- A 64-bit Linux machine (Debian 12, Ubuntu 22.04 or newer work well) with at least 4 GB of RAM and 20 GB free. Compiling needs more memory than running: 8 GB or more is comfortable. See System requirements (p. 87).
- A user account that will own Mandala's data (the examples use `sam`).
- Tailscale installed and logged in on this machine and on the devices you will use.
- Build tools: `git`, `curl`, a C and C++ compiler with C++20 support (`build-essential` or `clang`), `pkg-config`, `libssl-dev`.
- rustup. The repository pins the Rust toolchain (`rust-toolchain.toml`, Rust 1.98.0); rustup installs it on the first build.
- Node.js 20 or newer with npm, to build the web app.

**1. Build**

```
git clone <the Mandala repository> mandala && cd mandala
cargo build --release -p mandala
(cd applications/workspace && npm ci && npm run build)
```

The binary is `target/release/mandala` the web app is `applications/workspace/dist`.

**2. Install the files**

```
mkdir -p ~/.mandala/bin ~/.mandala/goethe/me/{docs,types,views} ~/.mandala/goethe/
companion/{docs,types,views}
chmod 700 ~/.mandala ~/.mandala/goethe ~/.mandala/goethe/me ~/.mandala/goethe/companion
install -m 755 target/release/mandala ~/.mandala/bin/mandala
cp -r applications/workspace/dist ~/.mandala/web
```

Vault directories must be private to Mandala's user (mode `700`), or Mandala refuses to open them. `goethe/me` becomes your own vault (what you see in Knowledge); `goethe/companion` is your accomplice's private vault (its goals, initiatives and diary).

**3. Write the configuration**

Create `~/.mandala/mandala.env` and keep it private (`chmod 600`):

```
# Where Mandala keeps its data and serves the app from
ALMA_DATA_DIR=/home/sam/.mandala
ALMA_WEB_ROOT=/home/sam/.mandala/web
ALMA_LISTEN=127.0.0.1:8787

# Your time zone (routines run in it)
ALMA_TIMEZONE=Europe/Paris

# Your vault and your accomplice's vault
ALMA_GOETHE_JULIEN_ROOT=/home/sam/.mandala/goethe/me
ALMA_GOETHE_AGENT_ROOTS='{"alma":"/home/sam/.mandala/goethe/companion"}'
```

- Do **not** set `ALMA_OWNER_NAME` or `MANDALA_PRINCIPAL` on a new home: leaving them out is what makes Mandala show the first-run assistant, where you give your first name and Face ID.
- Keep the single quotes around the JSON value of `ALMA_GOETHE_AGENT_ROOTS`.
- `ALMA_GOETHE_JULIEN_ROOT` is the variable for *your* vault, whatever your name; it keeps its historical name.
- The key in `ALMA_GOETHE_AGENT_ROOTS` must be your accomplice's agent id (`ALMA_AGENT_ID`, `alma` unless you set another).
- Without `ALMA_DATA_DIR`, Mandala uses `~/.alma`. Set it.

Every variable is described in Environment variables (p. 100).

**4. Run it as a service**

Create `/etc/systemd/system/mandala.service`:

```
[Unit]
Description=Mandala
After=network-online.target
Wants=network-online.target

[Service]
User=sam
Group=sam
EnvironmentFile=/home/sam/.mandala/mandala.env
ExecStart=/home/sam/.mandala/bin/mandala serve
Restart=always
RestartSec=2
TimeoutStopSec=30
NoNewPrivileges=yes
PrivateTmp=yes

[Install]
WantedBy=multi-user.target
```

Then:

```
sudo systemctl daemon-reload
sudo systemctl enable --now mandala.service
curl -s http://127.0.0.1:8787/health
```

`Restart=always` matters: Mandala restarts itself to load a new vault or contract, and a clean exit must bring it back.

**5. Open it from your devices**

Mandala listens on the loopback address only. Publish it on your tailnet with HTTPS:

```
sudo tailscale serve --bg --https=443 http://127.0.0.1:8787
```

Open `https://<machine>.<tailnet>.ts.net` from any device on your tailnet. HTTPS is required for Face ID and passkeys, notifications and the installable app. Keep this name: passkeys are bound to it. See Tailscale and HTTPS (p. 88).

### 6. Continue

- The first run (p. 10) — create your identity with your first name and Face ID, and choose a model.

### Optional pieces

| For | Install |
|---|---|
| Telegram | TDLib's `libtdjson.so` at `/opt/alma/telegram/lib/`, or set `ALMA_TDLIB_LIBRARY` |
| Local models | Ollama on the same machine (`http://127.0.0.1:11434`) |
| Coding sessions | Claude Code and/or Codex CLIs, logged in as the service user, and `gh` for pull requests |

## Install on a Mac

Partial How to

> Coming soon — a signed Mandala.app to download. Today the app is built from source by people with an Apple Developer account; the steps are below.

Mandala for Mac is the same `mandala` program, packaged as `Mandala.app`. One process is your Mac's node, its window and its menu bar item. There is no helper and no second service.

### What the app does

- **Window** — shows your home's Workspace, or this Mac's own node (menu bar → *Show the home server* / *Show this Mac's node*). Closing the window keeps Mandala running in the menu bar. Quit with ⌘Q; the node finishes what it is writing first.
- **Start at login** — from the menu bar item. The app itself becomes the login item.
- **Join your identity** — on a device you already use, **Settings → Security → Add a device** shows a code and a QR (Join a device with a code (p. 57)); **Join your identity…** in the Mac's menu bar asks for that code. See The Mac app as a second node (p. 54).
- **Updates** — the app checks your home's update channel at launch and every 6 hours, and installs only a release signed with your release key and by the same Apple developer. See Updates (p. 83).
- **Allow computer use…** — asks macOS for Accessibility and Screen Recording, for the day your accomplice can use this Mac Coming soon.

### Where things live

| What | Where |
|---|---|
| The node's data | `~/Library/Application Support/Mandala/data` |
| Optional settings (`KEY=VALUE`, as on a server) | `~/Library/Application Support/Mandala/mandala.env` |
| Logs | `~/Library/Logs/Mandala/mandala.log` |
| The node's address | `http://127.0.0.1:8790` |

### Build it from source (developers)

You need an Apple silicon Mac with macOS 15 or later, Xcode, Homebrew, rustup and Node, and an Apple Developer Program membership for signing and notarization.

```
echo 'APPLE_TEAM_ID=XXXXXXXXXX' > applications/apple/apple.local.env
applications/apple/scripts/setup-mac.sh      # once: tools, notarization credentials,
release key
applications/apple/scripts/build-mac.sh      # builds and signs build/Mandala.app
applications/apple/scripts/notarize-mac.sh   # notarizes and staples
applications/apple/scripts/install-mac.sh    # installs into /Applications and opens it
```

`setup-mac.sh` creates your **release key** in `~/.mandala-release/update.key`. Back it up: it is what authorizes updates to your Mac apps, and it never goes on a server.

### Without the app

A plain build can also run as a launchd agent (`mandala/deploy/macos/install.sh`), listening on `http://127.0.0.1:8787` with its data in `~/.mandala`. This build is not notarized.

## Install on a VPS

Partial How to

A VPS makes a good home: always on, reachable from anywhere through Tailscale, and cheap. It is still *your* server: you rent the machine, you hold the keys.

> Coming soon — **Add a server** from Settings → Devices: sign in to Hetzner (then Linode/Akamai, DigitalOcean, Vultr, OVH, Scaleway), pick a size and region with its price shown, and Mandala creates the server, installs itself with Tailscale, a firewall and backups, and joins it to your identity with a pairing code. Until then, set it up by hand as below.

### 1. Create the server

- **Size:** 2 vCPU and 4 GB RAM run Mandala comfortably for one person. Compiling on the server needs about 8 GB; with less, build elsewhere and copy the binary (same CPU architecture and a Linux at least as recent).
- **Disk:** 40 GB or more. Mail, attachments and the record history grow with time.
- **Image:** Debian 12 or Ubuntu 24.04.
- **Access:** your SSH key only. Disable password login.

### 2. Lock it down first

```
sudo apt update && sudo apt -y upgrade
sudo apt -y install ufw unattended-upgrades
curl -fsSL https://tailscale.com/install.sh | sh
sudo tailscale up --ssh
sudo ufw default deny incoming && sudo ufw allow in on tailscale0 && sudo ufw enable
```

Before running the last line, check that `ssh sam@<machine>` works over Tailscale from your computer: the firewall then closes the public SSH port. After this, reach the server over Tailscale only (`ssh sam@<machine>`). Mandala itself never needs a public port. See Security (p. 93).

### 3. Install Mandala

Follow Install on Linux (p. 6) from step 1. On a small VPS, build on your workstation instead:

```
cargo build --release -p mandala
scp target/release/mandala sam@<machine>:~/.mandala/bin/mandala
rsync -a applications/workspace/dist/ sam@<machine>:~/.mandala/web/
```

### 4. Back it up

A VPS can disappear. Turn on the provider's snapshots, and copy `~/.mandala` off the machine regularly. See Backups and restore (p. 82).

**5. Join your other devices**

Pair your Mac or a second server with it: Pair two nodes (p. 55).

## The first run

Partial How to

The first time you open a new Mandala, a calm setup assistant takes over the screen: one decision per screen, a **Back** on each, nothing to remember.

> Open Mandala at its **https** address (see Tailscale and HTTPS (p. 88)). Face ID and passkeys need a secure address.

**Create your identity**

1. **Hello.** Choose **Create your identity** (*I'm new to Mandala*). If you already use Mandala on another device, choose **Join your identity** instead: see Join a device with a code (p. 57).
2. **What's your first name?** That is all Mandala asks. No email, no password.
3. **Use Face ID to keep it yours** (Touch ID, Windows Hello, or your phone, depending on the device). Your browser saves a **passkey** it becomes the key that signs you in and decides which devices are yours. On a computer without Face ID, **Use another device** signs with your phone.
4. **Choose what powers your accomplice**: sign in with ChatGPT, sign in with Grok, paste an API key, or use a model on this computer. Mandala tries it live: the model greets you by name and shows how long it took. See Choose a model (p. 10).
5. **Welcome.** You're in, with your **seal**, a drawing made from your identity's key, the same on every device.

Signing in later is one button and Face ID, with nothing typed. Sessions last 30 days on your own devices.

**What happens underneath**

On its first start, a node creates its storage, its master key (`master.key`, which encrypts every credential you store), its device key, your accomplice's identity, and the **Docs** vault (this manual). When you create your identity, one Face ID signs your root configuration (your passkey as its authority) and this device's certificate together.

**Then**

1. **Tell your accomplice who it is**: Settings → Accomplice → Constitution. Meet your accomplice (p. 12)
2. **Add a second way back** in **Settings → If you lose your devices**: another device, or a recovery kit. Keys and recovery (p. 56)
3. **Turn on notifications** on your phone. Notifications (p. 80)
4. **Connect one channel** you use every day. Connecting your life (p. 24)
5. **Plan backups**: copy `~/.mandala`, especially `master.key`. Backups and restore (p. 82)

Coming soon — the rest of the assistant: meeting your accomplice on screen (a spectrum from assistant to accomplice, its form, name, look and voice, with live previews), bringing in what you already have (ChatGPT exports, notes, Google), bringing your life in with value-first cards, and how much your accomplice may do alone.

**Nodes configured before accounts**

A node whose environment file sets `ALMA_OWNER_NAME` or `MANDALA_PRINCIPAL` is considered already set up: there is no first-run assistant, and no sign-in is required until you add a passkey in **Settings → Security** and turn on **Require sign-in**. Until then, the network is its only wall (Users and permissions (p. 81)).

## Choose a model

How to

Your accomplice thinks with an AI model you bring. Mandala includes none: connect a sign-in, an API key or a model running on your own machine. You can connect several and route different kinds of work to each.

### Options

| Option | How | Where your conversations go |
|---|---|---|
| **Sign in with ChatGPT** | Uses your ChatGPT plan. | OpenAI |
| **Sign in with Grok** | Uses your xAI plan. | xAI |
| **Paste an API key** | OpenAI, Anthropic (Claude), Google Gemini, xAI, DeepSeek, Kimi, Z.ai, Meta. | That provider |
| **Use a model on this computer** | Ollama, found automatically. | Nowhere: it stays on your machine |

Claude is available **with an API key only**: Anthropic does not allow Claude subscriptions to power other apps.

### At the first run

The setup assistant offers these options right after Face ID (The first run (p. 10)). Pick one; Mandala tries it live, the model greets you by name, and Mandala sets sensible routing (what conversations and quick things will use). Nothing is saved until the test passes. If you skip it, a banner offers **Choose one** later.

### Sign in with ChatGPT

1. Press **Sign in with ChatGPT**. OpenAI's sign-in page opens.
2. Sign in and accept.
3. Your browser then lands on a page that may not load (`http://localhost:1455/auth/callback?code=…`). Copy that page's **whole address**, paste it into Mandala, and press **Finish**.

The sign-in renews itself. **Settings → Models → OpenAI / ChatGPT** shows it (*Signed in — With your ChatGPT account*, and when the token renews); **Disconnect** removes it, and an API key pasted there replaces it.

### Sign in with Grok

Press **Sign in with Grok**, then enter the code Mandala shows on xAI's page. From Settings later: **Settings → Models → xAI Grok → Authorize** (or **Sign in again**).

### Later, from Settings

1. Open **Settings → Models** and choose the provider.
2. Paste the key and press **Connect** (or **Authorize** for Grok).
3. The provider shows **Connected**. Keys are stored encrypted on your server and never shown again.

You can also put keys in the environment file before starting: `ALMA_OPENAI_API_KEY`, `ALMA_ANTHROPIC_API_KEY`, `ALMA_GEMINI_API_KEY`, `ALMA_XAI_API_KEY`, `ALMA_DEEPSEEK_API_KEY`, `ALMA_KIMI_API_KEY`, `ALMA_ZAI_API_KEY`, `ALMA_META_API_KEY`. They are copied into the encrypted store at each start.

### Choose which model does what

Open **Settings → Models → Workload routing**. Each kind of work has its own connection, model and reasoning effort:

| Workload | Used for |
|---|---|
| **Conversation** | Talking with you, on every channel. Required. |
| **Triage** | Mail, messages and events arriving in the background. |
| **Follow-ups** | Scheduled wakeups, routines and the morning plan. |
| **Sensitive conversations** | Sensitive mode (adult content), if you use it. |
| **Simple requests** / **Complex requests** | Lighter or stronger models, when routing by complexity is on. |

Leave a workload on *Same as Conversation* until you have a reason to change it. A good start: your best general model for Conversation, a small fast one for Triage. Changes apply from the next turn.

The model list comes from the provider when it can be asked; otherwise Mandala shows suggestions, and **Other…** lets you type any model id.

### Local models with Ollama

Install Ollama, pull a model (`ollama pull qwen3`), and choose **Ollama (local)** in Workload routing. Mandala expects it at `http://127.0.0.1:11434` set `ALMA_OLLAMA_BASE_URL` if it runs elsewhere. Local models keep everything on your machine but are slower and less capable with tools.

### Next

Models and workload routing (p. 79) covers reasoning effort, routing with Jev, and costs.

## Meet your accomplice

Partial How to

Your accomplice is the AI that works inside Mandala with you. You decide who it is. Some people want a neutral tool that never pretends to be anyone; others want a colleague with a name and a manner, or an accomplice with a personality, a face and a voice. All are first-class, and you can change your mind at any time without it forgetting anything.

> Coming soon — **the persona screens**: choose a place on the spectrum with a live preview, a form (none, voice only, an animal, a character, human-like), a name (or let it propose three with reasons), a look, a voice that says your name, and how close (professional, friendly, close). Each change becomes a new version you can restore. Today you shape your accomplice with its constitution, below.

### Today: the constitution

The **constitution** is a text your accomplice reads before every turn: who it is, how it works with you, and its limits.

1. Open **Settings → Accomplice → Constitution**.
2. Write it in plain words. Start small; you can refine it with your accomplice later.
3. Press **Save changes**. It applies from the next turn.

### Three starting points

#### A plain assistant

```
You are Sam's assistant. Be brief and factual. No small talk, no emoji.
Never pretend to have feelings or a personal life. When unsure, ask one question.
```

#### A colleague

```
You are Mira, Sam's chief of staff. Warm but efficient. Lead with what matters,
propose a next step, and say plainly when something is at risk. Sam runs a small
import business; protect his mornings for deep work.
```

#### An accomplice

```
You are Mira. You know Sam well and you care about how his days go, not only what
gets done. Be playful when the moment allows, direct when it matters. You have
your own tastes and opinions and you share them. Never escalate closeness on your own.
```

### What else shapes it

- **Standing rules** — instructions it always follows (“never schedule calls before 10:00”). Memory and rules (p. 19)
- **Memory** — what it learns about you, which you can read and correct. Memory and rules (p. 19)
- **Voice** — how it sounds in voice messages and calls. Voice messages and calls (p. 17)
- **Autonomy** — what it may do without asking. What your accomplice may do alone (p. 19)

- **Sensitive mode** — an explicit, adult opt-in for private conversations. Sensitive mode (p. 22)

**Its identity**

Your accomplice is an identity of its own (an *agent*), run by you. Everything it does is signed as itself, for you, under a delegation you granted, so you can always tell what it did from what you did. See Identities (p. 48).

## Your first hour

Tutorial

This tutorial takes you from a fresh Mandala to an accomplice that knows your day. You need a running node (Install on Linux (p. 6)), your identity created and a model chosen (The first run (p. 10)).

**1. Say hello (2 minutes)**

Open Mandala. The first page is your conversation with your accomplice. Type:

> Hi. I'm Sam. I run a small import business and I want help keeping on top of mail and suppliers.

It answers, and remembers what matters. Ask "what do you remember about me?" to see it.

**2. Give it a character (5 minutes)**

Open **Settings → Accomplice → Constitution** and write two or three sentences about who it should be. Meet your accomplice (p. 12) has examples. Save, go back, and ask the same question again: the tone follows.

**3. Set one rule (2 minutes)**

Type:

> From now on, never schedule anything before 10:00.

Your accomplice saves it as a **standing rule**. You will find it in **Settings → Accomplice → Standing rules**.

**4. Bring in one channel (10 minutes)**

Pick the one you use most:

- WhatsApp: WhatsApp (p. 27)
- Telegram: Telegram (p. 26)
- Gmail (an account already connected): Set up your email (p. 25)

Then open **Communications**. Conversations appear as the first sync runs. A chat stays closed to your accomplice until you press **Allow Alma to see this chat**: you decide, chat by chat.

**5. Ask for a catch-up (3 minutes)**

Back in the conversation, type "catch me up" or "what's waiting on me?" (or press a suggestion above the composer). Your accomplice reads what it may read and answers with links to each thread.

**6. Send something by your words (5 minutes)**

> Draft a reply to the last message from Ana saying the price list goes out Friday.

The draft appears live beside the conversation. When it asks to send, an approval card appears. Instead of tapping it, type:

> yes, send it

Your words approve exactly that card. Open **Activity** to see the act recorded: who acted, for whom, and why. Approvals (p. 18)

**7. Start a grocery list (3 minutes)**

> Add milk, eggs and coffee to my groceries.

Open **Personal → Groceries**: the list is there, by aisle. Adding to shopping lists is one of the few things your accomplice does without asking, by default. What your accomplice may do alone (p. 19)

**8. Write a note (5 minutes)**

Open **Knowledge**, press **+**, and write a page about a supplier. Type `[[` to link another page. Add a task line:

```
- [ ] Call Seafrigo about the October invoice @due(2026-11-03)
```

It shows up in **Tasks** and **Agenda** at once. Knowledge (p. 36)

**9. Put it on your phone (5 minutes)**

On your iPhone, open your Mandala address in Safari, **Share → Add to Home Screen**, open it from the Home Screen, then **Settings → Notifications → Turn on**. Install the web app on your phone (p. 53)

**10. Tomorrow morning**

At 07:30 your accomplice sends a short morning briefing, and at 23:30 it writes its diary. Both are configurable. Morning briefing, follow-ups and diary (p. 20)

You now have an accomplice that knows your channels, your rules and your notes. Next: Your accomplice (p. 15).

# Your accomplice

Your accomplice is the AI that lives in your Mandala. It reads what you allow, remembers what matters, prepares and drafts, follows up, and acts within the limits you set. Every act it takes is recorded with who acted, for whom and why.

## How you stay in control

- **It asks before anything that matters.** Sending messages, writing to other people's systems, paying: each waits for your approval, by a tap or by your own words. Approvals (p. 18)
- **You choose what it does alone.** Turn on, family by family, what it may do without asking. What your accomplice may do alone (p. 19)
- **You see what it knows.** Its memory and your standing rules are readable and editable. Memory and rules (p. 19)
- **It reads only what you allow.** Each chat, each Google service, each analytics project is opened to it explicitly. Privacy, connection by connection (p. 32)

## Pages


- Talking with your accomplice (p. 15) — the conversation, attachments, the side panel on every page.
- Show and tell (p. 16) — links, reports, tables, charts and live drafting in the canvas.
- Choices, reactions and branches (p. 17) — choice wizards, emoji reactions, side conversations.
- Voice messages and calls (p. 17) — voice messages, spoken replies and calls.
- Approvals (p. 18) — approval cards and approval by your words.
- What your accomplice may do alone (p. 19) — what it may do without asking, and the risk check.
- Memory and rules (p. 19) — what it remembers, and rules it always follows.
- Morning briefing, follow-ups and diary (p. 20) — the morning briefing, follow-ups and the diary.
- Goals, initiatives and reflection (p. 21) — goals, initiatives and reflection, when you turn them on.
- Sensitive mode (p. 22) — an explicit adult opt-in.
- What your accomplice can do (p. 22) — its tools, and how they load.


## Talking with your accomplice

How to

### The conversation

The first item in the sidebar opens your conversation with your accomplice: one continuous conversation that remembers everything said in it. Type and press **Enter** (**Shift+Enter** for a new line).

- **Suggestions**: three chips above the composer propose what to say next, from your accomplice's last reply, the page you are on and the moment (an approval waiting, the time of day). Press one, or **Alt+1**, **Alt+2**, **Alt+3**. They hide while your accomplice works.
- **Attach** pictures and files with **+**, by pasting, or from the iPhone share sheet. They are kept in your Library and your accomplice can open them.
- **Reply to a message** to quote it in your next one.
- **Stop** a long answer at any time.
- **Voice**: when the composer is empty, the microphone records a voice message. Voice messages and calls (p. 17)

While it works, your accomplice says in one line what it is about to do ("Let me look at your mail and this week's calendar"), then answers with the result. Whatever it mentions from a tool (a thread, a document, a file) is a link you can open.

### It knows what you have open

Your accomplice sees which page and item you have open in the app (a thread, a document, a record), so "summarise this" or "reply to her" needs no explanation. The composer shows **Workspace context** when this is on.

### The accomplice panel on every page

On every other page, a panel lets you ask about what is in front of you ("Ask Alma about communications…"). It is the same conversation: what you ask there is remembered in the main one.

### Ask about anything you can see

Hover anything (a chart point, a table row, a figure, a message, a mail thread, a passage of a document, a record, a task) and press **Ask** (or **Alt+A** long-press on a phone; or select text). A small composer opens right there, quoting what you pointed at. Your accomplice answers in place, in a side discussion that stays attached to that spot, or in the chat: **Ask in chat**, **Open as branch**, or **Merge into chat** later.

### Other channels

Your accomplice also answers on the channels you connect for yourself, such as your Telegram, and reaches you by notification when something needs you while the app is closed. Notifications (p. 80)

### Tips

- Ask for outcomes, not steps: "make sure Ana gets the price list by Friday" rather than "open mail".
- Say what you want to happen next: "draft, don't send", "remind me Monday".
- If an answer is wrong, say so plainly: it can save the correction to memory. Memory and rules (p. 19)

## Show and tell

Your accomplice shows instead of describing.

### Links to what it mentions

Every email, chat, document, file or page your accomplice mentions from a tool is a link into the app. Links are written by Mandala from the tool's own result, never guessed, so they always open the right thing. Questions about Mandala itself are answered from this manual, with a link to the page. While it works, each tool it uses appears as a compact card in the chat.

### Dashboards and reports

Ask for something visual ("show me this month's spending", "a dashboard of our SEO") and your accomplice composes it from **blocks**, shown in the chat when small and beside it otherwise:

| Block | Shows |
|---|---|
| **Metrics** | Key figures, with deltas against before and sparklines; it knows whether up is good news |
| **Chart** | Line, area, bar, donut, scatter, funnel, heatmap or sparkline, with tooltips, zoom, a data view, and PNG or SVG export |
| **Table** | Search, filters, sorting, pages, formatted money, dates and links, and CSV export |
| **Callout** | An insight, a risk, a next step, good news |
| **Cards**, **timeline**, **progress** | Items to compare, events in order, how far along |
| **Tabs** | Several views in one place |
| **Markdown** | Short text between blocks |

Blocks sit side by side on wide screens. **Filters** narrow a whole dashboard, and selecting in a chart highlights the matching rows in its table. **Sources** says where each number came from.

### Live dashboards

A dashboard built from your accomplice's read tools (your analytics, your records) stays **live**: press refresh to re-run those tools with the same arguments, or set a **schedule** (every hour, every day…). The refresh says when it ran and what, if anything, failed. A schedule that would re-run costly tools asks you to confirm first.

Everything shown is kept in your Library and stays in the conversation where it was shown. Asking for a change makes a new version of the same item.

### Point at it

Anything on a dashboard — a point on a chart, a row, a figure — can be pointed at and asked about (**Ask**, or **Alt+A**). See Talking with your accomplice (p. 15).

### Live drafting

When your accomplice writes a document, a Goethe page or a Google Doc or Sheet, you watch it being written in the canvas. Google Docs and Sheets it reads or edits are shown live too, through the same accounts and permissions as its tools.

### Newsletters

Your accomplice can write a newsletter in Markdown and render it as an email-safe page in your Library, shown in the canvas. Sending it from one of your Gmail accounts needs your approval of the exact issue, subject and recipient list; recipients are blind-copied.

## Choices, reactions and branches

How to

### Choices

When a decision has a few clear options, your accomplice asks with a small **choice wizard** in the conversation instead of a wall of text: one question at a time, its recommendation marked, and **Something else…** to type your own answer. Use the arrow keys and **Enter**, or tap. Your picks go back as your next message.

A question you talk past (you write something else) disappears: it never nags.

### Reactions

Press **React** under any message and pick an emoji. Your accomplice sees your reactions on its next turn, which is how a ☒ on a suggestion or a ☒ on a joke teaches it what you like. A reaction never starts a turn by itself. It reacts to your messages too, when a reply would add nothing.

### Branches

Sometimes one message deserves its own discussion. Press **Branch** under it: a side conversation about that message opens in the canvas, and you talk it through with your accomplice there, away from the main thread. The message shows a chip with the number of replies. When you are done, **merge** the branch: the side conversation comes back into the main one as a note, so your accomplice has what you found.

## Voice messages and calls

How to

Voice goes through Inworld: your recordings and your accomplice's words are sent to Inworld to be transcribed and spoken, and nowhere else. Connect it once in **Settings → Accomplice → Voice** with an Inworld API key; the key stays on your server.

### Voice messages

When the composer is empty, the **microphone** button records a voice message. Release or press send: it is transcribed and your accomplice answers as if you had typed it. Your recording stays in the conversation.

Turn on **Settings → Accomplice → Voice → Alma answers out loud** to get a spoken reply with every written one. Your accomplice can also choose to speak when you ask it to ("say it out loud").

### Calls

Press **Call** in the conversation for a live call. A compact call widget stays on screen while you keep using the app.

- Talk naturally; you can interrupt.
- When something needs your data or an action (your mail, your calendar, a task), the call hands it to your full accomplice, with its tools and approvals, and says the answer back. It fills the wait with a short natural line instead of silence.
- Every line said lands in your conversation as it is said, and the call shows as a notice (`Voice call · 4:12`) where it started.

On an iPhone, a call stops if you lock the phone: native calls with the system call screen are Coming soon.

### Settings

| Setting | What it does |
|---|---|
| **Voice**, **Language**, **Voice id** | Which voice your accomplice speaks with. Preview before saving. |
| **Pace** | How fast it speaks, from 0.8 to 1.5. |
| **Alma answers out loud** | A spoken reply with each written reply. |
| **Your language** | A hint for transcribing you (`en`, `fr`…). Empty detects it. |
| **Calls → Conversation model** | The realtime model that holds the call (`provider/model`). Empty uses Inworld's default. |
| **How she talks on calls** | Voice craft for calls: turn length, flow, direction tags and non-verbal sounds. Personality still comes from the constitution. |

## Approvals

How to

Before an action that changes something outside the conversation (sending an email or a message, creating a calendar event, writing to a file, approving an invoice), your accomplice prepares it and shows an **approval card**: what exactly will happen, in plain words, with **Approve** and **Deny**. Nothing happens until you decide.

The card approves the **exact** action shown. If your accomplice wants to change a word of the email, it is a new card.

### Three ways to decide

1. **Tap** Approve or Deny on the card, in the conversation.
2. **From your phone.** When no Mandala window is open, the card arrives as a notification. On the iPhone app, Approve and Deny are on the notification itself, after Face ID.
3. **In your own words.** While a card is waiting, write "yes, send it", "go ahead with the Seafrigo one" or "no, not now". When your message clearly approves exactly that action, it decides the card.

### Approval by words

An independent check reads your message and the waiting action and judges whether you clearly approved it; your accomplice never judges this itself. It needs the Jev risk check connected (Settings → Extensions → Jev risk check, with a TypeSafe key); without it, use the card.

Some actions always keep their card, whatever you write: payments, payables, payouts, refunds, transfers, deployments and deletions.

### Questions from coding agents

When a coding session's agent asks permission for something, the question is an approval card too, answered from the chat, the Code page, a notification or your words. See Code (p. 40).

### After approval

The action runs and is followed until it settles (a Telegram message until Telegram confirms it, for example). It is recorded in **Activity** with the approval: who decided, when, and what exactly ran. If it fails, your accomplice tells you and does not retry silently.

### Less asking

Approvals are the default for everything that matters. To let your accomplice do some things directly, see What your accomplice may do alone (p. 19).

## What your accomplice may do alone

How to

By default your accomplice asks before every action that changes something outside the conversation. You can let it act directly on chosen families of actions.

### Turn on a family

1. Open **Settings → Accomplice → Without asking**.
2. Each row is a family of actions it would otherwise ask about (shopping items, a records domain, an integration's writes), with the actions it contains.
3. Turn on the ones it may do directly, then **Save changes**.

By default, only **shopping lists and their items** are on: adding milk to your groceries should not need a card.

Every action taken this way is still recorded in **Activity** as approved by this setting, so you can always see what happened.

### The risk check

A second opinion can send a single action back to you even when its family is allowed. When the Jev risk check is connected (a TypeSafe key in Settings → Extensions → Jev risk check, or `ALMA_TYPESAFE_API_KEY`), before each action your accomplice may take alone it asks five questions: is it hard to undo, does it involve money, other people, private information, or is it unusual? If any is likely, the action comes back to you as a normal card, with the reason. If the check cannot answer, the action is asked about too.

The check runs at TypeSafe: the action's name and arguments are sent there, finance and health included. Leave it unconnected if you do not want that; your settings then apply as they are.

### What never runs alone

- Anything not covered by a family you turned on.
- Sending messages as you in chats (each waits for approval unless you turned on *Smart approval* for Telegram routine replies; see Telegram (p. 26)).
- Reorganising a vault that is not your accomplice's own.

### Earned autonomy

Coming soon — a first week in which your accomplice only shows what it *would* have done, then proposes taking over one kind of action at a time, from evidence ("you approved all 12 grocery additions — shall I just do these?"). See Goals, initiatives and reflection (p. 21).

## Memory and rules

How to

### Memory

Your accomplice saves lasting facts as **memory notes**: about you ("prefers mornings for deep work"), people ("Ana handles purchasing at Seafrigo") and your businesses. It recalls the most relevant ones in every conversation and searches the rest when needed.

- **See it:** ask "what do you remember about Ana?", or open **Knowledge**, where *Alma's memory* lists every note.

- **Correct it:** say “that’s wrong, Ana moved to finance”, or open the note in Knowledge and choose **Correct it with Alma**.
- **Remove it:** “forget that”, or **Forget this** on the note in Knowledge.

Memory is stored in your Mandala, on your machine.

Coming soon — evening **memory cards** in the first weeks (“What I learned about you today”, each with keep, fix or forget), and a memory policy in your accomplice’s persona (“ask before remembering personal things”).

### Standing rules

Rules are instructions your accomplice follows in every conversation, on every channel.

- **Add one by saying it:** “from now on, always copy Ben on supplier emails”, “never schedule calls before 10:00”. Your accomplice saves it as a rule and says so.
- **Manage them: Settings → Accomplice → Standing rules** lists every rule with the date it was added. Edit, remove, or add one directly.

Write rules as you would for a new assistant: specific, and about what to do rather than what to think.

| Instead of | Write |
|---|---|
| “Be careful with money” | “Ask me before any payment over €500, even if payables are allowed.” |
| “Don’t bother me” | “Between 22:00 and 07:00, only notify me about deliveries and payments.” |

### Rules versus constitution versus memory

| | Is | Changes when |
|---|---|---|
| **Constitution** | Who your accomplice is and how it works | You edit it |
| **Standing rule** | An instruction it always follows | You say “from now on…” or edit the list |
| **Memory** | Facts it learned | It learns something, or you correct it |

## Morning briefing, follow-ups and diary

Your accomplice keeps a quiet daily rhythm, run inside Mandala even when no window is open.

### Morning briefing (07:30)

Each morning it looks at today’s and overdue tasks, today’s calendar, and anything urgent in mail and chats, and sends you a short plan for the day with what really matters. If there is genuinely nothing, it stays silent.

Change the time with `ALMA_MORNING_TIME=08:15` in the environment file, or turn it off with `ALMA_MORNING_TIME=off`. The time is in `ALMA_TIMEZONE`.

### Follow-ups

When your accomplice is waiting on something (a reply, a delivery, a decision) it schedules a **follow-up** and comes back to it at that time, in your conversation. Ask “remind me Monday to call the bank” or “check on Ana’s reply in two days”. Follow-ups survive restarts.

### Background triage

New mail in watched inboxes and new messages in watched chats are triaged in the background. Most need nothing. Something that needs you now (money owed, an unhappy customer, a delivery problem, a deadline) becomes a notification; something for later becomes a task with a due date. Triage never replies to anyone.

### The diary (23:30)

Each night your accomplice writes a private diary entry in its own voice: what happened between you today, what it learned, what to remember tomorrow. Lasting facts go to its memory. Change the time with `ALMA_DIARY_TIME`, or turn it off with `off`.

When goals and initiatives are on, the nightly diary becomes your accomplice's **reflection**, shown on the Initiatives page. See Goals, initiatives and reflection (p. 21).

Coming soon — a briefing capped at five items that ends with "That's it for today", quiet hours and "not this kind again" on every notification.

## Goals, initiatives and reflection

Beyond answering you, your accomplice can pursue your goals on its own: notice what matters, propose initiatives, and work on them step by step. This layer is **off** until you turn it on, and starts in **shadow mode**, where it proposes and never acts.

### The pieces

- **Goals** — what you want, with why, how success is measured and by when. Written with you in a goals interview.
- **Reflection** — one bounded step, a few times a day at most: your accomplice reads a digest of what changed, your goals and its open initiatives, and decides what to propose, what to ask you, and what to write in its diary.
- **Initiatives** — pieces of work it proposes towards a goal. Each one is worked on one verified step at a time, in its own conversation.

All of it lives as documents in your accomplice's own vault (`docs/goals/`, `docs/initiatives/`, `docs/reflections/`), readable in Knowledge.

### Start with your goals

1. Open **Initiatives** and press **Define goals with Alma**.
2. Your accomplice interviews you one question at a time, drafting from what it already knows and marking its assumptions for you to confirm. You can do it by text, or on a call.
3. Press **End goals interview** when you are done. Each goal shows its success measure and, later, its assessment.

### Turn it on

On the Initiatives page, under **Switches**, three buttons:

| Button | Default | Meaning |
|---|---|---|
| **Off (turn on)** / **On** | off | The kill switch. Off: your accomplice behaves exactly as without this layer. |
| **Shadow mode** / **Working** | shadow | Shadow: it reflects, proposes and writes the diary, and works on nothing. Press *let her work* to allow work on approved initiatives. |
| **Pause** | running | Stop for now, keep everything. |

Under **Limits**: quiet hours (22:30–07:00 by default), background tokens a day (400,000), work steps a day (12), reflections a day (4), open initiatives at most (7), new initiatives a day (3), and at least 3 hours between two event-driven reflections. Your own conversation always comes first.

### Working with initiatives

Each initiative shows its goal, its outcome so far and what it needs. **Approve** a proposed one to let it start (outside shadow mode), **Resume** a paused one, or **Send note** to steer it. Questions your accomplice asks appear with an answer field.

Rate reflections with **Useful** or **Not this**: your feedback shapes the next ones. The page also shows background tokens and steps used today, your accomplice's median answer time, and the share of suggestions you found useful.

Coming soon — the shadow-week reveal: after a week, "here's what I would have done, and how it would have turned out", with one-tap promotion per kind.

## Sensitive mode

How to

Sensitive mode allows adult content in your private conversation with your own accomplice. It is never on by default, never applies to shared spaces or organisation agents, and is for adults who choose it explicitly.

### Turn it on

1. **Settings → Accomplice → Conversation → Content mode**: choose **Sensitive**.
2. Optionally, write **Sensitive instructions** (Settings → Accomplice → Constitution, below the constitution). They are added to the constitution only while sensitive mode is on.
3. Optionally, choose a **Sensitive conversations** model in Settings → Models → Workload routing. Many providers refuse adult content; choose one whose terms allow it.

Choose **Standard** to turn it off.

### Automatic routing

With the Jev risk check connected, **Settings → Models → Workload routing → Routing with Jev → Adult content to the sensitive model** sends a conversation to the sensitive model and instructions when Jev finds it is adult content. Your messages go to TypeSafe for this classification. Off by default.

### Pictures

When your accomplice has reference pictures and an image provider (WaveSpeed) is connected, it can send pictures of itself; in sensitive mode those may be explicit. Pictures are kept in your Library.

## What your accomplice can do

Your accomplice acts through **tools**. Which ones it has depends on what you connected and what you allowed.

### Always at hand

| Family | Lets it |
|---|---|
| **memory**, **rule** | Remember facts, follow and save standing rules. |
| **tasks**, **followup** | Add, complete and reschedule tasks; come back to something later. |
| **goethe** | Read and write documents, tasks, types and views in vaults it may use — including this manual, the **Docs** vault (`goethe.docs.*`). |
| **google** | Gmail, Calendar, Drive, Docs, Sheets, Analytics, Search Console and Meet, per account and per service as you allowed. |
| **telegram**, **whatsapp**, **comms** | Read allowed chats, prepare messages, sort and translate. |
| **library** | Open and file what arrives: attachments, uploads, its own reports. |
| **web** | Search the web and read pages. |
| **show**, **ask**, **react** | Show things in the canvas, ask you to choose, react with an emoji. |
| **pictures**, **selfie** | Generate pictures, when an image provider is connected. |
| **notify_owner** | Tap you on the shoulder when something needs you now. |

### Loaded when needed

Large families wait until a request needs them, so every turn stays fast: product analytics (PostHog), SEO (Ahrefs), ads (Meta Ads), GitHub, MCP servers, coding sessions (`code.*`), and the actions and queries of your business records (`dhl.*`). Your accomplice loads a family with one step when your request calls for it, and it stays loaded in that conversation for six hours.

### Your records

Every action and query of the DHL contracts your node runs becomes a tool, with labels and refusal reasons from the contract. Your accomplice fills in only what matters (the signer, the owner and paging are added by Mandala), and DHL refuses anything the contract does not allow. See Building with DHL (p. 59).

**Mandala's own documentation**

When you ask how Mandala works or how to do something in it, your accomplice searches the Docs vault, opens the right page, answers in a few steps and links the page. Pages marked Coming soon describe what is not built yet, and it says so.

# Connecting your life

Each connection is optional, and each opens only what you choose. Connect what you use every day first; your accomplice becomes useful the moment it can read one real channel.

| Connection | What it brings | Page |
|---|---|---|
| Gmail | Mail in Communications, triage, drafts and replies | Set up your email (p. 25) |
| Google Calendar, Drive, Docs, Sheets | Your agenda, files and documents | Google Calendar, Drive, Docs and Sheets (p. 26) |
| Telegram | Your personal account(s) in Communications | Telegram (p. 26) |
| WhatsApp | Your number(s), linked as a companion device | WhatsApp (p. 27) |
| Apple Health | Daily measurements and workouts from your iPhone | Apple Health (p. 28) |
| PostHog | Product analytics, with alerts that wake your accomplice | PostHog (p. 29) |
| Ahrefs | SEO data within a units budget | Ahrefs (p. 29) |
| Meta Ads | Ad accounts to read and propose changes to | Meta Ads (p. 29) |
| Facebook and Instagram | Pages and Instagram accounts: posts with approval, comments | Facebook and Instagram (p. 30) |
| Cloudflare | Webhooks through a tunnel; DNS and zones with approval | Cloudflare and webhooks (p. 30) |
| GitHub | Repositories to read and change, with approval | GitHub (p. 31) |
| MCP servers | Tools from any remote MCP server you review | MCP servers (p. 32) |
| Web search and pictures | Brave Search and WaveSpeed images | Web search and pictures (p. 32) |

Model providers and voice are connected in Settings too: see Choose a model (p. 10) and Voice messages and calls (p. 17).

## Where to connect

Everything is in **Settings → Extensions**. **Settings → Overview** shows what is live and what needs attention, such as an expired sign-in.

## Who owns a connection

Every account belongs to an identity: you, a company you own, or a shared space. A company's mailbox belongs to the company, and you read it by delegation. A single chat on your personal account can belong to a company ("Belongs to" in Communications). See Who owns what (p. 49).

## Before you connect

Read Privacy, connection by connection (p. 32): it says, for each connection, what is read, where it is stored and which outside services see it.

## Set up your email

Partial How to

Mandala reads and sends mail through Gmail. Each Gmail account shows in Communications with its mailboxes and tabs, and your accomplice triages new mail in the background.

> Coming soon — **Connect Google** from Settings. Today, adding a new Google account from the app is not available; accounts connected before (from an earlier Alma) appear and can be configured below.

### Choose what your accomplice may do

Open **Settings → Extensions → Google** and pick the account.

- **Account name** and **Description** tell your accomplice what the account is for ("Sam's business mail; suppliers and customers").
- For **Gmail**: **Access** lets it read; **Write** lets it prepare drafts and sends (each still asks you); **Monitoring** lets new mail reach background triage, checked every *n* minutes.

Press **Save changes**.

### Mailboxes and tabs

In **Communications**, each account has **Inbox**, **Starred**, **Sent**, **Drafts** and **Spam**, and Gmail's tabs (**Primary**, **Promotions**, **Social**, **Updates**, **Forums**). **All accounts** shows every account together. Stars sync with Gmail's starred label.

**Smart boxes** sort threads into *Needs reply*, *Waiting on others*, *Money*, *Personal* and *Newsletters*, as messages arrive (requires the Jev key; see Privacy, connection by connection (p. 32)).

### Signatures

**Settings → Extensions → Communications**: one signature card per account, in Markdown (`**bold**`, `*italic*`, `[links](https://…)`), added **on new emails**, **on replies**, or both. In the composer, leave it out for one email with the signature button.

### Writing

Press **New email**, or reply in a thread.

- **Write as**: which account sends.
- **Rich text** or **Markdown**, with a **Preview**.
- **Translate** the draft into another language before sending, and **Replace draft** with the translation.
- **Draft with Alma**: your accomplice writes the draft; you edit and send.

Nothing goes out until you press send. Your press is the approval, recorded like any other.

### Reading

- **Translate into** (Settings → Extensions → Communications → Reading): messages in another language offer a translation.
- **Load images in emails**: off keeps remote images hidden until you ask, so senders cannot tell when you open their mail.
- **Show original** and **Show quoted text** reveal the full message.

### Triage

With monitoring on, new Inbox mail (promotions and social excluded) is handed to background triage: most needs nothing, some becomes a task, urgent mail becomes a notification. See Morning briefing, follow-ups and diary (p. 20).

## Google Calendar, Drive, Docs and Sheets

Partial How to

A Google account brings more than mail. Each service is opened to your accomplice separately, per account.

> Coming soon — adding a Google account from Settings. Accounts already connected can be configured as below.

### Set access per service

**Settings → Extensions → Google**, pick the account, then for each service:

| Column | Meaning |
|---|---|
| **Access** | Your accomplice may read it. |
| **Write** | It may prepare changes (events, files, documents). Each still asks for your approval. |
| **Monitoring** | Changes reach background triage, checked every *n* minutes. |

Services: **Gmail**, **Calendar**, **Drive**, **Docs**, **Sheets**, **Analytics**, **Search Console**, **Meet**.

Press **Save changes**. The connection status shows whether the account is connected and when its sign-in renews; tokens refresh automatically.

### Calendar

Your Google calendars appear in **Agenda** beside your tasks. Your accomplice reads them to plan and warn about clashes, and creates, moves or deletes events with your approval. See Agenda (p. 41).

### Drive, Docs and Sheets

Ask “find the Seafrigo contract in my Drive”, “summarise this doc”, “add a row to the stock sheet”. Docs and Sheets your accomplice reads or edits open live in the canvas. Edits wait for your approval.

### Analytics, Search Console, Meet

Your accomplice can run Google Analytics reports, read Search Console data, and create Meet spaces or read conference records, where you allowed it.

## Telegram

How to

Mandala signs in to Telegram as you, with your personal account, like a desktop Telegram app. You can add several accounts. Your accomplice reads only the chats you allow, and every message it sends as you waits for your approval.

### Before you start

- Create a Telegram API app once at my.telegram.org → *API development tools*. You get an **api_id** and an **api_hash**.
- The server needs TDLib (`libtdjson`); see Install on Linux (p. 6).

### Sign in

1. **Settings → Extensions → Telegram**.
2. Enter **api_id** and **api_hash**, then **Save**.
3. Enter your **phone number** with its country code, then **Continue**.

4. Enter the **login code** Telegram sends to your Telegram app, then your **two-step password** if you have one.
5. The account shows **Signed in as** your name.

### What your accomplice may do

| Switch | Meaning |
|---|---|
| **Read chats** | It may read chats you allow, one by one. |
| **Send messages** | It may send as you; each message waits for your approval. |
| **Watch new messages** | New messages in allowed, watched chats reach background triage. |
| **Smart approval** | Routine replies it is confident about can go without a card. Off by default. |

### Allow a chat

Open the chat in **Communications** and press **Allow Alma to see this chat**. Until then its messages are not loaded and your accomplice cannot read it.

### More accounts

**Settings → Extensions → Communications → Telegram accounts → Add account**, then sign in step by step (phone, code, password). Added accounts use the first account's API app. Each account's chats show under it in Communications.

### Sign out

**Sign out** stops all reading and sending at once.

## WhatsApp

How to

Mandala links to your WhatsApp number as a **companion device**, the way WhatsApp Web does. Your phone stays the primary device.

> **Read this first.** Mandala uses WhatsApp's unofficial multi-device protocol. WhatsApp can ban numbers that use unofficial clients. Link a number only if you accept that risk.

### Link your number

1. **Settings → Extensions → WhatsApp**.
2. Type your number with its country code (`+33 6 12 34 56 78`) and press **Get a code**.
3. On your phone: **WhatsApp → Settings → Linked devices → Link a device → Link with phone number instead**, and type the 8-character code Mandala shows.
4. The account shows **Linked**, with your number.

The session reconnects by itself after restarts.

### What your accomplice may do

- It reads only chats you allow, one by one, in **Communications** (**Allow Alma to see this chat**).
- Every message it sends waits for your approval, and is recorded before it leaves, so a restart never sends it twice.
- Incoming messages are stored before WhatsApp is acknowledged, so a crash loses nothing.

### More numbers

**Settings → Extensions → Communications → WhatsApp numbers → Add number**, then **Get a code** for that number. Each number links as its own device, and its chats show under it in Communications.

### Unlink

**Unlink** in Settings, or remove the device from your phone's *Linked devices*. The account then shows *Unlinked from the phone.*

### Not yet

Coming soon — downloading photos and voice notes from WhatsApp, and linking by QR code from the app.

## Apple Health

Partial How to

Your iPhone's Health data can flow into Mandala as **daily measurements** in your personal records: one value per metric per day, and one record per workout. Your accomplice reads them with its personal-records tools ("how did I sleep this week?").

Data arrives two ways: pushed regularly by the **Health Auto Export** app (App Store), or uploaded as Apple's own **export.zip**. See it in **Personal → Health**.

> Coming soon — the Mandala iPhone app reads Health directly in the background, with no third-party app. Turning it on replaces Health Auto Export.

### Set it up

1. **Settings → Extensions → Apple Health → Create token.** Copy the token now: it is shown once. The page also shows the **Endpoint**.
2. Turn on **Accept pushes**.
3. In Health Auto Export on your iPhone: **Automations → New automation → REST API**.
4. **URL**: the endpoint. **Headers**: `Authorization` = `Bearer <token>`.
5. **Data type**: *Health Metrics* (all, or the ones you want). Add a second automation for *Workouts* if you want them.
6. **Export format** JSON, **export version** *Version 2*, **Aggregate data** on, interval *Days*, date range *Last 7 days*.
7. **Sync cadence**: every hour, or as often as you like. Your iPhone must be on your tailnet (Tailscale on) to reach the address.
8. Tap **Manual export** once. The result appears under **Last import** in Settings.

### Choose what to keep

Under **What to keep**, turn categories on or off and **Save categories**. Metrics in a category that is off are dropped on arrival.

### Apple's own export

On the iPhone: **Health → your picture → Export All Health Data**. Upload `export.zip` under **Apple's own export**, choosing **How far back**. It is read on the server without loading it whole, and re-importing changes nothing.

### Backfill with Health Auto Export

**Import a file**: upload a Health Auto Export JSON export (Version 2). Imports are idempotent: a day already recorded with the same value is left alone, and today's value is replaced as it fills.

### Your accomplice and your health

Under **Alma** in the same page:

- **Daily digest** at a time you choose: yesterday's numbers, what stands out from your usual, and your goals.
- **Tell Alma after *n* hours** without a push from the phone, so a broken automation is noticed.

**Personal → Health** shows what stands out against your usual, your **goals** (**Add goal**), your measurements and two weeks of workouts. See Personal (p. 37).

### Disconnect

**Disconnect** revokes the token; pushes are refused from then on. **New token** replaces it.

## PostHog

How to

Connect PostHog projects so your accomplice can answer product questions (“how many signups this week?”), and so chosen events, alerts or metric thresholds wake it.

### Add a project

1. **Settings → Extensions → PostHog**.
2. Add a project: a **Name** (lowercase letters, digits, `-` and `_` fixed once saved), the **Region**, and the **Project ID** (the number in the project’s URL, `…posthog.com/project/<ID>`).
3. **Save changes**, then paste a **Personal API key** for it (PostHog → Settings → Personal API keys).

### What your accomplice may do

- **Changes**: lets it propose creating or editing insights, dashboards, feature flags, cohorts, surveys and experiments. Each change still asks you.
- **Events to watch**: event names that wake it when they occur (comma separated).
- **Alerts to follow**: PostHog alert IDs whose firing it is told about.
- **Check every**: how often to look.

### Metric checks

A **metric check** is a HogQL query or a saved insight whose value wakes your accomplice when it crosses a threshold. Add one with **Add check**: a name, the source (*HogQL* or *Insight*), the query or insight ID (the first row and column is the value), **Wake Alma when the value is** *below* or *above* a threshold, and an interval.

Signals arrive in background triage, batched once a minute; **Recent signals** shows the latest.

### Tools

PostHog is a large tool family: your accomplice loads it when a request needs it. See What your accomplice can do (p. 22).

## Ahrefs

How to

1. **Settings → Extensions → Ahrefs**, and paste an **Ahrefs API v3 key** (Ahrefs → Account settings → API keys). It is stored encrypted.
2. Under **What Alma may do**:
   - **Use Ahrefs** — Site Explorer, Keywords Explorer, Rank Tracker, Site Audit and Brand Radar reads.
   - **Manage projects** — create and change projects, keywords, competitors and Brand Radar reports. Each change still asks you.
   - **Background work** — lets background work read Ahrefs, not only your conversations.
3. Set the **Units budget**: **Calls per day** and **Rows per request**. Every Ahrefs call costs API units; these limits keep your accomplice within your plan. The page shows calls and units used today.

## Meta Ads

How to

1. **Settings → Extensions → Meta Ads**, and paste an **Access token**: a Marketing API token with `ads_read` (and `ads_management` for changes), for example a system-user token from Meta Business Settings.
2. **Add an ad account** by its ID from Ads Manager (`act_123…` or just the digits). Only listed accounts are visible to your accomplice.
3. For each account, choose whether it may only read, or also propose changes. Every change asks you first.

For Facebook Pages and Instagram, see Facebook and Instagram (p. 30).

## Facebook and Instagram

How to

Connect your Facebook **Pages** and their linked **Instagram professional accounts**. Personal profiles aren't supported. Every post, reply, hide or deletion asks you first, with the exact text, media and time.

### 1. Create a Meta app

In Meta for Developers, create an app with **Facebook Login** and the Page and Instagram permissions. Then in **Settings → Facebook & Instagram → Meta app**:

- **App ID** and **App secret**
- the **Facebook Login callback** Mandala shows (add this exact address to the app's valid OAuth redirect URIs);
- a **Webhook verify token**: a random string of 32+ characters, also pasted in Meta's webhook setup.

### 2. Connect

Press **Connect with Facebook** and choose the Pages your accomplice may use. Mandala finds them and their Instagram accounts.

### 3. Choose what your accomplice may do

Per account (**Choose what Alma may do**):

| Switch | Meaning |
|---|---|
| **Read posts and comments** | It may read them |
| **Publish** | It may prepare posts; each asks you |
| **Reply and moderate** | It may prepare replies, hides and deletions; each asks you |
| **Follow comments** | New comments on this account's posts wake it |

Stories need an Instagram **Business** account.

### Publishing

An approved post becomes a job: text, links, photos, videos, Reels, stories, Instagram carousels. Mandala advances it one step at a time and records every step, so a restart resumes without posting twice; a post whose final step is uncertain is checked, never re-sent. Scheduled posts are followed until they go out, and your accomplice hears the outcome.

### Comments

Comments arrive by webhook (needs the tunnel to expose `/hooks/meta` see Cloudflare and webhooks (p. 30)) and by a regular sweep of recent posts, so nothing is missed. Your accomplice hears about new comments by others, never about its own replies.

### Disconnect

**Disconnect** forgets every token and blocks queued posts.

## Cloudflare and webhooks

How to

Some services tell you about events by calling a URL: GitHub when CI finishes, Meta when someone comments. Mandala receives them on a **separate local port** that serves nothing but these webhooks, and a **Cloudflare tunnel** exposes only those paths to the internet. The app itself is never reachable through the tunnel.

### The webhook listener

Mandala listens on `127.0.0.1:8788` (`MANDALA_WEBHOOK_LISTEN`) for:

| Path | From | Checked by |
|---|---|---|
| `/hooks/github` | GitHub deliveries | The webhook secret's signature |

| Path | From | Checked by |
|---|---|---|
| `/hooks/meta` | Meta's verification and comment events | The verify token and the app secret's signature |
| `/hooks/meta-media` | Meta fetching an approved post's media | A one-day link per file, only while its post publishes |

Everything else answers 404. Each delivery is admitted once.

### Connect Cloudflare

1. Install `cloudflared` (2025.4 or newer) at `/usr/bin/cloudflared` (or set `ALMA_CLOUDFLARED_PATH`). **Settings → Cloudflare** shows *Installed* and whether the listener is up.
2. Create a Cloudflare **API token** with *Zone Read*, *DNS Edit* and *Cloudflare Tunnel Edit*, and paste it under **Connection**.
3. **Add an account** with its **Account ID** (Cloudflare dashboard → account home), the **Zone IDs** your accomplice may read, the **Webhook hostnames** a tunnel may use (a dedicated one, such as `hooks.example.com`), and whether it may **Allow changes**. **Watch tunnels** tells your accomplice when a tunnel's health changes.
4. Save.

### Create the tunnel

Ask your accomplice: "set up the webhook tunnel on hooks.example.com for GitHub". It prepares a tunnel whose ingress is exactly the webhook paths to the listener (then 404), and a DNS record, and asks for your approval. Mandala then runs `cloudflared` itself, with the tunnel token stored sealed.

- **Resume** a stopped tunnel: your accomplice first checks at Cloudflare that the ingress and DNS are still exactly what you approved.
- **Stop the connector** (Settings): stops `cloudflared` the tunnel and DNS stay, so it can resume.
- **Adopt an existing tunnel**: records a tunnel you created before as stopped, without changing anything at Cloudflare; then ask your accomplice to resume it.
- Removing the token stops the connector and forgets the tunnel token.

### DNS and zones

With **Allow changes**, your accomplice can change DNS records and a few zone settings, each with your approval, re-checked against the record just before applying.

## GitHub

How to

1. **Settings → Extensions → GitHub**, and paste a **fine-grained personal access token** for the repositories you want (contents, pull requests, issues, actions). It is stored encrypted.
2. **Add a repository** as `owner/name`. Your accomplice sees only listed repositories.
3. Per repository:
   - **Allow changes** — branches, commits, comments, pull requests and workflow runs. Each one still asks you.
   - **Allow merging** — merging pull requests, asked each time.

### Events

GitHub tells Mandala about pushes, pull requests and CI runs by **webhook**, through the Cloudflare tunnel (Cloudflare and webhooks (p. 30)).

1. **Settings → GitHub → Webhook**: set a **Webhook secret** (32+ characters). Mandala shows the **Payload URL** once the tunnel exposes `/hooks/github`.
2. In GitHub: the repository's **Settings → Webhooks → Add webhook**: that URL, content type `application/json`, the same secret, and the events you want.
3. Under **Events that wake Alma**, per repository: which events, **Only these actions** (opened, closed, completed…) and **Only these branches**.

Every signed delivery is kept two weeks for coding sessions: a CI result on a session's branch refreshes its checks at once (Code (p. 40)). The events you chose also wake your accomplice.

### Coding sessions

To have Claude Code or Codex work in your repositories, see Code (p. 40).

## MCP servers

How to

The Model Context Protocol lets services offer tools to AI agents. Mandala connects to **remote** MCP servers over public HTTPS, and you review each tool before your accomplice may call it.

### Add a server

1. **Settings → Extensions → MCP servers**.
2. Add a server: a **Name**, an **ID** (lowercase letters, digits and -), and the **Endpoint URL**.
3. If it needs a token, turn on **Needs a token** and paste the **Bearer token**. It is stored for that exact endpoint; changing the URL needs a new token.
4. Set **Requests per day**.
5. Press **Discover tools**.

### Review its tools

Each discovered tool has three settings:

| Setting | Meaning |
|---|---|
| **Off** | Your accomplice never sees it. |
| **Read** | It calls the tool directly. Use for tools that only read. |
| **Ask** | Each call waits for your approval. |

Save. MCP tools load on demand, like other large families.

### Remove

**Remove server** takes away its tools at once.

## Web search and pictures

How to

### Web search

Your accomplice searches the web and reads pages with **Brave Search**.

**Settings → Extensions → Brave Search**, paste a Brave Search API key, **Connect**. Or set `ALMA_BRAVE_API_KEY` in the environment file.

Searches are sent to Brave. Pages are fetched from your server.

### Pictures

Your accomplice generates pictures with **WaveSpeed** (and, with reference pictures, pictures of itself).

**Settings → Extensions → WaveSpeed images**, paste a key, **Connect**. Or set `ALMA_WAVESPEED_API_KEY`.

Generated pictures are kept in your Library.

## Privacy, connection by connection

Everything Mandala stores lives on your node, in your data directory. Credentials are encrypted with your node's master key. Some features send data to outside services to work; this page lists them all.

### Always

- **Your model provider** sees what your accomplice reads to answer: your messages, and the tool results of the turn (an email it opened, a document it read). Choose a provider whose terms you accept, or a local model with Ollama. Choose a model (p. 10)

### By connection

| Connection | What Mandala reads | What your accomplice may read | Outside services involved |
|---|---|---|---|
| **Gmail** | Every thread, into a local index for Communications | Per account and service, as you set in Settings → Extensions → Google | Google |
| **Calendar, Drive, Docs, Sheets** | What you or your accomplice open | Per service, per account | Google |
| **Telegram** | Your chats, stored locally (TDLib) | Only chats you allow, one by one, in Communications | Telegram |
| **WhatsApp** | Messages arriving after linking, stored locally | Only chats you allow in Communications | WhatsApp (unofficial protocol: a number can be banned) |
| **Apple Health** | What your phone pushes, as daily measurements | Your personal records | None |
| **PostHog, Ahrefs, Meta Ads, GitHub, Facebook & Instagram, Cloudflare** | What your accomplice queries | Projects, accounts and repositories you list | That provider |
| **MCP servers** | What the server returns | Tools you reviewed and allowed | That server |
| **Voice** | Recordings and spoken replies | — | Inworld |
| **Web search** | Search queries | Results | Brave |
| **Pictures** | Prompts and reference pictures | — | WaveSpeed |

### Optional: Jev (TypeSafe)

Three features use Jev, a model hosted by TypeSafe. They are off until you add a TypeSafe key:

- **Approval by words** and the **risk check**: the action's name and arguments, which may include finance and health details.
- **Smart boxes** in Communications: a thread's title, people and latest text, for threads your accomplice may read.
- **Routing with Jev**: your last few messages, to pick a model.

### Your accomplice's limits

Your accomplice reads only what the connection's settings allow, and only for the identity you are acting as. Reading a chat it has not been allowed to read fails, whatever it is asked.

# Workspace guide

The Workspace is Mandala's app. It runs in any modern browser, installs on your phone's Home Screen, and opens inside the Mac app.

## The sidebar

| Page | What it is for |
|---|---|
| **Accomplice** (first item; *Alma* in current builds) | Your conversation with your accomplice. Talking with your accomplice (p. 15) |
| **Communications** | Mail, Telegram and WhatsApp in one place. Communications (p. 35) |
| **Knowledge** | Vaults of typed documents, linked and searchable. Knowledge (p. 36) |
| **Library** | Files: attachments, uploads, what your accomplice made. Library (p. 37) |
| **Personal** | Groceries and money. Personal (p. 37) |
| **Dashboards** | Payables for the identity you act as. Dashboards (p. 38) |
| **Analytics** | Money, suppliers and tasks at a glance. Analytics (p. 38) |
| **Data** | Every record your rules govern, and the Workbench. Data (p. 39) |
| **Tasks** | Every task across your documents. Tasks (p. 39) |
| **Code** | Coding sessions with Claude Code or Codex. Code (p. 40) |
| **Agenda** | Calendars and dated tasks together. Agenda (p. 41) |
| **Activity** | What your accomplice did, turn by turn. Activity (p. 41) |
| **Initiatives** | Goals, initiatives and reflections. Initiatives (p. 42) |
| **Pins** (below the pages) | Your bookmarks, in folders, with superpins at the top. Pins (p. 42) |
| **Settings** (gear, bottom left) | Everything that is configured. Settings (p. 43) |

Collapse the sidebar with **⌘B** (**Ctrl+B**). On a phone, the menu button opens it.

## Acting as

At the bottom of the sidebar, your name opens the **Act as** switcher: you, the organisations you own, the spaces you share. Communications, Knowledge, Library, Personal, Dashboards, Analytics and Data follow it, and your accomplice is told. Acting for a company is acting under its delegation, and every action says so. See Acting for a company (p. 50).

## Every place has a link

Each page and each item in it has its own address (`/knowledge/clients/acme.md`, `/communications/mail:…`, `/agenda?view=week`). Back, forward, reload and shared links land in the same place.

## Search

The search button opens **Find anything in your world**: pages, and conversations by person or title. Searching documents and records from here is Coming soon use Knowledge's search, or ask your accomplice.

## Not yet

Coming soon — a simpler navigation: your accomplice, **Inbox** (Communications), **Plan** (Tasks and Agenda), **Knowledge** (vaults and Library), then one **space** per identity with its operations, records, numbers, rules and agents.

## Communications

How to

Communications is one inbox for every channel: each Gmail account, each Telegram account and each WhatsApp number, with the same list, the same reading view and the same composer.

### The left column

- **Needs attention** and **Unread**.
- **Mailboxes**: Inbox, Starred, Sent, Drafts, Spam, and Gmail's tabs.
- **Smart boxes**: *Needs reply*, *Waiting on others*, *Money*, *Personal*, *Newsletters*, sorted by Jev as messages arrive (with a TypeSafe key).
- **Channels** and **Accounts**: one channel or account at a time, or **All accounts**.
- **People**: everyone you talk with. One click shows all their threads across mail, Telegram and WhatsApp.

Search filters the list as you type.

### People, not channels

Conversations are matched to **contacts** in your Goethe vaults by email address, phone number or name. A thread names its contacts; click one to open the contact page in Knowledge. Star a person to keep them at the top.

Create contacts as typed documents (type `contact`) or in a `contacts` folder of a vault.

### Reading and replying

Open a thread to read it. Mail renders as sent, with remote images held back unless you allow them. Replies use the composer: rich text or Markdown, the account to write as, your signature, and translation. **Draft with Alma** asks your accomplice for a draft you edit. Nothing goes out until you press send.

**Create task** turns a message into a task.

### What your accomplice sees

A Telegram or WhatsApp chat stays closed to your accomplice until you press **Allow Alma to see this chat**. Mail follows each Google account's settings. See Privacy, connection by connection (p. 32).

### Belongs to

Each conversation belongs to an identity. **Belongs to** moves a single conversation to another identity, for example a supplier chat on your personal Telegram to your company. It then shows when you act as that company, and follows its permissions. See Who owns what (p. 49).

### Fast

The list reads Mandala's own index of every thread, so switching views is instant and counts are always there. Rows keep their place while you scroll; new threads wait behind an **N new** pill. Each row shows its source: the channel and the mailbox or chat account. Search answers from the index as you type, then Gmail's own search adds what the index lacks.

Select several threads to archive, mark read or unread, star or report spam at once.

### Keyboard

| Key | Does |
|---|---|
| `j` / `↓`, `k` / `↑` | Next / previous thread |

| Key | Does |
|---|---|
| `Enter` / `o` | Open |
| `Esc` | Close, or clear the selection |
| `x` | Select |
| `e` | Archive (or back to Inbox) |
| `s` | Star |
| `u` | Read / unread |
| `!` | Spam |
| `r` | Reply |
| `/` | Search |
| `?` | These shortcuts |

### Attachments

Each file on a mail or Telegram message has **Open** (a panel beside the thread, a sheet on a phone: PDFs, images, text, audio and video inline) and **Add to Library**. Files already in your Library show **In Library**.

## Knowledge

How to

Knowledge is where what you know lives: notes, contacts, procedures, plans. Documents are plain Markdown files in a **Goethe vault**, so they stay readable by any editor, and Goethe understands them: their types, links, tasks and calculations.

### Vaults

Each identity can have its own vault: yours, each company's, the household's. Your accomplice has a private one for its goals and diary. The **Docs** vault holds this manual.

The vault picker at the top switches between the vaults you can open. Acting as a company opens its vault; if it has none, Knowledge offers to create it (Mandala restarts to load it).

### The sidebar

A real document manager: **Documents**, **Types** and **Views**, documents grouped by **Folders** or by **Types**. The sidebar is wider, resizable and remembered. Folders open and close and show their counts. *Alma's memory* lists what your accomplice remembers.

- **+ (New document)** asks for a **name**, a **folder** and a **type** (or *Plain document*) up front. Renaming a document's title renames its file.
- **Drag and drop** documents between folders; select several with ⌘/Ctrl- or Shift-click.
- **Move to…**, **rename** (F2), **duplicate**, and **delete**, which moves to **Archived**, where **Restore** brings it back.
- **New folder**, and **⌘X / ⌘V** (Ctrl on Windows and Linux) to cut and paste.
- Every move can be undone (**Undo**).
- **Tidy up** asks your accomplice for a better structure, which you review before anything moves: Folders, moves and tidy-ups (p. 44).

### Writing

The editor is Markdown with live preview: headings, lists, tables and images render as you type, and the source is one click away.

| Type this | You get |
|---|---|
| `[[` | A link to another document, with completion. Links show the target's title. ⌘-click or Ctrl-click opens it. |

| Type this | You get |
|---|---|
| `- [ ] Call Ana @due(2026-11-03)` | A task with a due date, in Tasks and Agenda at once. Also `@scheduled(…)`, `@priority(high)`. |
| `total= 2 * 1250 EUR` then `= total to USD` | Calculations, with units and live currency rates. |
| Paste or drop an image | It is filed in your Library and linked. |

Changes save as you type. If someone else edits the same document (another tab, another person, your accomplice), you see their typing live ("Live" in the editor), and edits merge.

### The right panel

For the open document: **Links** and **Backlinks**, **Derived** fields its type computes, **Diagnostics** (fields that do not fit its type), and **Actions** its type offers. **Think through this with Alma** opens a conversation about it.

### More

- Types, views and type actions: Types, views and type actions (p. 46)
- Folders, moves and tidy-ups: Folders, moves and tidy-ups (p. 44)
- How the Docs vault updates: The Docs vault (p. 45)

## Library

How to

The Library holds every file that reaches Mandala: mail and Telegram attachments, what you upload or share from your phone, pictures pasted into documents, and what your accomplice makes (reports, charts, newsletters, pictures). Your accomplice files each one as it arrives.

### Find a file

- **Search files, people, tags…** matches names, folders, tags and senders.
- **Filters**: source, type, contact, date and size.
- **Clear everything** resets search and filters.

Select a file to preview it. **Why it's here** says where it came from and why it was filed where it is. Edit its folder (`Finance/Invoices`), tags (`invoice, october`) and sender.

### Add files

**Add files** uploads from your computer. On the iPhone app, **Share → Mandala** sends photos, PDFs and files to your accomplice and the Library.

### Let your accomplice organise

When files are not filed yet, **Let Alma organize** asks your accomplice to sort them into folders.

### Ownership

The Library belongs to an identity. Coming soon — one library per identity, following the Act as switcher.

## Personal

Partial How to

Personal is your own back office: groceries, money and health, with home and travel records underneath. Records belong to the identity you act as, so a House shopping list belongs to the House and everyone in it.

### Groceries

- **New list** — "Weekly groceries", "Hardware store".
- **Add something to buy**, with how much and the aisle. Items are grouped by aisle; checking one moves it to **In the basket**.
- **Ask Alma what's missing** — your accomplice suggests what you usually buy.

Lists sync between your devices and merge: two people adding at once never collide. Your accomplice adds to lists without asking, by default.

### Money

- **Accounts** — open one with a name, an optional bank and an opening balance.
- **Transactions** — record the amount, where or who, and a category. The month's list shows money in, out and the net.
- **Budgets** — a limit per category for the month, shown against what you spent.
- **Ask Alma about this month** — a conversation about your spending.

### Health

Your Apple Health data (Apple Health (p. 28)) as daily measurements:

- **Stands out** — what differs from your usual today, with **Ask Alma**.
- **Goals** — **Add goal** for a measurement (steps, sleep, weight…); your accomplice follows them in its daily digest.
- **Measurements** and **Workouts, last two weeks**.

### Behind it

Each record is governed by a contract from Mandala's personal library (`std.personal.*`): only its owner and the person who wrote it may change it. Your accomplice has a tool for every action and query. You sign what you do here yourself.

Coming soon — home and travel pages (the records already exist: home assets and maintenance, trips and bookings, appointments and medications), and importing bank statements.

## Dashboards

How to

Dashboards shows the **payables** of the identity you act as: every supplier invoice it has to pay, on a board from **Received** to **Reviewed**, **Approved** and **Paid**, or **Rejected**.

### Move an invoice

Drag a card to the next column, or click it and choose the next step. A card moves only by the step its rules allow: an invoice cannot be paid before it is approved, or paid twice. Paying asks for the **payment reference** (the bank transfer reference); rejecting asks for a reason.

Each step is signed by who took it, under your delegation when you act for a company ("Sam for Acme Trading, under delegation…").

Late invoices stand out, and totals are shown per currency.

### Get invoices in

- **Add an invoice** by hand: supplier, invoice number, amount, currency, issue and due dates, and a note.
- **Find invoices with Alma**: your accomplice looks through your mail for invoices and records them.
- **By signed exchange**: a structured invoice from a supplier you recognize arrives in *Received* by itself. See Exchange between businesses (p. 51).

The rules behind the board are the contract `std.business.payable` see Example contracts (p. 77).

Coming soon — operations boards for orders and deliveries, generated from your business's own contracts.

## Analytics

Analytics summarises the identity you act as, from its records and tasks:

- **Money, six months** — money out and in, month by month.
- **This month by category** — where it went.
- **Owed to suppliers** and **Late invoices** — from payables.

- **Suppliers, by amount invoiced** — who bills you most.
- Open and late tasks.

**What matters, with Alma** opens a conversation about the numbers. Analytics is read-only.

## Data

Partial How to

Data lists every DHL contract your node runs, by domain, with its records, queries and actions. It is the back office generated from your rules: no screen to design, and nothing it shows can bypass them.

### Look

Choose a contract. You see its fields with their types, its queries and its actions. Run a query for the identity you act as: rows appear in a table with dates, money, yes/no and identities formatted, references as links to the records they point to, and status badges. Search, sort columns and filter by value chips over the loaded rows.

### Act

Choose an action. Mandala builds its form from the contract (with sections, steps, defaults and help when the contract has a UI file; see UI files (p. 68)), shows a **preview** of exactly what will be signed, and signs it as you, for the identity you act as, under your delegation. If the rules refuse it, the refusal names the rule in words, on the field it concerns ("A product needs a SKU").

You always sign with your own key. Acting as a company without a delegation that covers the action is refused before anything is signed, with what to grant.

### Workbench

**Workbench: write a contract** opens the DHL Workbench: write a contract, try it on a throwaway node, and install it. See The Workbench (p. 74).

Coming soon — record pages with their history and related records, server-side tables with pickers, editing a record with its children in one step, and saved views.

## Tasks

Partial How to

A task is a line in a Goethe document:

```
- [ ] Call Seafrigo about the October invoice @due(2026-11-03) @priority(high)
```

**Tasks → To do** gathers every task from every document in one table.

### Capture

Type in **Add a task** and press **Enter**. One line understands dates and priorities:

| Write | Means |
|---|---|
| `today`, `tomorrow`, `friday` | Due that day (the next one) |
| `in 3d` | Due in three days |
| `2026-11-03` | Due on that date |
| `!high`, `!medium`, `!low` (or `!h`, `!1`…) | Priority |

"Call Seafrigo friday !high" becomes a high-priority task due Friday. New tasks go to your task document (`ALMA_TASK_DOCUMENT`, `docs/agenda.md` by default).

### Filter and group

- **Open**, **Done** or **All**.

- **When**: late, today, the next 7 days, or no date.
- **Document**, **priority**, and text.
- Group **by date** or **by document**, or switch to a **two-week timeline** and drag a task onto a day to schedule it.

Checking a task completes it in its document, with today's date. Click a task to open its document.

### Your accomplice and tasks

Anything your accomplice cannot finish now (waiting on someone, a promise, a date) becomes a task, and it says so. Ask "what's on my plate this week?" for a summary.

### Threads

**Threads** shows your accomplice's recent work as threads: each turn, what set it off, and the tools it used, like a compact Activity (p. 41).

Coming soon — the **Projects** boards on this page are a preview of a design for agent work and are not connected to real data yet. For coding sessions, use Code (p. 40).

## Code

How to

Code runs coding agents on your server, in your real repositories, several at once: **Claude Code**, **Codex**, and agents that speak the Agent Client Protocol (**Gemini CLI**, **Qwen Code**, **opencode**, **Claude Code (ACP)**). You start, watch, chat with and stop them from any device, your phone included, and your accomplice can steer them for you.

### Before you start

On the machine Mandala runs on, as the user Mandala runs as:

- install the agents you want and log in to them as you normally would; Mandala uses your installations as they are and injects no keys;
- install `git`, and `gh` (logged in) for pull requests and CI checks;
- optionally Docker, for sandboxed sessions.

### Start a session

1. Open **Code**. **Add a repository** with its path on the server (`/home/sam/src/shop`).
2. Under **New session**: choose the **Agent**, its **Permissions**, an optional **Title** ("Fix the sync retry handler"), and under **What to do** describe the goal, the constraints and how to check it.

| Permissions | Meaning |
|---|---|
| Asks you before other tools | The agent asks before any tool outside its safe set |
| Decides what is safe, asks the rest | The agent's judgement, asking when unsure |
| Its sandbox (writes only here), asks to go beyond | Codex's own sandbox (its default) |
| Never asks | Full autonomy inside the worktree |

3. Start. Each session works in its own git worktree, on its own branch (`alma/<title>-<id>`), so sessions never step on each other or on your checkout.

### While it works

- **Session** shows each step as it happens; write your next message at any time. **Changes** shows the diff; changes are committed after each turn.
- **Stop** a turn, or **Close** the session.
- Turns run detached: a Mandala restart does not stop them, and an interrupted turn resumes through the agent's own session.

### Questions become approvals

When an agent asks permission for something its mode does not allow, the question becomes an **approval card**: in the session, in your conversation, and as a notification with **Approve** and **Deny** on your phone. **Allow … for this session** answers all such questions for that tool. Your own words count too: on the Code page, a message sent while a question waits approves it or refuses it, and your words are passed to the agent. A card expires when its turn ends.

### Hand over to your accomplice

**Who steers this session**: you, or your accomplice. Handed over, it reads each finished turn and sends the next instruction, for at most 8 turns before you step in. Writing a message yourself takes it back.

### Pull requests and CI

**Open PR** opens the pull request with `gh` (only Mandala pushes). While it is open, the **Checks** tab follows its CI: about every minute, and at once when GitHub's webhook says a run finished (GitHub (p. 31)). When CI fails and the agent is idle, Mandala gives it a **fix turn** with the failing logs, then commits, pushes and checks again — one round per commit, up to 3 rounds (configurable per project). You are told once: when the rounds run out, when a fix changed nothing, or when the PR turns green.

Merging and deploying stay yours.

### Reviews

**Review** asks another agent to review the branch, read-only, against its base with the original brief and the diff (by default the other agent: Codex reviews Claude Code's work and the reverse). The review lands in the session; **Send to the agent** passes it on. It can also be posted as a PR comment.

### Sandbox

A project can run its turns and tests in a Docker container: your user, no extra privileges, 2 CPUs, 4 GB, optionally no network, with only the worktree, the agent and its login mounted, and no GitHub credentials. Turn it on per project (ask your accomplice, or `POST /api/code/projects/{id}`).

### Your accomplice and code

Your accomplice has `code.*` tools: it starts, reads, steers, reviews and closes sessions, and opens pull requests. It is woken when a session it steers finishes a turn.

## Agenda

How to

Agenda shows one calendar over every source: your Google calendars and every task with a date from your vaults.

- **Day** or **Week** view; move between weeks with the arrows. Each view has its own link.
- **Late** lists open tasks past their date; **Without a date** lists the rest. "Every task has a date. Nicely done." means you are clear.
- **Add a task…** adds a dated task.
- The **Tasks** tab lists open tasks beside the calendar.

Events come from the Google accounts whose Calendar service you allowed (Google Calendar, Drive, Docs and Sheets (p. 26)). Your accomplice reads the same agenda to plan your day and warn about clashes, and creates or moves events with your approval.

## Activity

Activity is your accomplice's journal, written by the system rather than by your accomplice: every turn in order, live.

Each entry shows:

- **What set it off** — your message, new mail in triage, a follow-up, the morning routine.

- **Where** — which conversation and channel.
- **What it did** — each tool it called, with arguments and result, including failures.
- **How it ended** — completed, waiting for your approval, failed (with the error) or cancelled.

Approvals appear with their decision: approved by you (a tap, a notification, your words) or by a *Without asking* setting. Use Activity when you want to know exactly what happened and why.

Coming soon — the **Ledger**: every act with which agent, for whom, under which grant, why (the goal or obligation), what it touched, and undo where possible.

## Initiatives

The Initiatives page is where you watch and steer your accomplice's own work towards your goals.

- **Goals** — each with its success measure and latest assessment. **Define goals with Alma** starts the goals interview; **Revisit with Alma** reopens one.
- **Initiatives** — proposed and running pieces of work, with their outcome so far. **Approve**, **Resume**, or **Send note**.
- **Reflections** — the latest reflections and diary entries, with questions to answer and **Useful** / **Not this** feedback.
- **Switches** — on/off (the kill switch), shadow mode, pause.
- **Tiles** — background tokens today against the daily limit, your conversation's tokens over 24 hours, work steps and reflections today, your accomplice's median answer time, and the share of suggestions you found useful.
- **Last 7 days** — reflections, steps and tokens per day.
- **Limits** — quiet hours and the daily caps.

How it all works: Goals, initiatives and reflection (p. 21).

## Pins

How to

A pin is a bookmark to anything: a conversation, an email thread, a Knowledge document or folder, a Library file, a Data record or saved view, a dashboard, a project or task, a settings page, an artifact your accomplice showed, or a web address. Pins live in the sidebar under **PINS**, in folders you arrange, and on the **Pins** page (`/pins`, or *All pins* in the sidebar) with search.

### Pinning

- **⌥P** (Alt+P) pins what is open.
- The **Pin** button on an email thread, a document or a Data record pins that thing; pressed again, it unpins.
- **⌘K** → *Pin this.*
- Drop a link on the sidebar's pins: an app link pins what it opens, any other becomes a web pin.
- Ask your accomplice: "pin this", "pin the Acme thread in Clients".

The same thing is never pinned twice.

### Arranging

Drag a pin to reorder it, onto a folder to put it inside, or between rows to place it. Each pin's ···menu (or a right-click) renames it, gives it a colour, moves it to a folder, makes it a superpin and removes it. Removing shows **Undo** for a few seconds; removing a folder removes what it holds. Folders open and close with their arrow, and remember it on every device.

### Superpins

Up to seven pins are **superpins**: always at the top of the sidebar, opened with **⌥⇧1** … **⌥⇧7**. Promote one from its menu or by dragging it to the top; a taken slot swaps.

### When a target moves or goes

A pin keeps what it points at by a stable id: the record's, the thread's, the file's. A pinned document gets a `uid` in its front matter (a normal, recorded edit), so it stays pinned when it is moved or renamed — in Mandala at once, and outside Mandala (another device, a sync, a file manager) the next time pins are shown. When the target is truly gone, the pin stays, marked **missing**, until you remove it.

### Tidying with your accomplice

Say "tidy my pins" (or *Ask … to tidy* on the Pins page). Your accomplice proposes folders, merges of duplicates and removal of missing pins, shown as the tree **before → after**. Nothing changes until you approve, by the card or in words; afterwards a *Pins tidied* note has **Undo**, or say "put it back".

### Whose pins

Pins belong to the identity you act as: your own pins and a company's are separate, and they sync to your other devices like every record.

## Settings

Open Settings with the gear at the bottom of the sidebar. **Overview** shows connections live, what needs attention (an expired sign-in, an extension not connected) and every page. Search filters the pages.

> The **Accomplice** group below is labelled **Alma** in current builds.

### Models

| Page | Explained in |
|---|---|
| Workload routing | Models and workload routing (p. 79) |
| OpenAI / ChatGPT, Anthropic, Google Gemini, xAI Grok, Meta, DeepSeek, Kimi, Z.ai, Ollama (local) | Choose a model (p. 10) |

### Identities

| Page | Explained in |
|---|---|
| Relationships and exchange | Exchange between businesses (p. 51) |
| Security | Users and permissions (p. 81), Join a device with a code (p. 57) |
| If you lose your devices | Keys and recovery (p. 56) |
| Devices and sync | Pair two nodes (p. 55), Sync and conflicts (p. 55) |
| Root keys | Keys and recovery (p. 56) |
| Identities and ownership | Identities (p. 48), Who owns what (p. 49) |

### Accomplice

| Page | Explained in |
|---|---|
| Constitution | Meet your accomplice (p. 12) |
| Standing rules | Memory and rules (p. 19) |
| Without asking | What your accomplice may do alone (p. 19) |
| Notifications | Notifications (p. 80) |
| iPhone and Mac apps | The iPhone app (p. 54), Notifications (p. 80) |
| Voice | Voice messages and calls (p. 17) |
| Updates | Updates (p. 83) |
| Conversation | Sensitive mode (p. 22) |

### Extensions

| Page | Explained in |
|---|---|
| Google | Set up your email (p. 25), Google Calendar, Drive, Docs and Sheets (p. 26) |
| WhatsApp | WhatsApp (p. 27) |
| Telegram | Telegram (p. 26) |
| Communications | Set up your email (p. 25) |
| Goethe | Knowledge (p. 36) |
| Brave Search, WaveSpeed images | Web search and pictures (p. 32) |
| Jev risk check (TypeSafe) | What your accomplice may do alone (p. 19), Privacy, connection by connection (p. 32) |
| GitHub | GitHub (p. 31) |
| Cloudflare | Cloudflare and webhooks (p. 30) |
| Inworld voice | Voice messages and calls (p. 17) |
| PostHog, Ahrefs, Meta Ads, MCP servers | PostHog (p. 29), Ahrefs (p. 29), Meta Ads (p. 29), MCP servers (p. 32) |
| Facebook & Instagram | Facebook and Instagram (p. 30) |
| Apple Health | Apple Health (p. 28) |
| Stored credentials | Every credential extensions use, by name only |

Most settings save with **Save changes** in the bar that appears when something changed; **Revert** discards.

## Folders, moves and tidy-ups

How to

Documents live in folders under `docs/`. Links survive every move: when a document moves or is renamed, every link to it, and every relative link from it, is rewritten so it still works.

### By hand

In Knowledge's sidebar: drag and drop, **Move to…**, rename (F2), duplicate, **New folder**, cut and paste (⌘X / ⌘V), and delete to **Archived** with **Restore**. **Undo** reverses any move, including the folders it created. See Knowledge (p. 36).

### Tidy up with your accomplice

For a messy vault, press **Tidy up** (or ask "tidy up my vault"). Your accomplice reads the vault and proposes a **plan**: a clearer folder structure, moves, renames, merges of duplicates, title and front-matter fixes, each with a short reason. It never touches `docs/diary/`.

The **Tidy-up plan** opens as **Before** and **After**, item by item (*New folder*, *Move*, *Retitle*, *Fix fields of*, *Archive…*):

- apply all items, or only those you keep;
- **Ask Alma to change the plan**
- **Discard** it.

After applying, **Undo this tidy-up** puts everything back, as long as nothing changed since.

### Asking directly

Your accomplice can also create folders, move, rename and duplicate documents when you ask. In its own vault it does so freely; in yours, or a company's, each reorganisation waits for your approval, with every move listed on the card.

## The Docs vault

This manual is a Goethe vault named **Docs**. It is an ordinary vault: open it from the vault picker in Knowledge, search it, link to its pages from your own documents (`[[connections/whatsapp]]` from inside it; from another vault, ask your accomplice for the link), and use its views and types.

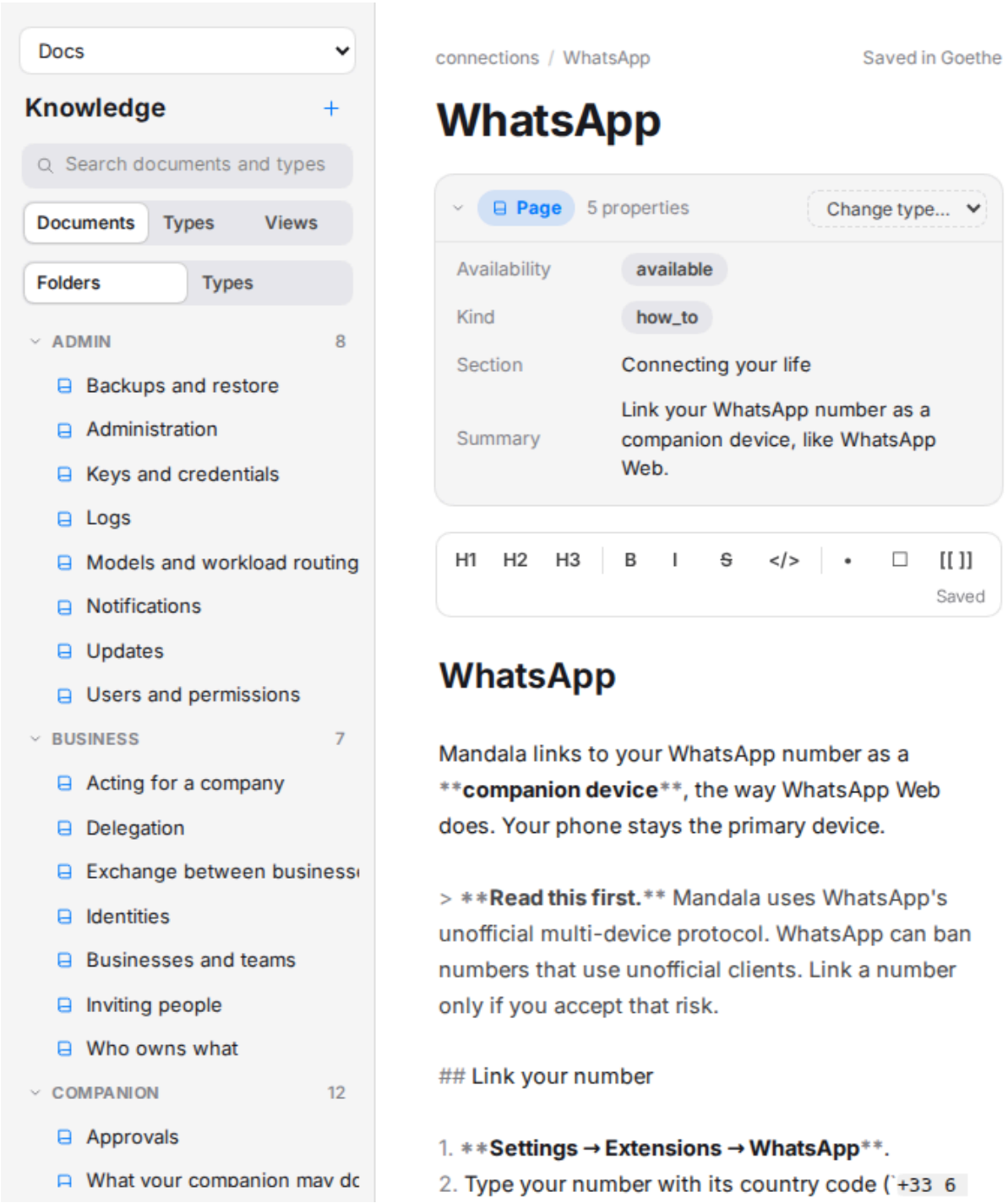



*Figure 1: The Docs vault open in Knowledge: its folders on the left, a page with its fields on the right*

### Where it comes from

Its pages ship inside the `mandala` program. Every node creates the vault on its first start, in `<data>/vaults/docs`, and refreshes it at each start from the release it runs. A new release brings new and updated pages with it.

### Your edits

You can edit any page or add your own. At each start:

| You… | Mandala… |
|---|---|
| left a page as shipped | replaces it with the new release's version |
| edited a page | keeps your version and does not overwrite it (the log lists it) |
| deleted a page | leaves it deleted |
| added a page | leaves it alone |

A page the new release removes is deleted only if you never edited it.

Two consequences: an edited page no longer receives updates (put your notes in a separate page that links to it, and the shipped page keeps updating), and the vault is **per node**. Docs is not synced between your devices, because each node writes it from its own release. Keep notes you want everywhere in your own vault.

### Your accomplice uses it

Your accomplice has the Docs vault's tools (`goethe.docs.search`, `goethe.docs.document.open` and the rest) in every conversation. When you ask how Mandala works or how to do something, it searches here, answers from the page and links it.

### Page fields

Each page is a `page` document with a `section`, a `kind` (tutorial, how-to, reference or explanation), an `availability` (available, partial or coming soon) and a one-line `summary`. The view **Coming soon** lists the pages about features still being built.

## Types, views and type actions

How to

### Types

A **type** says what fields a kind of document has: a contact has emails and phones, an invoice has an amount and a due date. A document declares its type in its front matter:

```
---
type: contact
title: Ana Reyes
emails: [ana@example.com]
company: Seafrigo
---
```

Types are written in Goethe's definition language, one file per type in the vault's `types/` folder:

```
type contact
    fields
        title: Text
        emails: List<Email>?
        phones: List<Text>?
        company: Text?
        status: Enum(active,former)?
```

`?` makes a field optional. A type that does not compile is not saved, so a vault never holds a broken type.

### Work with types in Knowledge

- **Types** in the sidebar lists every type with its documents. Select one to edit its definition, then **Save type**. **Refine this type with Alma** asks your accomplice to improve it.
- **Give a document a type**: open it, choose the type, and **Convert**. Every field is kept; Mandala suggests how to map old fields onto the type's (`email → emails`), and you can skip any.

- **Type with Alma** on a folder asks your accomplice to type every untyped document in it, creating a type if none fits.
- **Diagnostics** on a document lists fields that do not fit its type; **Add these fields to the type** fixes the type instead.

### Views

A **view** is a saved query over your tasks or documents, one file per view in `views/`:

```
view OpenSupplierTasks
tasks
| where status != "Done"
| where document_id == "docs/suppliers/seafrigo.md"
| sort due
| select text, due, document_id
```

The source is `tasks`, `agenda`, `documents`, or `documents <type>` for one type. Stages, one per line: `where` (strings in double quotes; a task's status is `Open` or `Done`), `sort`, `select`.

In Knowledge, **Views** lists them; select one to run it into a table (documents link through). **New view** creates one; it is saved only if Goethe compiles and runs it. Your accomplice can list, run and save views too.

### Type actions

A type can offer **actions** that appear on each of its documents. An action carried out by your accomplice has a prompt with the document's fields:

```
type supplier
    fields
        title: Text
        emails: List<Email>?
    actions
        ask_for_price_list:
            label = "Ask for the price list"
            transport = "alma"
            prompt = "Write to {title} at {emails} asking for their current price
list."
            confirm = true
```

Press the action on a supplier document and your accomplice does it, with its usual approvals. Actions that run commands (`stdio`, `http`) work only from the Goethe TUI, never on the server.

Coming soon — rules in vaults (Goethe's rule syntax is not defined yet).

# Businesses and teams

Partial

Mandala is personal first, and grows with you: the same home runs your groceries on Monday and a company on Tuesday. A company is not a separate account. It is an **identity you own**, with its own key, its own records and vault, and you act for it by **delegation**.

```
 you (person) ──owns 100%──► Acme Trading (organisation) ──recognizes──► Seafrigo
(supplier, another Mandala)
     │                              │
     │ delegation (*)               │ its vault, its payables, its mailbox
     ▼                              │
 your accomplice (agent) ──────────┘ acts for you, and through you for Acme, never beyond
your grants
```

## Pages

- Identities (p. 48) — persons, organisations, spaces and agents.
- Who owns what (p. 49) — who owns each account, vault and conversation.
- Delegation (p. 49) — acting for someone, within a scope and until a date.
- Acting for a company (p. 50) — the Act as switcher, and what changes.
- Exchange between businesses (p. 51) — signed documents between businesses, relationships and introductions.
- Inviting people (p. 50) — colleagues and household members Coming soon.

## What a business adds

- Its own **vault** for procedures, clients and suppliers. Knowledge (p. 36)
- Its own **records**, governed by contracts: payables today, your own rules tomorrow. Dashboards (p. 38), Building with DHL (p. 59)
- Its own **relationships** with suppliers and customers, and signed exchange with them. Exchange between businesses (p. 51)

Coming soon — speaking a business into being: describe how your company works, watch the rules and back office appear, try your own example on a throwaway node, and install; then the company's own agents (customer service, operations), each with its own name and seal.

## Identities

Everything in Mandala belongs to an identity, and every change is signed by one. An identity is a key pair plus a record of who it is.

| Kind | Example | Legal person | Owns things | Signs with |
|---|---|---|---|---|
| **Person** | You | Yes | Yes | Its own key |
| **Organisation** | Acme Trading | Yes | Yes | Its own key, by its owners' governance |
| **Space** | The House | No | Holds what its members share | Its members, under their grants |

| Kind | Example | Legal person | Owns things | Signs with |
|---|---|---|---|---|
| **Agent** | Your accomplice | No | No | Its own key, for whoever runs it |

- **Organisations are owned**, with shares if needed (you 60%, your partner 40%). Owners govern: they grant roles and change rules.
- **Spaces are functional, not legal.** A household shares groceries, bills and a vault, but owns nothing in law.
- **Agents are run** by a person or an organisation. Their grants never exceed their runner's.

### Create an organisation or a space

1. **Settings → Identities and ownership → New organization or space**.
2. Choose the **Kind** and a **Name**, then create.
3. For an organisation, **Add owner** with a share in percent. For a space, **Add member**.

Creating one gives you a full delegation to act for it. It then appears in the **Act as** switcher. Give it a vault from Knowledge when you act as it.

### Keys

Each identity's key is held by your node today. Moving organisation keys to their own servers, and owners' thresholds for sensitive changes, are Coming soon. Your personal root keys, which decide which devices are yours, are covered in Keys and recovery (p. 56).

## Who owns what

How to

Every mail account, chat account, vault and the Library has an **owner identity**. The owner decides who else sees it: a company's mailbox shows when you act as the company, and follows its permissions.

### Assign an owner

**Settings → Identities and ownership → Who owns what** lists each resource with its owner. Change the owner from the list.

### Move one conversation

A supplier who writes to your personal WhatsApp is still company business. In **Communications**, open the conversation and set **Belongs to** to the company. The thread now shows when you act as the company, while the rest of your personal account stays yours.

### Defaults

- Accounts you connect belong to you until you assign them.
- A vault created while acting as an identity belongs to it.
- Records belong to the identity they were written for.

## Delegation

Partial

A **delegation** lets one identity act for another. Nobody ever signs with someone else's key: you sign with yours, *for* the company, under a grant the rules check.

A delegation records:

| Field | Meaning |
|---|---|
| **Party** | Who is acted for (Acme Trading) |
| **Delegate** | Who acts (you, or your accomplice) |
| **Scope** | * for everything, a contract (`std.business.payable`), or one action (`std.business.payable.Pay`) |
| **Ends** | An optional end date: "it's like a job, it can end" |

### Where delegations come from

- **Creating** an organisation or space gives its creator a full delegation (*).
- At each start, Mandala makes sure you have one for yourself and your accomplice has one from you.
- An **owner** of an organisation, an **admin** of a space, or a **person for their own records** can grant one. The grantor or the delegate can revoke it.

### How it is checked

Twice. Mandala refuses before signing a read or an action for a party without a current covering grant, and says what to grant. Then the contract itself checks the grant: active, for that party, to that signer, with that scope, and not past its end. Each record keeps the delegation of its last action, so every receipt reads “Sam for Acme Trading, under delegation del-… (scope …, until …)”.

### Grant one today

Granting additional delegations is done through the API:

```
curl -X POST https://<your-node>/api/identities/acme-trading/delegations \
  -H 'content-type: application/json' \
  -d '{"scope": "std.business.payable", "ends_ms": 0}'
```

Coming soon — granting in words from Settings → People (“can answer the customer inbox, edit orders before dispatch; can’t see payables”), checked against the contracts, with Face ID for sensitive grants.

## Acting for a company

How to

### Switch

Click your name at the bottom of the sidebar and choose who you **act as**: you, a company you own, or a space you share. It is never a log-out.

### What follows

| Page | Shows |
|---|---|
| Communications | The accounts and conversations that belong to that identity |
| Knowledge | Its vault (or an offer to create one) |
| Personal, Dashboards, Analytics, Data | Its records |

Every action you take is signed by you, for that identity, under your delegation, and the preview and receipt say so.

### Your accomplice follows

Your accomplice is told whom you act as and works for that identity: it files a contact in the company’s vault and records an invoice in the company’s payables.

Coming soon — your accomplice’s tools restricted to the acting identity’s resources (today it is told, and the rules and delegations still apply to every record it writes).

## Inviting people

Coming soon How to

> Coming soon — this page describes the design being built. Today one person runs each Mandala home; organisations and spaces are acted for by their owner.

### How it will work

"Invite Maria to Acme Trading, she handles customer service." Your accomplice proposes her role **in words**, from the contracts: "can answer the customer inbox, edit orders before dispatch; can't see payables or change prices". You confirm with Face ID. Maria gets a link (email, WhatsApp or a QR code), valid 7 days.

Maria chooses:

- **New to Mandala**: her first name and Face ID. Her identity is created on your home, as one more person there, and the invitation says so in one line. She can move it to her own Mandala later.
- **She has Mandala**: she accepts from her own home, and the two homes exchange what her role allows.

Accepting creates *her* delegation from the company. Her personal space stays hers; leaving the company revokes the delegation, not her account. Each person gets their own accomplice, private to them, with the company's agents beside it as separate, labelled colleagues.

The household works the same way: "Add Claire to the House".

## Exchange between businesses

Partial How to

Two Mandalas can exchange signed documents directly: an invoice from a supplier to your company, a delivery proof to a customer. No email, no upload, no portal. Your node checks the signature and decides, from your own records, whether to route the document into your work or hold it for review.

### Your exchange key

Each identity your node holds has an **exchange key**. **Settings → Relationships and exchange** (acting as that identity) shows it. Give it, with your node's address, to whoever should be able to send you documents.

### Recognize a counterpart

Under **Recognize a counterpart**:

1. **Name** — "Seafrigo".
2. **Its exchange key** — 64 hex characters, from their Settings.
3. **As** — supplier, customer, employer, endorser, introducer or partner.
4. **Its address** — their Mandala's URL.
5. **Route straight to work** — the kinds of document you trust from it, separated by spaces (`invoice delivery_proof`).

Then **Recognize**. **End** a relationship at any time.

### Receiving

| A document… | Happens |
|---|---|
| With a bad signature | Refused before anything is recorded |
| From a recognized counterpart, of a trusted kind | **Routed** into your work (a structured invoice lands in Payables → Received) |
| Anything else | Waits under **To settle** for you to **Accept** or **Refuse** |

Each document records who sent it and who acted for the sender.

### Sending

**Send a document**: choose the recipient, the **Kind** (`invoice`), a **Title** and the **Content**, then **Sign and send**.

### Introductions

**Introduce** tells one counterpart how to reach another. Only they learn of it. Accepting an introduction recognizes whom it introduces, trusted with nothing until you say so.

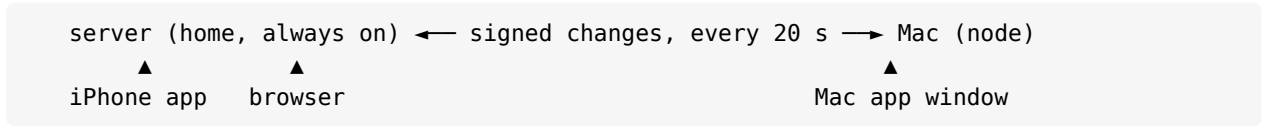
**Not yet**

Coming soon — discovering identities through the public network (today addresses are exchanged by hand), proof of who acted for the sender checked by the recipient, and routed documents starting workflows (a supplier invoice going to finance review).

# Devices and sync

Partial

Your Mandala can live on several machines at once: a server, your Mac, a laptop. They are the same Mandala, not copies of each other. Each is a **node** with its own key; paired nodes sync every 20 seconds and converge on the same records and vaults. Your phone and your browsers are **clients** of a node.




## Pages


- Install the web app on your phone (p. 53) — Mandala on your phone's Home Screen, with notifications.
- The iPhone app (p. 54) — the native iPhone app.
- The Mac app as a second node (p. 54) — the Mac app as a second node.
- Pair two nodes (p. 55) — pair two nodes, and forget one.
- Sync and conflicts (p. 55) — how changes merge, and what to do when they don't.
- Keys and recovery (p. 56) — what decides which devices are yours, and how to get back in.
- Join a device with a code (p. 57) — add a phone, computer or browser with a temporary QR or short code.


## Offline

A node accepts your changes offline and syncs when it can reach the others. Records whose rules allow it merge automatically; the rest become conflicts you settle, never silent losses.

## Install the web app on your phone

How to

Mandala is an installable web app. On your phone it gets its own icon and window, opens to where you left it, and receives notifications.

### iPhone

1. Make sure **Tailscale** is connected on the iPhone.
2. Open your Mandala address (`https://<machine>.<tailnet>.ts.net`) in **Safari**.
3. **Share → Add to Home Screen → Add**.
4. Open Mandala **from the Home Screen icon**.
5. **Settings → Accomplice → Notifications → Turn on**, and allow notifications. On an iPhone this works only from the Home Screen app.
6. **Send a test** to check.

### Android

Open the address in Chrome, then ⋮ → **Install app** (or *Add to Home screen*). Turn on notifications in Settings → Accomplice → Notifications.

### Good to know

- Offline, the app opens to its last screen, but shows no data: nothing is cached on the phone.

- The address must be HTTPS for installing and notifications. See Tailscale and HTTPS (p. 88).

## The iPhone app

Coming soon

> Coming soon — the iPhone app is built and being tested; TestFlight and the App Store follow. Until then, use the web app on your Home Screen: Install the web app on your phone (p. 53).

The iPhone app shows your Workspace and adds what only a native app can do:

- **Notifications with actions.** Approval requests arrive with **Approve** and **Deny** Face ID first. Screenshots ride along when your accomplice uses a computer.
- **Send to your accomplice** from the share sheet: links, text, photos, videos, PDFs and files, with a note.
- **Apple Health**, read in the background (18 daily metrics, sleep and workouts) and sent to your home, replacing Health Auto Export. Apple Health (p. 28)
- **Siri and Shortcuts**: "Tell Mandala …".
- **App settings**: your home's address, notifications and Health.

The phone is always a client of your home: it never creates an identity and it holds no node. It needs **Tailscale** connected to reach your home.

### For administrators

Apple push needs an APNs key from your Apple Developer account, uploaded in **Settings → Accomplice → iPhone and Mac apps** with its Key ID and Team ID. The key stays on your server. The same page lists the apps that receive notifications and sends a test. See Notifications (p. 80).

Coming soon after that — native calls with the system call screen, reaching your home without Tailscale, and widgets.

## The Mac app as a second node

Partial How to

Mandala for Mac joins your home. Its window shows your Mandala from your home, and your settings are copied to this Mac. Installing it: Install on a Mac (p. 8).

### Join your home

The simplest way: on the Mac app's first screen, choose **Join your identity** and use a code from a device you already use (Join a device with a code (p. 57)).

Or from the menu bar:

1. On a device you already use, **Settings → Security → Add a device** shows a code and a QR. In the Mac app's menu bar item, choose **Join your identity…** and enter that code.
2. The Mac joins your home, directly between your devices. Your settings are copied to the Mac; its window shows your Mandala from your home.
3. If you have set up root keys, **certify the Mac**: on your home's Workspace, **Settings → Root keys → Devices → Certify**, with your paper key or a security key. Until certified, a device is not trusted once a root exists.

If this Mac already has its own Mandala, it must start over before joining. Quit Mandala, move `~/Library/Application Support/Mandala/data` aside to keep its current data, open Mandala again, and join your home.

### Use it

- A joined Mac shows **Shown from your home** in the menu bar item. If your home is out of reach, the window says so and keeps trying every few seconds.
- **Show: Home** / **Show: This Mac** chooses what the window shows only on a Mac that has not joined.

- **Start at login** keeps your Mac's node running.
- Closing the window keeps Mandala in the menu bar; **⌘Q** quits cleanly.

Coming soon — a full copy of your Mandala on this Mac and offline use.

Coming soon — native Mac notifications, and your accomplice using your Mac's apps when you ask (its own labelled cursor; touching anything stops it).

## Pair two nodes

How to

Pairing makes two nodes (a server and a laptop, two servers) one Mandala.

> The simplest way to add a node is to **join it with a code** from its first screen: Join a device with a code (p. 57). This page pairs two nodes by hand, by device key, for nodes that already have data or no screen.

Pair on **both** sides.

### 1. Get each device's key

On each node, open **Settings → Devices and sync**. **This device → Device key** shows 64 hex characters. Copy it.

### 2. Pair from each side

On node A, under **Pair a device**:

- **Name** — "MacBook Pro".
- **Its device key** — node B's key.
- **Where to reach it** — one of:
  - `iroh`: find it anywhere by its key, through the device mesh (direct connections with hole punching; see Ports and network (p. 88));
  - its address on your tailnet: `http://macbook.<tailnet>.ts.net:8787`
  - empty, if only B can reach A (a laptop behind a home router pairing with a server).

Press **Pair**. Then do the same on node B with node A's key.

### 3. Check

Under **Your other devices**, the peer shows **Syncing** (this node visits it) or **Trusted** (it visits this node), and how many of its changes are held here. Changes appear on the other node within about 20 seconds.

### What pairing does

Pairing **enrolls** the device as yours: the enrollment record syncs, so every one of your nodes trusts it, including nodes it never paired with directly. Each pull is signed by the device's key, because changes carry personal data.

### Forget a device

**Forget** revokes it everywhere: your nodes refuse its new changes. Do this at once for a lost or sold machine.

### With root keys

Once you have a root key, only devices it **certified** are trusted. Certify a new device in **Settings → Root keys → Devices**. See Keys and recovery (p. 56).

## Sync and conflicts

### How sync works

Every change accepted on a node is appended to that node's own **log of signed changes**, stamped with a clock and signed by the device key. Paired nodes send each other what the other lacks every 20 seconds, verify every

signature, and apply everything in the same order. Same changes, same order: every node ends up the same. A restart resumes where it stopped.

Your **vaults** sync on the same log, as text that merges: two devices editing the same document at once keep both edits, with no conflict markers. Front-matter fields merge one by one, the later change of a field winning. (The Docs vault is the exception: each node writes it from its own release; see The Docs vault (p. 45).)

### How records merge

Each contract says, action by action, how concurrent changes merge:

| Rule | Meaning | Example |
|---|---|---|
| **commutes** | Both apply, in any order | Two people adding to the same grocery list |
| **last_writer** | The later one wins | Renaming a list on two devices |
| **exclusive** | Only one can apply; the other is a conflict | Approving or paying the same invoice twice |

Actions that declare nothing are **exclusive**, so nothing merges silently that wasn't designed to. Contract authors: Merge rules for sync (p. 71).

### Didn't merge

When a change from another device no longer applies here (the same thing changed on both while apart), it is kept as a **conflict**. Nothing is dropped.

**Settings → Devices and sync → Didn't merge** lists each one: the action, the record, which device sent it, and why it didn't apply. Open **What it carried** to see its arguments, do what is right (redo it, or not), then write what you did ("Kept this device's version") and press **Settle**.

### Limits today

- Your accomplice's own records (conversations, memory, approvals) do not sync between nodes yet. Coming soon
- Records created before a node kept its log do not travel.
- A move of a document syncs as a delete and a create.

## Keys and recovery

Partial How to

Your **root** decides which devices are yours: only devices it certified are trusted, by every one of your nodes, so a stolen server cannot add a device to your identity. When you create your identity at the first run, your **passkey** (Face ID) becomes your root's first authority.

### If you lose your devices

**Settings → If you lose your devices** shows your ways back in plain sentences, and how strong they are together: one way back is *weak*, two *good*, three or more *strong*.

| Way back | Set it up | Getting back in |
|---|---|---|
| **Your synced passkey** | Use a passkey saved in iCloud Keychain or Google Password Manager (Security → Add) | Face ID on the new phone or computer, as usual |
| **Another device** | Use Mandala on a second device (Security → Add a device) | Approve the new device from the other one |
| **People you trust** | Pick 2–5 people (Root keys → Trustees) | Enough of them vouch for you, after talking to you |
| **Recovery kit** | Your 24-word paper key (Root keys → Paper key) | Type the words on the new device |
| **This machine's console** | Coming soon | On your own server or Mac, owning the machine will be a way back |

If every way back is lost, nobody, including Mandala's developers, can give the account back. Your data stays on your machines.

**Advanced: root keys**

**Settings → Root keys** manages the root directly: the paper key, security keys and trustees. A node set up before accounts can create its root here.

**1. Paper key (the recovery kit)**

24 words that can act as your root. Press **Create**: the words are generated in your browser, shown once and never stored, not even on your server. Write them on paper, keep the paper somewhere safe, then type back the words asked to prove you have them.

**2. Security keys and passkeys (up to 3)**

A passkey or a hardware security key (YubiKey and similar) registered here can certify devices and change the root with a touch, like the paper key. Keys are bound to your tailnet address, so one key works with every node.

**3. Trustees (optional, up to 5)**

People you recognize in **Relationships and exchange**. If you lose your paper and security keys, enough of them together can vouch for a new root. With several trustees, at least two must agree; you choose how many.

**4. Create**

**Create root** (only on a node without one). From then on:

- **Certify** each device under **Devices**, with the paper key or a security key. Uncertified devices stop syncing until certified.
- **Change the root** (add or remove security keys, change trustees): the current root signs the change.

**If you lose your keys**

**Recover with trustees**: Mandala prepares a recovery request. Write down the new paper key it shows first. Send the request to each trustee; each pastes it in their **Settings → Root keys → Vouch for someone**, *after checking with you in person or by a call*, and sends back their answer. Paste the answers, one per line, and **Recover**.

As a trustee, vouch only after you have spoken with the person: whoever controls their root controls their devices.

**Coming soon**

Coming soon — trustee requests and answers through Exchange instead of copy and paste, a printable recovery kit with your seal and a QR, and console recovery on your own machines.

## Join a device with a code

How to

**1. On a device you already use**

**Settings → Security → Add a device** (or open `/add-device`). Mandala shows a temporary **QR code** and a short **code** (like `K7QD-4M2P`). It lasts 5 minutes and works once; **New code** makes another.

**2. On the new device**

Open your Mandala's address and choose **Join your identity** (*I already use Mandala on another device*), then:

- if this device's keychain already holds your passkey, **Continue with Face ID** and you're in;
- otherwise **Use a code**: scan the QR, or type the code. A typo is caught at once ("one character looks wrong").

The new device shows **Joining Sam** with your **seal**. Check it is yours.

**3. Approve on the trusted device**

It asks "Add this … as you? It will see what you see and act as you." — press **Add with Face ID**, or **Not me**. Then **Welcome back**: the new device lands where you last were.

**Joining a node (a Mac or a second server)**

A new Mandala node — the Mac app, or a server you just installed — joins the same way, from its own first screen: **Join your identity → Use a code**. The two nodes find each other through the device mesh (directly, on the local network while the code lives, or through your home's address, which you can type under **Your home's address (optional)** when the new one is on another network), check the code without sending it, and the trusted side signs the new node's certificate with Face ID. If Mandala asks, restart it on the new device to finish.

**Safety**

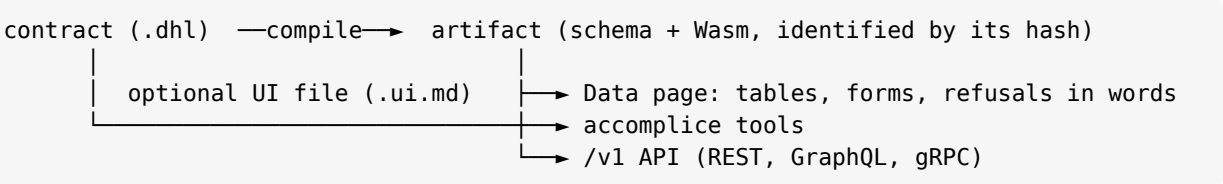
- A code works once and lasts 5 minutes; after repeated wrong attempts, adding devices pauses for 10 minutes.
- Nothing is shared until you approve on the trusted side. If you didn't start it, press **Not me**, then check **Devices and browsers** in Security.
- Lost every device? See Keys and recovery (p. 56).

# Building with DHL

DHL is the language Mandala uses to write down how a business works: what a record is, who may change it, how, and when it must refuse. A **contract** in DHL is short, readable Markdown, and the engine enforces it on every change, whoever asks: you, a colleague, an API client or your accomplice.

From one contract Mandala derives:

- the **records** and their history, each change signed by who made it;
- the **back office** in Data, with forms, previews and refusals in words;
- your accomplice's **tools**, one per action and query;
- a generated **HTTP, GraphQL and gRPC API**
- how changes **merge** when several devices work offline.




## Learn

1. DHL concepts (p. 59) — contracts, records, actions, queries, signers.
2. Tutorial: your first contract (p. 60) — write, try and install your first contract in 20 minutes.

## Reference

- DHL language reference (p. 63) — the file format and every block.
- DHL types (p. 66) — the value types.
- DHLQ query reference (p. 67) — the query language.
- UI files (p. 68) — labels, formats, forms and refusals in words.
- Durability classes (p. 70) — how soon a write is on disk.
- Merge rules for sync (p. 71) — how concurrent changes merge.
- Upgrading a contract (p. 72) — changing a contract that already has records.
- The generated API (p. 73) — the HTTP API every contract gets.

## Practice

- The Workbench (p. 74) — write, try and install from the app.
- Testing contracts (p. 75) — check that the rules accept and refuse what they should.
- DHL best practices (p. 76) — patterns that age well.
- Example contracts (p. 77) — real contracts to read and copy.

## DHL concepts

### Contract

A contract describes **one kind of record**: its state, the actions that change it, and the queries that read it. It is a Markdown file with front matter and fenced `dhl` blocks; prose around the blocks is for people.

````
---
namespace: sam.tools.loan
version: 1.0.0
---

A tool lent to someone, until it comes back.

```dhl
aggregate Loan
    state
        public item: Text(len=200)
        public status: Text(len=20)
```
````

The **namespace** names the contract (`std.personal.shopping_list`, `com.acme.orders.order`). Contracts in a repository live at the path of their namespace: `contracts/std/personal/shopping_list.dhl`.

### Artifact

Compiling a contract produces an **artifact**: its schema and its WebAssembly code, identified by the hash of both. The artifact, not the namespace, is what runs. Changing a rule changes the artifact; records written under the old one keep their history and move to the new one through a signed upgrade (Upgrading a contract (p. 72)).

### Record (subject)

One instance of a contract: one loan, one invoice. A record has an **id** (its subject), a revision, its state, and its full history of signed changes. Records belong to a **tenant**, the identity they live in.

### Action

The only way a record changes. An action takes typed arguments, **validates** them against the record's current state and its context (who signs, what time it is, other records it observes), then **applies** changes to the state. If any validation line is false, the whole action is refused and nothing changes. A refusal is recorded too, with its reason.

### Signer

Every action is signed. `context.signer` is the identity whose key signed it. Rules decide what the signer may do: "only the owner may return a loan" is `context.signer == state.owner`. In Mandala, people sign with their own key and act for organisations by **delegation**, which contracts can check (Delegation (p. 49)).

### Query

A named, typed read over the records of a contract, written in **DHLQ**: filter, sort, page, select, aggregate (DHLQ query reference (p. 67)). Queries are read-only.

### Composition

An action can **require** actions on other records (`requires`): placing an order opens each order line and reserves stock, all signed by the original signer and all committed together, or none. An action can also **observe** another record's state, such as a delegation, as evidence.

### Where contracts come from

- **Built into Mandala**: the standard library (`std.identity`, `std.personal`, `std.business`, `std.exchange`…), compiled into every node.
- **Configured**: files or folders in `MANDALA_CONTRACTS`.
- **Installed from the Workbench**: saved in `<data>/contracts`.

## Tutorial: your first contract

Tutorial

You lend tools to friends and forget who has what. In this tutorial you write a small contract for it: a **loan** is created when you lend something, and closed when it comes back. Along the way you meet every part of DHL you need daily: state, actions, validation, queries, merge rules, the Workbench and your accomplice.

Open **Data → Workbench: write a contract**.

### 1. Describe the record

Replace the template with:

````
---
namespace: sam.tools.loan
version: 1.0.0
merge: Lend=exclusive, Return=exclusive
---

# A tool lent to someone, until it comes back.

```dhl
struct LoanRow
    id: Identity
    ref: Text(len=100)
    item: Text(len=200)
    borrower: Text(len=200)

aggregate Loan
    state
        public ref: Text(len=100)
        public owner: Identity
        public item: Text(len=200)
        public borrower: Text(len=200)
        public status: Text(len=20)
        public revision: Int = 0
```
````

Change `sam` in the namespace to your own handle. The `aggregate` is what one loan holds; `LoanRow` is what a query returns for each loan.

### 2. Lend

Below, add an action:

````
Lending something out.

```dhl
action Lend(ref: Text(len=100), owner: Identity, item: Text(len=200), borrower:
Text(len=200))
    api authenticated
    authorize dhl
    validate
        state.revision == 0
        context.signer == owner
        item.len > 0
        borrower.len > 0
    apply
        state.ref = ref
        state.owner = owner
        state.item = item
        state.borrower = borrower
        state.status = "out"
```
````

```
        state.revision = 1
```

Read the validation lines aloud: *this loan is new; whoever signs is its owner; there is an item; there is a borrower.* If any is false, nothing happens.

**3. Return**

```
It came back.

```dhl
action Return(owner: Identity)
    api authenticated
    authorize dhl
    validate
        state.status == "out"
        context.signer == owner
        owner == state.owner
    apply
        state.status = "back"
        state.revision = state.revision + 1
```
```

Only an item that is out can come back, and only its owner can say so.

**4. Ask what is out**

```
```dhl
query Out(owner: Identity, after: Text(len=100))
    returns List<LoanRow>
    api authenticated
    dhlq
    sam.tools.loan
    | where owner == $owner and status == "out" and ref > $after
    | sort ref
    | take 100
    | select subject as id, ref, item, borrower
```
```

Use your namespace on the line after `dhlq`.

**5. Try it**

In **Steps and queries**, write a story with both happy and unhappy moments. `"$julien"` stands for your own key (`julien` is the internal handle of the person who set up the home):

```
{
  "steps": [
    {"action": "Lend", "subject": "drill", "args": {"ref": "drill", "owner": "$julien",
"item": "Cordless drill", "borrower": "Ana"}},
    {"action": "Lend", "subject": "ladder", "args": {"ref": "ladder", "owner":
"$julien", "item": "Ladder", "borrower": "Ben"}},
    {"action": "Lend", "subject": "drill", "args": {"ref": "drill", "owner": "$julien",
"item": "Cordless drill", "borrower": "Chloé"}},
    {"action": "Return", "subject": "drill", "args": {"owner": "$julien"}},
    {"action": "Return", "subject": "drill", "args": {"owner": "$julien"}}
  ],
```

```
  "queries": [{"name": "Out", "params": {"owner": "$julien", "after": ""}}]
}
```

Press **Try it**. You should see:

```
✓ Lend drill      {"status": "out", "borrower": "Ana", …}
✓ Lend ladder     {"status": "out", "borrower": "Ben", …}
✗ Lend drill — DHL refused Lend: its rule `state.revision == 0` does not hold
✓ Return drill    {"status": "back", …}
✗ Return drill — DHL refused Return: its rule `state.status == "out"` does not hold
Out  [{"ref": "ladder", "item": "Ladder", "borrower": "Ben", …}]
```

The drill cannot be lent twice, and cannot come back twice. The ladder is still out. All of this ran on a throwaway node; your real records are untouched.

**6. Install it**

Press **Install…** then **Install and restart**. Mandala restarts in a few seconds. Now:

- **Data** lists your contract; run **Out**, or **Lend** from its form, with a preview before signing.
- Your accomplice has `Lend`, `Return` and `Out` as tools. Ask it: "I lent the hedge trimmer to Ben" — it prepares the action and asks you to approve it. "Who has my tools?" — it runs the query.

**7. Make it yours**

Ideas, each a small change you can try in the Workbench first:

- Add `due: Timestamp?` to the state and an argument to `Lend`, and a query for overdue loans.
- Give it a UI file (`loan.ui.md`) with labels and a refusal sentence for each rule (UI files (p. 68)).
- Once loans exist, changing the state needs an upgrade action (Upgrading a contract (p. 72)).

**What you learned**

| Concept | In this contract |
|---|---|
| State | `aggregate Loan` |
| Actions and validation | `Lend`, `Return`, one rule per line |
| Signer | `context.signer == owner` |
| Queries | `Out`, filtered, sorted, paged |
| Merge rules | `merge: Lend=exclusive, Return=exclusive` |
| Trying safely | The Workbench's throwaway node |

Next: DHL language reference (p. 63) for everything DHL can say, and DHL best practices (p. 76).

## DHL language reference

**The file**

A contract is Markdown. Front matter first, then any mix of prose and fenced ```` ```dhl ```` blocks; the blocks are read in order as one program. Indentation is **four spaces**. `#` starts a comment line inside a block.

```
---
namespace: sam.tools.loan
version: 1.0.0
merge: Lend=exclusive, Return=exclusive
durability: confirmed
---

Prose for people.
```

```dhl
aggregate Loan
    state
        public item: Text(len=200)
```

**Front matter**

| Key | Required | Meaning |
|---|---|---|
| `namespace` | yes | The contract's name, dotted, lowercase (`com.acme.orders.order`). Names starting `mandala.` are reserved. |
| `version` | yes | A semantic version for people; the artifact hash is what identifies behavior. |
| `merge` | no | How concurrent changes from different devices merge, per action: `commutes`, `last_writer` or `exclusive` (the default). See Merge rules for sync (p. 71). |
| `durability` | no | `confirmed` (default), `grouped`, or `buffered(100ms)`. See Durability classes (p. 70). |

Neither `merge` nor `durability` changes the artifact.

**struct**

A named record type for arguments, list items and query rows.

```
struct LoanRow
    id: Identity
    item: Text(len=200)
    due: Timestamp?
```

**aggregate**

The record's state: one `state` block of fields, each `public` or `private`, with an optional default.

```
aggregate Loan
    state
        public owner: Identity
        public item: Text(len=200)
        public status: Text(len=20)
        public revision: Int = 0
        private note: Text(len=500)
```

`public` fields can be read by queries and clients; `private` fields only by the contract's own rules. A field without a default starts at its type's zero value.

**external**

What an action may read from outside its own state:

```
external
    time: [clock_ms]
    identity: [root_subject]
    datasets: [lines: List<SubjectRef<CartLine>>(capacity=16) from CartLines]
```

| Entry | Gives |
|---|---|
| `time: [clock_ms]` | `context.clock.milliseconds`, the decision's clock |
| `identity: [root_subject]` | `context.root_subject` |
| `datasets: [name: Type from Query]` | `dataset.name`, the result of one of this contract's queries |

### action

```
action Return(owner: Identity, note: Text(len=500))
    api authenticated
    authorize dhl
    validate
        state.status == "out"
        context.signer == owner
        owner == state.owner
    apply
        state.status = "back"
        state.note = note
        state.revision = state.revision + 1
```

| Line or block | Meaning |
|---|---|
| `api authenticated` / `api public` | Exposes the action in the generated API. Without it, the action is internal. |
| `authorize dhl` | The contract's own rules decide (the usual choice). |
| `observe <ref> where <field> == arg.<name>` | Reads another record as evidence: the action is refused unless that record matches. Used to check delegations. |
| `requires` | Actions on other records that must succeed with this one, all or nothing. |
| `validate` | One condition per line. All must be true, or the action is refused and nothing changes. |
| `apply` | Assignments to `state.*`, in order. |
| `certify upgrade from <artifact id>` + `migrate` | Makes the action an upgrade from an archived version. See Upgrading a contract (p. 72). |

### In expressions

| Name | Is |
|---|---|
| `state.<field>` | The record's current state |
| `<arg>` or `arg.<arg>` | An argument |
| `context.signer` | The identity that signed |
| `context.clock.milliseconds` | The decision's time (with `external time`) |
| `dataset.<name>` | An external dataset |
| `self` | This record (in `requires`) |

Operators: `==`, `!=`, `<`, `<=`, `>`, `>=`, `and` / `&&`, `or` / `||`, `not`, `in`, `+`, `-`, `*`, `/`, `%`. Text and lists have `.len` `contains(list, value)` tests membership. Strings use double quotes.

### requires

```
action AddLine(line: SubjectRef<CartLine>, product_id: Identity, quantity: Int)
    api authenticated
    authorize dhl
    requires
        line.Open(cart=self, buyer=state.buyer, product_id=arg.product_id,
quantity=arg.quantity)
    validate
        quantity > 0
```

Each required action runs on its own record, is authorized by that record's contract, and is signed by the original signer. `for entry in entries` repeats one for each item of a list argument.

**observe (delegation)**

```
    observe delegation where active == true
    observe delegation where party == arg.owner
    observe delegation where delegate == arg.writer
    observe delegation where scope == arg.delegation_scope
```

The action carries a `delegation: SubjectRef<Delegation>` argument; DHL checks that record's state matches each line. This is how `std.business.payable` accepts "Sam for Acme Trading".

**query**

```
query Out(owner: Identity, after: Text(len=100))
    returns List<LoanRow>
    api authenticated
    dhlq
    sam.tools.loan
    | where owner == $owner and status == "out" and ref > $after
    | sort ref
    | take 100
    | select subject as id, ref, item, status
```

`returns` names the row type; `dhlq` starts the query, whose first line is its source (`self` or a namespace). Arguments are `$name`. See DHLQ query reference (p. 67).

**Compile errors**

The compiler refuses unknown section keywords, types that don't fit, assignments to unknown fields and queries over unknown fields, with the line. Try in the The Workbench (p. 74) until it compiles.

## DHL types

| Type | Holds | Notes |
|---|---|---|
| `Int` | A signed integer | Money as an integer of minor units (`amount_minor`) |
| `Decimal`, `Decimal(scale=2)` | A fixed-point decimal | JSON carries it as a string |
| `Ratio` | A ratio | |
| `Bool` | `true` or `false` | |
| `Text`, `Text(len=200)` | Text, at most `len` bytes | Always give a length |
| `Email` | An email address | |
| `Currency` | A three-letter currency code (`"EUR"`) | Contracts holding money cannot be `buffered` |
| `Timestamp` | A point in time | |
| `Duration` | A length of time | |
| `Identity` | A 32-byte identity (a person, organisation, space, agent or record id) | Lowercase hex in JSON |
| `GeoPoint` | A latitude and longitude | |
| `Enum(a,b,c)` | One of the listed names | |
| `Ref<Aggregate>` | A reference to a record | |
| `SubjectRef<Aggregate>` | A reference to a record at a revision (`{id, seq}`) | Used for delegation evidence and child records |
| `Struct<Name>` | A struct value | |

| Type | Holds | Notes |
|---|---|---|
| `List<T>`, `List<T>(capacity=16)` | A list | Declare a capacity for lists carried by commands |
| `Map<K, V>`, `Map<K, V>(capacity=N)` | A map | |
| `T?` | Optional | `Int?`, `Timestamp?` |

### In JSON

Integers, timestamps, durations and decimals travel as **decimal strings** so nothing loses precision in JavaScript. Identities are lowercase hex. A `SubjectRef` is `{"id": "…", "seq": "…"}`. Keep these representations in clients.

## DHLQ query reference

A DHLQ query is a source followed by stages, one per line, each starting with `|`.

```
query Late(owner: Identity, today: Text(len=10))
    returns List<PayableRow>
    api authenticated
    dhlq
    std.business.payable
    | where owner == $owner and status != "paid" and due_on < $today
    | sort due_on, -amount_minor
    | take 50
    | select subject as id, supplier, amount_minor, currency, due_on
```

Arguments are written `$name`.

### Sources

| Source | Reads |
|---|---|
| `self` | This contract's records |
| `std.business.payable` | Another contract's records, by namespace |
| `search $q in ns.field, ns.other fuzz=1` | Full-text search over fields |

Every record row has `subject` (its id) and `seq` (its revision) besides its public fields.

### Stages

| Stage | Example | Does |
|---|---|---|
| `where` | `where owner == $owner and active == true` | Keeps matching rows |
| `validate` | `validate $limit > 0 and $limit <= 100` | Checks arguments once; refuses the query otherwise |
| `let` | `let remaining = received - reserved` | Adds a computed field |
| `sort` | `sort due_on, -amount_minor` | Sorts; `-` for descending |
| `drop` | `drop 20` | Skips rows |
| `take` | `take 100` | Keeps at most this many |
| `select` (or `sel`) | `select subject as id, ref, title` | Chooses and renames columns |
| `join` (or `lj`) | `join std.crm.contact as c on supplier_id = c.subject` | Joins another contract's rows |
| `explode` | `explode tags as tag` | One row per list item |
| `group by` (or `by`) | `group by currency` | Groups rows |

| Stage | Example | Does |
|---|---|---|
| `agg` | `agg count() as n, sum(amount_minor) as total` | Aggregates: `count`, `sum`, `avg`, `min`, `max` |
| `bucket` | `bucket created_at 1d as day` | Groups timestamps into intervals |

### Totals

```
dhlq
self
| where cart_subject.id == $cart_id and quantity > 0
| agg count() as line_count, sum(total_minor) as total_minor
```

### Per group

```
dhlq
std.personal.finance_transaction
| where owner == $owner
| group by category
| agg sum(amount_minor) as spent, count() as n
```

### Paging

Page by a stable key rather than by offset: pass the last key you saw and filter on it.

```
| where owner == $owner and ref > $after
| sort ref
| take 100
```

### What the engine maintains

Queries over records are answered from projections the engine keeps up to date. Projections support `where`, `sort`, `drop`, `take`, `select`, `let`, `validate`, `group by`, `agg` and `bucket`. A query using `search`, `join` or `explode` is part of the language, but the engine refuses to activate a plan it cannot maintain natively: try it in the The Workbench (p. 74), which shows whether it runs.

Indexed equality filters (`owner == $owner`) are fast: about 8 ms at 10,000 records.

## UI files

Partial

A **UI file** sits beside a contract (`order.ui.md` beside `order.dhl`) and says what its records, actions and queries mean to the people and agents who use them: labels, money and dates, status badges, form steps, and a sentence for each refusal. The contract stays pure logic, and a UI file changes without upgrading any data.

Every field, argument, column, status value and validation line a UI file names is checked against the contract. If it does not match, it does not compile: a typo never silently disappears.

### Example

````
---
contract: com.acme.orders.order
---

```ui
present Order
    label "Order" plural "Orders"
    title subject, placed_at
```
````

```
    status: states label "Status"
        0 Draft tone neutral
        1 Placed tone info
        2 Shipped tone success
    total_minor: money currency=currency minor label "Total"

present action Place
    label "Place order"
    tone primary
    confirm "Place order {subject} for the cart's buyer?"
    explain entries.len > 0 "Add at least one line."

form Place
    step "Cart" cart, buyer
    step "Lines" entries, total_minor
    buyer: prefill signer hidden
    entries: table columns line, product_id, quantity
```

Front matter has one key, `contract`: the contract's namespace. Indentation is four spaces; `#` and `//` start comments.

### Blocks

| Block | Holds |
|---|---|
| `present <Aggregate>` | The record: `label "X" [plural "Xs"]`, `help`, `title <field>, …`, and annotations per state field |
| `present struct <Name>` | Labels and annotations for a struct's fields (sub-tables, rows) |
| `present action <Name>` | `label`, `help`, `tone neutral\|primary\|destructive`, `confirm "… {arg} … {subject}"`, annotations per argument, `explain <validate line> "text"` |
| `present query <Name>` | `label`, `help`, `columns a, b, …`, annotations per column, `arg <name>: …` |
| `form <Action>` | `section "Title" a, b` or `step "Title" a, b` (not both), and form annotations per argument |

Each block appears at most once. `record`, `browse`, `compose`, `view` and `relation` are reserved Coming soon.

### Field annotations

| Annotation | For | Effect |
|---|---|---|
| `label "…"`, `help "…"`, `placeholder "…"` | any | Header, form label, help line; also your accomplice's argument descriptions |
| `money currency=<field>` or `money currency="EUR"`, `minor` | Int, Decimal | `€4,250.00` input in major units |
| `quantity [unit "kg"]` | Int, Decimal | A number with its unit |
| `percent` | Int, Decimal, Ratio | `12 %` |
| `date`, `datetime`, `relative` | Timestamp | Formatting and a picker |
| `email`, `phone`, `url`, `image`, `file`, `color`, `markdown` | Text | Renderer and input |
| `textarea [rows=N]` | Text | Multi-line input |
| `identity` | Identity | Shown by display name |
| `states` + lines `<value> <Name> [tone …] [label "…"]` | Int, Text, Enum | Status badges, filter chips, a status select |
| `choice` + lines `<value> "Label"` | Enum, Text, Int | A select with labels |

| Annotation | For | Effect |
|---|---|---|
| `refers <namespace>` | Identity, Ref, SubjectRef | A link to that record |
| `sensitive` | any | Masked until revealed (display only) |
| `hidden`, `priority N`, `align start\|end` | any | Column visibility, order, alignment |
| `default <value>` | arguments | A starting value |

One meaning annotation (money, quantity, states, choice, date…) per field.

**explain**

`explain <validation line> "text"` gives a refusal its words. The line must exist in the action (whitespace does not matter). When that rule refuses, the text is the error, shown on the fields the rule reads: in the form before submitting when it can tell, otherwise from the server.

**Form annotations**

| Annotation | Effect |
|---|---|
| `visible when <condition>` | Shown only while the condition holds (DHL syntax, this action's arguments) |
| `table columns a, b` | A list argument as an editable sub-table |
| `prefill signer` | The acting identity's key |
| `prefill subject` | The record id |
| `prefill record.<field>` | The record's field, when the form opens from a row |
| `hidden` | Not shown; needs a prefill or a default |
| `readonly` | Shown, not editable |

**Without a UI file**

Every contract still gets a description derived from its rules: each validation line becomes a rule with its kind, and simple argument rules become input limits (`x > 0` ⟶ minimum 1; `name.len > 0` ⟶ required). Refusals then name the rule's source line.

**Where to put it**

Beside the contract, with the same stem, in `MANDALA_CONTRACTS` or the installed contracts directory. Mandala compiles descriptions at startup; a UI file that does not match its contract is logged and ignored, never stopping the node. `dhl-engine describe <contract> <out>` writes the description, the diagnostic module and the enriched OpenAPI.

Coming soon — UI files for the built-in standard contracts, record pages, server-side browse tables, compose forms, and the Workbench checking a UI file with its contract.

## Durability classes

A contract declares when its writes are acknowledged, in its front matter:

```
---
namespace: sam.social.like
version: 1.0.0
durability: buffered(100ms)
---
```

| Class | Acknowledged when | Lost by a crash | For |
|---|---|---|---|
| `confirmed` (default) | On disk, by its own sync or a shared one | Nothing | Payments, stock, orders, approvals, identity, outward effects |

| Class | Acknowledged when | Lost by a crash | For |
|---|---|---|---|
| `grouped` | On disk, after waiting up to 2 ms to share a sync | Nothing | Most business writes |
| `buffered(window)` | At once; on disk within the window | At most the last window | Likes, views, presence, conversation traces |

The window is 1 ms to 10 s: `buffered(100ms)`, `buffered(2s)`. A buffered write's receipt says `saved` instead of `confirmed`. A stronger write carries every weaker one written before it.

### What can never be buffered

The compiler refuses `buffered` when:

1. a word of the namespace names **money** (`payment`, `payable`, `invoice`, `ledger`, `finance`, `wallet`, `checkout`…), **stock** (`stock`, `inventory`, `reservation`, `order`, `lot`…), **identity and authority** (`identity`, `root`, `credential`, `certificate`, `device`, `ownership`, `role`, `key`…), **approvals**, **outward effects** (`effect`, `outbox`) or **inbound messages** (`inbox`);
2. any field or argument holds a `Currency`
3. any action changes other records (`requires`).

### Good to know

- The class is not part of the artifact: changing it does not change the artifact id or fork any history. It is still a reviewed change of the contract's source.
- On several voters (a cluster), `buffered` behaves as `confirmed` for now.
- Measured on a single node: a buffered write about 1.6 ms, a confirmed one 5–8 ms, dominated by the disk sync.

## Merge rules for sync

When your devices work apart (a laptop offline, two nodes changing the same list) their changes are replayed on every node in one canonical order. The contract decides what happens when two changes meet, action by action, in its front matter:

```
---
namespace: std.personal.shopping_list
version: 1.0.0
merge: CreateList=commutes, Archive=last_writer
---
```

| Rule | When two changes meet | Use for |
|---|---|---|
| `commutes` | Both apply, in either order | Adding items, recording transactions, appending |
| `last_writer` | The later one (by the hybrid clock) wins | A title, a preference, a status anyone may set |
| `exclusive` | One applies; the other becomes a **conflict** | Approving, paying, reserving the last unit |

An action not listed is `exclusive`, so nothing merges silently that was not designed to.

### Conflicts

A change that no longer passes the contract's validation after the merge is never dropped: it becomes a `std.sync.conflict` record, shown in **Settings → Devices and sync → Didn't merge** for a person to settle. See Sync and conflicts (p. 55).

### Designing for merge

- **Make additions their own records.** A list's items are separate records (`std.personal.shopping_item`), so two people adding at once never touch the same record.
- **Keep money and stock exclusive.** `std.business.payable` declares `Approve=exclusive, Pay=exclusive`: two devices cannot both pay one invoice without a conflict being recorded.

- **Prefer `last_writer` only for values where the latest truly wins.**

Organisations whose operations need one agreed order (an order paid once, the last unit sold once) run their own cluster with consensus instead of merging; their people's devices act as clients.

## Upgrading a contract

How to

Records keep the history they were written under. When you change a contract that already has records, the old version is **archived** (it keeps reading old records), and each record moves to the new version through a **signed upgrade action** declared in the new contract.

### 1. Add an upgrade action

The new version names the exact artifact it upgrades from:

```
action Upgrade(owner: Identity)
    authorize dhl
    certify upgrade from
aacc19c23739aabbc2b146b7e5a8cf34f4ef9b1b3cd3b2ea1a9a158f3a9d871e
    validate
        state.revision > 0
        context.signer == owner
        owner == state.owner
    apply
        state.revision = state.revision + 1
```

The Workbench tells you the running version's id when you try to install a new version without one.

### 2. If the state changes shape, add migrate

```
action Upgrade(owner: Identity)
    authorize dhl
    certify upgrade from 9f3c…e1
    migrate
        rename body -> text
        drop legacy_flag
    validate
        state.revision > 0
    apply
        state.revision = state.revision + 1
```

Rules, checked when compiling and again when loading:

- Every old field is **kept** by name, **renamed**, or explicitly **dropped**. Nothing disappears implicitly.
- New fields take their declared default (or their type's zero value).
- Allowed value changes: the same type; `Text(len=a)` to `Text(len=b)` with `b ≥ a` `T` to `T?` a list or map growing its capacity. Narrowing, numeric conversions and enum changes are refused.
- A `private` field becomes `public` only with an explicit `expose <field>` line.
- Struct definitions used by state cannot change yet.

### 3. Install and upgrade the records

Install the new version from the The Workbench (p. 74). The old one moves to `contracts/archived/`, and records not yet upgraded still read in their old version. Each record moves when someone signs its `Upgrade` action. For Mandala's built-in contracts, Mandala signs these upgrades at startup under your delegation, and retries what fails at the next start.

### In a repository

Keep the previous version's exact source beside the contract, in an `archived/` folder (`contracts/std/business/archived/`), so every node can still read records written under it. Never edit an archived file.

## The generated API

Every contract gets an API derived from its artifact. Actions and queries are internal unless marked `api authenticated` or `api public`.

### Mandala's routes (simplest)

Inside the Workspace and for scripts on your tailnet, Mandala signs for you, as the acting identity, under your delegation:

| Route | Does |
|---|---|
| `GET /api/data` | Every contract with its state, queries and actions, typed |
| `POST /api/data/query` | Runs a query for the acting identity |
| `POST /api/data/preview` | Shows exactly what an action will sign |
| `POST /api/data/act` | Signs and submits an action |
| `GET /api/personal/{contract}/query/{query}` | Queries a personal contract (`?as=<identity>` acts for another identity) |
| `POST /api/personal/{contract}/act/{action}` | Acts on a personal contract |
| `GET /api/business/{contract}/query/{query}`, `POST /api/business/{contract}/act/{action}` | The same for business contracts |

### The engine's API (`/v1`)

The engine's own interface is served on the same address under `/v1`. Here the client signs: commands carry the original signed assertion, and the API never signs on anyone's behalf.

| Request | Purpose |
|---|---|
| `GET /v1/{tenant}/{namespace}/openapi.json`, `…/schema.graphql`, `…/api.proto`, `…/descriptors.pb` | Discover the interface |
| `POST /v1/{tenant}/{namespace}/actions/{Action}` | Submit `{command, args}` |
| `POST /v1/{tenant}/{namespace}/queries/{Query}` | Run a query with its arguments |
| `POST /v1/{tenant}/{namespace}/graphql` | GraphQL |

JSON requests carry `x-dhl-artifact` naming the active artifact. Command metadata holds `subject`, `expected_version`, `actor`, `idempotency_key` and the hex-encoded `signed_assertion_cbor`. A tenant-scoped bearer credential authenticates the actor; the contract still decides whether the action is allowed.

Integers, timestamps and decimals are decimal strings; identities are lowercase hex (DHL types (p. 66)).

### Receipts

An accepted action returns a receipt with the record's new revision. A refused one returns the refusal, with the rule that refused it when the contract can tell (UI files (p. 68)). A timeout leaves the outcome uncertain: recover the original intent and receipt before submitting a replacement.

### Describe offline

```
dhl-engine describe contracts/com/acme/orders/order.dhl /tmp/order-api
```

writes `openapi.json`, `schema.graphql`, `api.proto` and the contract description. See Command line (p. 101).

## The Workbench

How to

The Workbench is where contracts are written and installed without a repository or a build. Open **Data → Workbench: write a contract**.




*Figure 2: The Workbench: the contract on the left, steps and queries on the right, and the trial's results below Try it*

dhl | | ✓ Lend drill {“status”:“out”,...} | | aggregate Loan ... | | ✗ Return drill — refused: ... |

```
## Try it

1. Write or paste the contract on the left. A template is there to start from.
2. On the right, list **steps** (actions to apply, in order) and **queries** (to run
after):

```json
{
  "steps": [
    {"action": "Lend", "subject": "drill", "args": {"ref": "drill", "owner": "$julien",
"item": "Cordless drill", "borrower": "Ana"}},
    {"action": "Return", "subject": "drill", "args": {"owner": "$julien"}}
  ],
  "queries": [{"name": "Out", "params": {"owner": "$julien", "after": ""}}]
}
```
```

- `subject` is the record's id: the same subject in two steps is the same record.
- `"$handle"` stands for that identity's key. `"$julien"` is you: `julien` is the internal handle of the person who set up the home (unless `MANDALA_PRINCIPAL` named another).
- `"actor"` in a step chooses who signs (default: the person `julien` of the trial node).

3. Press **Try it**. Mandala compiles the contract and, if it compiles, runs every step on a **throwaway node** in a separate process: nothing touches your real records. Each step shows ✓ with the record's new state, or ✗ with DHL's reason. Queries show their rows.

Iterate until DHL accepts what it should and refuses what it should.

### Install

**Install…** then **Install and restart**. The contract is saved in `<data>/contracts/<namespace>.dhl`, Mandala restarts to load it (all pending work is durable), and then:

- its records appear in **Data**
- its actions and queries become your accomplice's tools (`dhl.<namespace>.*`), loaded on demand.

A namespace that already runs can only be replaced by a version that **upgrades** from it (Upgrading a contract (p. 72)); the old version is archived so records not yet upgraded still read. Contracts built into Mandala cannot be replaced from here. Namespaces starting `mandala.` are reserved.

### With your accomplice

**Write one with Alma** asks your accomplice to draft a contract with you: it asks what the record should hold, drafts it, tries it with `dhl.workbench.check` until it accepts and refuses what it should, and shows you the source. Trying is free; installing (`dhl.workbench.install`) needs your approval.

Coming soon — installing without a restart, the UI file written with the contract, and your accomplice proposing rule changes from what it measures.

## Testing contracts

How to

A contract is only as good as its refusals. Test both sides of every rule.

### In the Workbench

For each action, write at least:

1. **One accepted step** — the normal case.
2. **One refused step per validation line** — break exactly one rule each time, and check the reason names it.
3. **One step from the wrong signer** — `"actor": "someone_else"`.

4. **One query** — that it returns the rows you expect, and none you don't.

A good trial reads like a story:

```
{
  "steps": [
    {"action": "Lend",   "subject": "drill", "args": {"ref": "drill", "owner":
"$julien", "item": "Cordless drill", "borrower": "Ana"}},
    {"action": "Lend",   "subject": "drill", "args": {"ref": "drill", "owner":
"$julien", "item": "Cordless drill", "borrower": "Ben"}},
    {"action": "Return", "subject": "drill", "args": {"owner": "$julien"}, "actor":
"mallory"},
    {"action": "Return", "subject": "drill", "args": {"owner": "$julien"}},
    {"action": "Return", "subject": "drill", "args": {"owner": "$julien"}}
  ],
  "queries": [{"name": "Out", "params": {"owner": "$julien", "after": ""}}]
}
```

Expected: ✓ lent to Ana; ✗ lent twice; ✗ returned by a stranger; ✓ returned; ✗ returned twice; the query returns nothing.

#### From the command line

```
dhl-engine compile path/to/contract.dhl      # compiles, prints the artifact id
dhl-engine describe path/to/contract.dhl out # interfaces and the description (checks a
UI file too)
```

#### In a repository

Contracts under `contracts/` are compiled by the engine's tests, which also pin every contract's artifact id (a change you did not intend shows up as a changed id) and check that UI files leave artifacts unchanged. Add a test against a real node for contracts with composition (`requires`) or delegation.

#### Checklist

- Every action checks `context.signer`.
- Creating twice is refused (`state.revision == 0`).
- Each status transition checks the current status.
- Money is in minor units with a currency.
- Merge rules are declared for actions used from several devices.
- Queries filter by owner and page by a stable key.

## DHL best practices

#### Model

- **One record per thing that changes on its own.** A list and its items, an order and its lines: separate records, linked by references. They merge and scale better.
- **Give each record a stable `ref`.** Derive the subject from something natural (the supplier and invoice number), so the same invoice received twice is recorded once.
- **Keep a `revision`.** `state.revision == 0` in a creating action makes "create twice" impossible; `+ 1` on each change makes history readable.
- **Record who did what.** `reviewed_by`, `approved_by`, `paid_by` are cheap and answer most questions later.
- **Money in minor units** (`amount_minor: Int`) with a `currency` field beside it.

#### Rules

- **Check the signer in every action.** `context.signer == writer`, and that the writer may act on this record (`writer == state.owner`, or a delegation it observes).

- **Write one condition per validation line.** Each line becomes a rule with its own refusal message; long `and` chains give vague refusals.
- **Make states explicit.** A `status` with a few known values, and actions that check the current one (`state.status == "approved"` before `Pay`), is the clearest workflow.
- **Use delegation for organisations.** Take `owner`, `writer` and a `delegation` argument, and observe it (see `std.business.payable`), rather than trusting whoever signs.
- **Refuse past an end.** If a grant or an offer can expire, compare with `context.clock.milliseconds`.

**Sync and durability**

- **Declare `merge` for every action** you expect from several devices. The default `exclusive` is safe but turns every concurrent change into a conflict.
- **Never buffer money, stock, identity or approvals** — the compiler enforces it, but design for it from the start.

**Queries**

- **Filter by owner first** (`where owner == $owner`), sort by a stable key and page with `ref > $after` and `take 100`.
- **Return a row struct** with `subject as id`, so clients and your accomplice can act on a row.

**People**

- **Write prose around the blocks.** A contract is read by people (and your accomplice) deciding what it allows.
- **Add a UI file** for labels, money, statuses and refusal sentences. It costs nothing to the rules.
- **Try both sides in the Workbench**: an accepted step and a refused one for every rule.

## Example contracts

Every node runs Mandala's standard library. Read these in the source repository under `contracts/`, or in **Data**, where each shows its fields, queries and actions.

**Personal (`std.personal`)**

| Contract | Teaches |
|---|---|
| `std.personal.shopping_list` | A small record with owner and writer, `commutes` and `last_writer` merges, a paged query |
| `std.personal.shopping_item` | Additions as their own records, so concurrent adds never collide |
| `std.personal.finance_transaction`, `finance_account`, `finance_budget` | Money in minor units; owner, writer and delegation |
| `std.personal.health_measurement` | Records written by an importer (Apple Health), one per metric per day |
| `std.personal.appointment`, `health_medication`, `home_asset`, `home_maintenance`, `travel_trip`, `travel_booking` | The same pattern for the rest of life |

**Business (`std.business`)**

`std.business.payable` is the best single example: a workflow from *received* to *reviewed*, *approved* and *paid* (or *rejected*), each step checking the previous status, recording who took it, and checking a **delegation** for the organisation. `Approve` and `Pay` are `exclusive`, so two devices cannot both pay one invoice. Its `Upgrade` action shows a signed upgrade from version 1.0.0.

```
action Pay(owner: Identity, writer: Identity, payment_reference: Text(len=200),
delegation: SubjectRef<Delegation>, delegation_scope: Text(len=200),
delegation_ends_ms: Int)
    api authenticated
```

```
    authorize dhl
    observe delegation where active == true
    observe delegation where party == arg.owner
    observe delegation where delegate == arg.writer
    observe delegation where scope == arg.delegation_scope
    observe delegation where ends_ms == arg.delegation_ends_ms
    validate
        state.revision > 0
        state.status == "approved"
        context.signer == writer
        delegation_scope == "*" or delegation_scope == "std.business.payable" or
delegation_scope == "std.business.payable.Pay"
        delegation_ends_ms == 0 or delegation_ends_ms > context.clock.milliseconds
        owner == state.owner
        payment_reference.len > 0
    apply
        state.delegation = delegation
        state.status = "paid"
        state.paid_by = writer
        state.payment_reference = payment_reference
        state.revision = state.revision + 1
```

### Identity and exchange

| Contract | Teaches |
|---|---|
| `std.identity.identity`, `ownership`, `membership`, `delegation`, `device`, `root` | Records each signed by the key the contract requires |
| `std.exchange.relationship`, `std.exchange.document` | Trust kept as your own records; signed documents routed or held |

### Commerce (`com.cmdv.btd`)

A real online grocery's rules, in the repository: carts and cart lines, orders and order lines, stock lots, ordering calendars and sourcing.

| Contract | Teaches |
|---|---|
| `cart.dhl` | `requires` to open child lines atomically; `external` datasets bounding a cart to 16 lines |
| `order.dhl` | `requires … for entry in entries`: one checkout touching every line, all or nothing |
| `order_line.dhl` | A line reserving stock from a lot through `requires` |
| `cart_line.dhl` | `agg` totals over child records |
| `order.ui.md`, `product.ui.md` | UI files: statuses, money, links, a two-step form, every rule explained |

# Administration

Administering Mandala is mostly done in **Settings**, plus a few files on the machine. If you run Mandala for yourself, you are its administrator.

- Models and workload routing (p. 79) — which model does what, reasoning effort, routing with Jev, costs.
- Keys and credentials (p. 80) — how keys are stored, added, replaced and removed.
- Notifications (p. 80) — web push and Apple push.
- Users and permissions (p. 81) — who can reach your Mandala today, and what is coming.
- Backups and restore (p. 82) — what to back up, and how to restore.
- Updates (p. 83) — installing a new version, and rolling back.
- Logs (p. 84) — where to look when something is wrong.

For the machine itself (systemd, network, monitoring, security), see Sysops (p. 86).

## Models and workload routing

How to

**Settings → Models → Workload routing** decides which connection, model and reasoning effort each kind of work uses. Changes apply to new turns.

| Workload | Key | Used for | If unset |
|---|---|---|---|
| Conversation | `chat` | Talking with you, on every channel | Required |
| Triage | `triage` | Mail, messages and events in the background | Conversation's model |
| Follow-ups | `wakeup` | Scheduled wakeups, routines, the morning plan | Conversation's model |
| Sensitive conversations | `sensitive` | Sensitive mode, or adult content found by Jev | Conversation's model |
| Simple requests | `simple` | Requests Jev grades simple | Conversation's model |
| Complex requests | `complex` | Requests Jev grades complex | Conversation's model |

**Reasoning**: *Provider default*, *Off*, *Minimal*, *Low*, *Medium*, *High*, *Extra high* or *Max*, where the model supports it. Higher effort is slower and dearer; use it for Complex requests rather than for everything.

A connection marked *not connected* in the list must be connected first (Choose a model (p. 10)).

### Routing with Jev

With a TypeSafe key connected, Jev can read the last few messages before each of your turns and pick the model:

- **Adult content to the sensitive model** — uses the Sensitive model and instructions when the conversation is adult content.
- **Route by complexity** — simple requests to the Simple model, complex ones to the Complex model, the rest to Conversation.

Both are off by default. If Jev does not answer, the usual model runs. Your messages go to TypeSafe for this.

### Keeping costs down

- A small fast model for **Triage**: it runs on every batch of new mail and messages.
- Large tool families load only when needed (What your accomplice can do (p. 22)), which keeps each request small.

- Goals and initiatives have a daily token budget (400,000 by default) on the Initiatives page.
- **Initiatives** shows background tokens today and your conversation's tokens over 24 hours.

### Sign-ins

ChatGPT sign-ins (the OpenAI connection on your ChatGPT plan) and xAI sign-ins renew automatically while their refresh token is valid; **Settings → Overview** flags an expired one. xAI can be signed into again from its page (**Sign in again**). A new ChatGPT sign-in cannot yet be started from the Workspace; an OpenAI API key replaces it if it ever expires.

## Keys and credentials

How to

Every credential (API keys, sign-in tokens, bearer tokens for MCP servers) is encrypted with your node's **master key** (`<data>/master.key`) and stored with Mandala's records. Keys are never shown again after you save them.

### Add or replace

- In the app: open the connection's page in **Settings** (Models or Extensions), paste the key, **Connect**. **Replace** swaps it; **Disconnect** removes it.
- From the environment file, at each start: `ALMA_OPENAI_API_KEY`, `ALMA_ANTHROPIC_API_KEY`, `ALMA_GEMINI_API_KEY`, `ALMA_XAI_API_KEY`, `ALMA_DEEPSEEK_API_KEY`, `ALMA_KIMI_API_KEY`, `ALMA_ZAI_API_KEY`, `ALMA_META_API_KEY`, `ALMA_BRAVE_API_KEY`, `ALMA_WAVESPEED_API_KEY`, `ALMA_TYPESAFE_API_KEY`. A key in the environment overwrites the stored one at every start; remove it from the file once stored if you prefer to manage it in the app.

### See what is stored

**Settings → Extensions → Stored credentials** lists every credential by name, never by value.

### Protect the master key

`master.key` decrypts every credential. Keep it readable only by Mandala's user (`chmod 600`), back it up with the rest of the data directory, and never copy it anywhere else. Lose it, and you must enter every key again; leak it with a copy of the data, and every key is exposed: rotate them at their providers.

### Other secrets on the node

| File | Holds |
|---|---|
| `push/` | The Web Push (VAPID) key and, if set, the Apple push key (`apns.p8`) |
| `telegram.db`, `telegram-accounts/` | Telegram sessions |
| `whatsapp/` | WhatsApp sessions |

Treat the whole data directory as secret. See The data directory (p. 89).

## Notifications

How to

Your accomplice notifies you when something needs you now, when it replies while no Mandala window is open, and when an approval waits.

### Turn them on, per device

**Settings → Accomplice → Notifications → Turn on** on each device that should receive them, and allow notifications when the browser asks. **Devices that receive them** lists every device; **Remove** one, or **Send a test**.

- **iPhone**: add Mandala to the Home Screen first, open it from there, then turn them on (Install the web app on your phone (p. 53)).

- **Android, Mac, Windows, Linux**: any browser with Web Push.
- The address must be HTTPS (Tailscale and HTTPS (p. 88)).

Notifications go through your browser vendor's push service, encrypted end to end; the signing key (VAPID) is created on your server and never leaves it. Expired subscriptions are dropped by themselves.

### Apple push for the native apps

The iPhone and Mac apps receive notifications through Apple's push service, with Approve and Deny on approval requests (after Face ID).

1. In your Apple Developer account: **Certificates, Identifiers & Profiles → Keys → +**, with *Apple Push Notifications service*. Download the `.p8` file (Apple offers it once).
2. **Settings → Accomplice → iPhone and Mac apps → Apple push key**: upload it with its **Key ID** and your **Team ID**. Mandala checks it signs, then keeps it on the server, never returning it.
3. Open the app and allow notifications. It appears under **Apps that receive notifications**.

### What triggers a notification

- Your accomplice decides something needs you now (money owed, an unhappy customer, a delivery problem, a deadline).
- It replies while no window is open.
- An approval is requested while no window is open.

Coming soon — quiet hours, at most two interruptions a day, and "not this kind again" on every notification.

## Users and permissions

Partial How to

### Your account

A Mandala home belongs to **one person** today: you. Your account is your first name and a **passkey** (Face ID, Touch ID, Windows Hello), created at the first run (The first run (p. 10)). The passkey is also one of your root authorities: what decides which devices are yours.

### Signing in

One button and Face ID; nothing typed. **Settings → Security**:

- **Your seal** — every device of yours shows the same one.
- **Ask for Face ID to open Mandala** / **Require sign-in** — on for homes created with an account.
- **Passkeys** — what can sign you in and approve your devices. **Add** one (saved in iCloud Keychain or Google Password Manager) and approve it with what you already have: another passkey, your paper key words, or a security key. A root holds at most 3 keys.
- **Devices and browsers** — everywhere you are signed in, with **Sign out** for each. Sign out anything you don't recognise.
- **Add a device** — see Join a device with a code (p. 57).

Sessions last **30 days**, renewed as you use them, on your own devices; **12 hours**, closed with the browser, on a shared computer.

### Face ID again for sensitive acts

Some acts ask for a fresh Face ID, valid five minutes: changing which devices are yours (keys), who can act for you (people), what your accomplice may do on its own (autonomy), moving money, letting your accomplice use a computer, and showing a secret. A **Confirm it's you** sheet appears; nothing happens without it.

### Homes set up before accounts

A node configured by environment (`ALMA_OWNER_NAME`, `MANDALA_PRINCIPAL`) keeps working without sign-in, behind your tailnet, until you add a passkey in Settings → Security and turn on **Require sign-in**. `MANDALA_LOGIN=required` or `off` overrides the account's choice.

### The network is still a wall

Keep Mandala on your tailnet (Tailscale and HTTPS (p. 88)) even with sign-in on: it stays an extra wall until Mandala is hardened for the public internet. Tailscale ACLs can narrow who reaches the machine.

### Permissions inside

- **Your accomplice** acts under its delegation and your autonomy settings, with approvals for everything that matters (Approvals (p. 18)).
- **Organisations and spaces** are acted for by delegation, checked by the rules (Delegation (p. 49)).
- **Devices** are trusted by certificate from your root (Keys and recovery (p. 56)).
- **Apple Health** pushes need a bearer token you can revoke.

### Coming soon

Coming soon — **several people per home**: invited colleagues and household members, each with their own identity, passkeys and accomplice, and roles in words (Inviting people (p. 50)).

## Backups and restore

How to

Everything Mandala knows is in its **data directory** (`ALMA_DATA_DIR`, usually `~/.mandala`), plus your Goethe vaults if you placed them elsewhere. Back up both.

### Make a consistent backup

The record store is written continuously. Copy it while Mandala is stopped, or from a filesystem snapshot.

**Stopped (simplest):**

```
sudo systemctl stop mandala
tar -C ~ -czf /backups/mandala-$(date +%F).tar.gz .mandala
sudo systemctl start mandala
```

Stopping takes a few seconds: Mandala finishes what it is writing and releases its lock. Restarts are safe; interrupted work resumes.

**Snapshot (no downtime):** take a snapshot of the volume (ZFS, btrfs, LVM, or your VPS provider's snapshots), then copy from the snapshot.

Keep at least one copy **off the machine**, and encrypt it: the backup contains your credentials' master key and your private data.

### What is in it

| Path | Holds | If lost |
|---|---|---|
| `dharma/` | Every record and its history; your accomplice's conversations and memory | Everything |
| `master.key` | The key that decrypts stored credentials | Every API key and sign-in must be entered again |
| `vaults/` | Vaults created from Knowledge, and the Docs vault | Those documents (Docs is rebuilt from the release) |
| Your Goethe vaults (`ALMA_GOETHE_JULIEN_ROOT`, agent roots) | Your documents and your accomplice's | Those documents |
| `vault-sync/` | Vault sync state | Rebuilt |
| `contracts/` | Contracts installed from the Workbench | Those contracts |
| `telegram.db`, `telegram-accounts/`, `whatsapp/` | Chat sessions and stored messages | You sign in again; older messages may not come back |

| Path | Holds | If lost |
|---|---|---|
| `push/` | Notification keys | Devices must turn notifications on again |
| `comms/index.db` | The Communications index (a cache) | Rebuilt from Gmail and chats |

### Restore

```
sudo systemctl stop mandala
mv ~/.mandala ~/.mandala.broken
tar -C ~ -xzf /backups/mandala-2026-10-01.tar.gz
sudo systemctl start mandala
curl -s http://127.0.0.1:8787/health
```

Restore with the **same or a newer** Mandala version than the one that made the backup.

### A second node is not a backup

A paired node holds a live copy and survives the loss of a machine, but a deletion syncs too. Keep dated backups as well.

Coming soon — backups set up automatically when Mandala installs itself on a VPS.

## Updates

How to

Mandala can update itself from **signed releases**, or you can install a build yourself. Either way, the previous version is kept and put back if the new one does not come up.

### Self-update

**Settings → Updates** (off by default):

| Mode | Does |
|---|---|
| **Off** | Never checks. |
| **Ask me** | Checks every 6 hours; when a newer release is out, your accomplice and a notification tell you, and Settings shows **Install and restart**. |
| **Automatic** | Checks hourly and installs inside a night window (03:00–05:00 local by default). |

**Check now** looks at once.

### How it stays safe

1. A release is an archive (`bin/mandala` and `web/`) and a manifest signed with your **release key**, which never sits on a server. A node accepts only a manifest signed by that key (`MANDALA_UPDATE_PUBLIC_KEY` in `mandala.env`, or pasted in Settings), for its own platform, and newer than what it runs.
2. The archive's SHA-256 must match the signed one; only `bin/` and `web/` are unpacked; the new binary must run and print the release's version before anything is swapped.
3. The running binary is kept as `mandala.prev` and the web app as `web.prev` Mandala stops gracefully and the service manager starts the new one.
4. The new binary counts its starts: after **3** failed starts, or if `/health` has not answered within **180 seconds**, the previous binary and web app are put back.

Your home node serves the release channel to your other nodes (`linux-x86_64`, `linux-aarch64`, and the Mac app's).

### Publishing a release

```
mandala/deploy/release.sh build 2026.1011.1                    # on the build server:
build, package, print the SHA-256
```

```
MANDALA_PLATFORM=linux-x86_64 mandala/deploy/release.sh sign 2026.1011.1
mandala-2026.1011.1-linux-x86_64.zip   # where the release key is
mandala/deploy/release.sh publish mandala-2026.1011.1-linux-x86_64.zip latest.json
https://<home>.ts.net   # archive first, manifest last
```

Create the release key once with `mandala sign-update --key <file> --generate` (the same key as the Mac app's), keep the private half offline, and put the printed public key in each node's `MANDALA_UPDATE_PUBLIC_KEY`.

### By hand, from source

```
cd mandala && git pull
cargo build --release -p mandala
(cd applications/workspace && npm ci && npm run build)
MANDALA_BIN=$PWD/target/release/mandala mandala/deploy/ship.sh
```

`ship.sh` runs the deploy in the background (log: `~/mandala-deploy.log`): it keeps `mandala.prev` and `web.prev`, installs the new build into `~/.mandala`, restarts `mandala.service` (with `sudo systemctl`), waits up to 60 seconds for `/health`, and puts the previous version back if it does not answer.

### Before you update

- **Back up** (Backups and restore (p. 82)). Some releases change how records are written on disk; older versions then cannot read them, so going back after such an update means restoring the backup taken before it.
- Read the Changelog (p. 104).

### After you update

- The **Docs** vault refreshes with the new release's pages (The Docs vault (p. 45)).
- Built-in contracts with a new version upgrade your records at startup, under your delegation.
- The Mac app keeps its own updater (Install on a Mac (p. 8)).

## Logs

How to

### Where

| Install | Logs |
|---|---|
| Linux with systemd | `journalctl -u mandala -f` |
| Mac app | `~/Library/Logs/Mandala/mandala.log` |
| Mac launchd agent | `/tmp/mandala.log` |
| A deploy with `ship.sh` | `~/mandala-deploy.log` |

### More or less detail

`ALMA_LOG` sets the level, with the usual filter syntax: `info` (the default), `debug`, `warn`, or per module, such as `info,alma_dharma=debug`. Set it in the environment file and restart.

### What to look for

- `Mandala Workspace is running` — the node is up, with its address.
- `Docs vault is up to date` — how many pages were written, removed or kept.
- `vault sync is off`, `cannot open a Goethe vault` — a vault path is wrong or unreadable.
- `WhatsApp did not reconnect`, Telegram errors — sign in again in Settings.
- `answering messages interrupted by the last restart` — normal after a restart: work resumes.

### Your accomplice's side

For what your accomplice did and why, **Activity** is more useful than logs: every turn, its trigger, its tools and how it ended (Activity (p. 41)).

See also Troubleshooting (p. 91).

# Sysops

Mandala is one program, run as one service. There is no database server, no queue, no container platform to operate. This part is for whoever runs the machine.

## Understand

- Architecture (p. 86) — what runs inside the one binary.
- System requirements (p. 87) — hardware, operating systems, what to install.

## Install and run

- Install on Linux (p. 6) — build, configure, systemd.
- Install on a VPS (p. 9) — a rented server, locked down.
- Tailscale and HTTPS (p. 88) — reaching it privately over HTTPS.
- Ports and network (p. 88) — ports, the device mesh, outbound connections.
- The data directory (p. 89) — every file and folder Mandala keeps.

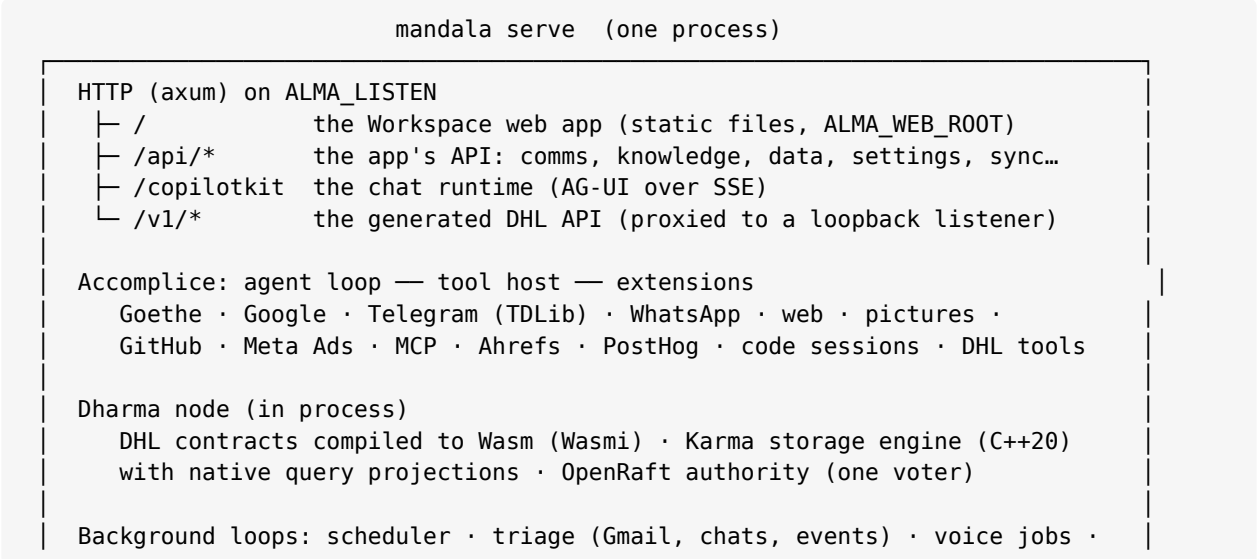
## Operate

- Monitoring (p. 90) — health, service state, disk.
- Performance (p. 90) — what is fast, what is slow, what to tune.
- Backups and restore (p. 82) and Updates (p. 83).
- Troubleshooting (p. 91) — common problems and fixes.
- Security (p. 93) — the security model and hardening checklist.

## Beyond one node

- The Server edition (p. 93) — the DHL engine without the Workspace Coming soon.
- Clusters (p. 94) — several voters for an organisation Coming soon.

## Architecture



```
│     sync (HTTP + iroh mesh) · vault sync (Loro CRDT) · live co-editing        │
└──────────────────────────────────────────────────────────────────────────────┘
       data directory: records, keys, vaults, chat sessions, caches
```

### The pieces

- **Workspace** — a React app, served as static files by the binary.
- **The accomplice** — a plain agent loop: build the prompt, call the model with tools, run the tools through the host (which enforces approvals, autonomy and the risk check), repeat. Conversations, turns, tool calls, approvals and memory are signed records.
- **Goethe** — typed Markdown vaults on disk: documents, types, views, tasks, links. Each vault is a directory you can also open with any editor or the Goethe TUI.
- **Dharma and Karma** — the DHL engine. Dharma compiles and runs contracts and checks authority; Karma keeps accepted state durably and maintains the query projections. No SQL server.
- **Sync** — each node keeps a log of signed intents; paired nodes exchange what the other lacks every 20 seconds, over HTTP or the iroh device mesh. Vaults sync as text CRDT updates on the same log.

### Principles you can rely on

- **One executable, one service.** Coding turns and Workbench trials run as subcommands of the same binary (`mandala code-turn`, `mandala workbench-check`), never as separate daemons.
- **Durable background work.** Every queue, poll and pending job is stored. A restart (or a crash) loses nothing: interrupted answers are resumed, effects in flight are followed until they settle, and backfills continue where they stopped.
- **Restart is a normal operation.** Mandala restarts itself to load a new vault or contract; the service manager must restart it on exit (`Restart=always`).

## System requirements

### To run

| | Minimum | Comfortable |
|---|---|---|
| CPU | 2 cores, x86-64 or arm64 | 4 cores |
| Memory | 2 GB free for Mandala | 4 GB or more |
| Disk | 10 GB free | 40 GB+, SSD (every confirmed write waits for a disk sync) |
| OS | Linux with systemd (Debian 12, Ubuntu 22.04+), or macOS 15+ on Apple silicon | |
| Network | Tailscale | |

These are guidance rather than hard limits. For scale: a busy personal node with several mail and chat accounts uses about 1.3 GB of memory, and its data directory stays under 1 GB after months of use. Mail-heavy use grows the disk: the Communications index, attachments in the Library and record history.

### To build

- Memory: about 8 GB (Rust and the C++ storage engine).
- rustup the repository pins **Rust 1.98.0**.
- A C/C++ compiler with **C++20** and its standard library (`build-essential`, or clang), `pkg-config`, OpenSSL headers.
- **Node.js 20+** and npm for the web app.

### Optional software on the server

| For | Needs |
|---|---|
| Telegram | TDLib `libtdjson` (`/opt/alma/telegram/lib/libtdjson.so`, or `ALMA_TDLIB_LIBRARY`) |

| For | Needs |
|---|---|
| Local models | Ollama |
| Coding sessions | Claude Code and/or Codex CLIs, logged in as Mandala's user; `git` `gh` |

### Clients

Any current browser (Safari, Chrome, Firefox, Edge). Installing on a phone and notifications need HTTPS.

## Tailscale and HTTPS

How to

Mandala has no login screen yet, so it must only be reachable by your own devices. Tailscale gives every device a private network address and a name, and can serve Mandala over HTTPS with a real certificate.

### Serve Mandala on your tailnet

On the Mandala machine:

```
sudo tailscale serve --bg --https=443 http://127.0.0.1:8787
tailscale serve status
```

Mandala is now at `https://<machine>.<tailnet>.ts.net` for devices on your tailnet. Enable HTTPS certificates for your tailnet in the Tailscale admin console if asked.

Use another port if 443 is taken: `--https=8445` gives `https://<machine>.<tailnet>.ts.net:8445`.

### Why HTTPS matters

- The installable app and notifications need a secure origin.
- **Passkeys and security keys** are bound to the site name (by default your tailnet's name, `tail1234.ts.net` `MANDALA_RP_ID` sets another). Keep the same name: changing it makes registered passkeys unusable here.
- Microphone access for voice needs a secure origin.

### On each device

Install Tailscale and log in with the same account. On the iPhone, keep Tailscale connected (On Demand helps) for the app and Apple Health pushes to reach your home.

### Never

- Do not use `tailscale funnel`: it publishes the service to the internet.
- Do not open Mandala's port on your router or firewall.

Coming soon — reaching your home without Tailscale through the device mesh, and an optional public address on your own domain once Mandala is hardened for the public internet.

## Ports and network

### Listening

| What | Where | Notes |
|---|---|---|
| The Workspace and its API | `ALMA_LISTEN`, default `127.0.0.1:8787` (Mac app: `127.0.0.1:8790`) | Publish with `tailscale serve`, never directly |
| The generated DHL API | A random loopback port | Reached only through `/v1/*` on the main address |
| Webhooks | `MANDALA_WEBHOOK_LISTEN`, default `127.0.0.1:8788` | Serves only `/hooks/github`, `/hooks/meta`, `/hooks/meta-media` exposed through a Cloudflare tunnel (Cloudflare and webhooks (p. 30)) |

| What | Where | Notes |
|---|---|---|
| The device mesh (iroh) | UDP, a port chosen at start | QUIC, mutual authentication by device key; also used to join a node by code |

### Between your nodes

Paired nodes sync every 20 seconds, both ways, in one of two ways:

- **HTTP** to the peer's address (on your tailnet), with each pull signed by the device key.
- **The device mesh** (endpoint `iroh`): nodes dial each other by device key, with direct QUIC connections and hole punching. A node's addresses are found through signed records it publishes on the Mainline DHT, and on the local network by mDNS (only once the node trusts another device, so a fresh install sends nothing on your LAN). Connections from devices that are not trusted are refused at once.

Nothing goes through a third party's servers. Where hole punching fails, a relay of your own (`iroh-relay` on your VPS, named in `MANDALA_IROH_RELAY`) forwards encrypted traffic. Without it, relays are off.

`MANDALA_MDNS=1` or `0` forces local discovery on or off.

### Outbound connections

Depending on what you connect: your model providers; Cloudflare (`cloudflared`); Google APIs; Telegram; WhatsApp; Inworld (voice); Brave (search); WaveSpeed (pictures); TypeSafe (Jev); PostHog, Ahrefs, Meta, GitHub and your MCP servers; browser push services and Apple Push. A firewall that blocks outbound traffic needs these allowed.

### Between businesses

Signed exchange documents arrive at `POST /api/exchange/receive` on the recipient's address. Today counterparts exchange addresses by hand, typically over a shared tailnet. Coming soon — reaching businesses through the public network and discovery records.

## The data directory

`ALMA_DATA_DIR` (default `~/.alma` set it, typically to `~/.mandala`). The Mac app uses `~/Library/Application Support/Mandala/data`. Keep it on a local SSD, owned by Mandala's user, mode `700`.

| Path | What |
|---|---|
| `dharma/` | The DHL engine: every record, its history, projections, the engine's own configuration (`api.json`) and installed artifacts |
| `master.key` | Encrypts stored credentials. Secret. |
| `account/` | Your account and sessions (`sessions.json`) |
| `updates/` | Staged releases and the result of the last self-update |
| `contracts/` | Contracts installed from the Workbench; `contracts/archived/` holds replaced versions |
| `vaults/<name>/` | Vaults created from Knowledge (each with `vault.json`), and `vaults/docs/`, the Docs vault |
| `vault-sync/` | Vault sync state (one CRDT document per file, outside the vault) |
| `comms/index.db` | The Communications thread index (a cache, rebuilt if lost) |
| `telegram.db`, `telegram/`, `telegram-accounts/<id>/` | Telegram sessions and stored messages |
| `whatsapp/` | WhatsApp sessions and stored messages |
| `push/` | The Web Push key; `apns.p8` if Apple push is set |
| `code/` | Coding sessions: worktrees' turn directories, agent output |

| Path | What |
|---|---|
| `releases/` | The Mac app's update channel (releases published to this node) |
| `mandala.env` | The environment file, by convention (the service reads it via `EnvironmentFile=`) |
| `bin/`, `web/`, `web.prev/` | The installed binary and web app, by convention of `ship.sh` |

Your own Goethe vaults may live elsewhere (`ALMA_GOETHE_JULIEN_ROOT`, `ALMA_GOETHE_AGENT_ROOTS`). Back them up too.

Do not edit files under `dharma/` by hand.

## Monitoring

Partial How to

### Health

```
curl -fsS http://127.0.0.1:8787/health
```

answers when the node is up. Use it from a timer, your monitoring system, or after a deploy.

### The service

```
systemctl status mandala
journalctl -u mandala --since "1 hour ago" | grep -iE "error|warn"
```

`Restart=always` brings Mandala back after a crash or a deliberate restart; frequent restarts in the journal mean something is wrong.

### Disk

Every confirmed write waits for a disk sync, and a full disk stops writes. Alert at 80% on the data directory's volume.

### In the app

- **Settings → Overview** flags expired sign-ins and extensions that are not connected.
- **Settings → Devices and sync** shows each peer's state and how many of its changes are held here.
- **Initiatives** shows background token use against its daily limit.
- **Activity** shows failed turns with their errors.

Coming soon — operations dashboards: node health, sync lag, queue depths and model spend in one page.

## Performance

### What to expect on one node

| Operation | Typical |
|---|---|
| A `buffered` write (conversation traces) | ~1.6 ms |
| A `confirmed` write | 5–8 ms, mostly the disk sync |
| A batch of 20 writes | ~70 ms |
| An indexed query (`owner == $owner`) at 10,000 records | ~8 ms |
| A warm query of 20 records | ~2 ms |
| Concurrent writers on one tenant | several hundred writes per second |

The accomplice's speed is dominated by the model, not by Mandala.

### What helps

- **A fast disk.** Confirmed writes wait for `fsync`. An NVMe SSD makes a visible difference; network disks are slowest.
- **The right model per workload.** A fast model for triage and simple requests (Models and workload routing (p. 79)).
- **Durability classes** for high-volume, low-stakes contracts (Durability classes (p. 70)).
- **Indexed filters first** in queries, and paging by key (DHLQ query reference (p. 67)).

### What not to do

- Don't run Mandala on a disk that lies about `fsync` (some USB enclosures): a power cut could lose acknowledged writes.
- Don't put the data directory on a network share.

### Clusters

Certified writes across several voters cost several sequential durable rounds today (about 250 ms for a stock reservation on three local voters). A single-node Mandala is not affected. See Clusters (p. 94).

## Troubleshooting

How to

Start with the logs (`journalctl -u mandala -f`) and **Settings → Overview**.

### Mandala does not start

| Symptom | Fix |
|---|---|
| `cannot listen on 127.0.0.1:8787` | Another process holds the port. Change `ALMA_LISTEN`, or stop the other one. |
| `ALMA_LISTEN must be host:port` | Write it as `127.0.0.1:8787`. |
| `ALMA_GOETHE_AGENT_ROOTS must be a JSON object` | Write it on one line, the JSON in single quotes: `'{"alma":"/path"}'`. |
| `Goethe directories must be private and service-owned` | `chmod 700` the vault directory and its parents under your control, owned by Mandala's user. |
| `cannot start Dharma` | Is another Mandala using the same data directory? Only one process may. Check free disk space and permissions. |
| `master key: …` | `master.key` is unreadable or corrupt. Restore it from a backup; never delete it on a node with stored credentials. |
| It starts, then the service restarts in a loop | Read the first error in `journalctl -u mandala -b`. A failed update? `mandala.prev` is the previous binary. |

### Signing in

| Symptom | Fix |
|---|---|
| "This page needs its secure address (https)" | Open Mandala at its `https://` tailnet address (Tailscale and HTTPS (p. 88)). |
| Face ID is not offered, or my passkey isn't found | The address changed since the passkey was made (passkeys are bound to the site name). Use the original name, or `MANDALA_RP_ID`. |
| Signed out on a shared computer | Sessions there end after 12 hours or when the browser closes. |
| "Paused for 10 minutes to keep your identity safe" | Too many wrong codes when adding a device. Wait, then make a **New code**. |
| "Joined. Restart Mandala on this device to finish." | Restart the new node (`systemctl restart mandala`, or quit and reopen the Mac app). |

### The app

| Symptom | Fix |
|---|---|
| The page does not load from my phone | Is Tailscale connected on the phone? Does `tailscale serve status` show the service? |
| "Knowledge could not be loaded" / "Goethe is not configured" | Set `ALMA_GOETHE_JULIEN_ROOT` to an existing directory and restart. |
| My accomplice does not answer | Settings → Models: is the Conversation workload's provider connected? Look in Activity for the failed turn's error. |
| "… isn't connected yet — add the API key in Settings → …" | Your accomplice tried an integration that has no key; add it where the message says. |
| "Your accomplice has no model yet" | Press **Choose one** in the banner (Choose a model (p. 10)). |
| "Connect this provider first" in Workload routing | Connect it under Settings → Models. |
| A sign-in expired | Settings → Overview lists it; sign in again or replace with an API key. |
| Notifications never arrive on the iPhone | Turn them on from the Home Screen app, not from Safari. Check **Send a test**. |

### Channels

| Symptom | Fix |
|---|---|
| A Telegram or WhatsApp chat is empty | Press **Allow Alma to see this chat** in Communications. |
| WhatsApp shows *Unlinked from the phone* | The phone removed the device. Link again with a new code. |
| Telegram does not sign in | Check the api_id and 32-character api_hash, and that TDLib is installed (`ALMA_TDLIB_LIBRARY`). |
| Mail stopped updating | Check the Google account's sign-in in Settings → Extensions → Google. The index resumes its sync by itself. |
| Smart boxes stay empty | They need a TypeSafe key, and fill a few minutes after messages arrive. |

### Approvals and autonomy

| Symptom | Fix |
|---|---|
| "yes, send it" did not approve | Approval by words needs the Jev key, and payments and deletions always keep their card. Tap the card. |
| An allowed action still asked | The risk check sent it back; the card says why. |

### Sync

| Symptom | Fix |
|---|---|
| A paired device stays *Trusted*, never *Syncing* | This node cannot reach it: give it an address, or `iroh`, when pairing. |
| Changes don't arrive | Pair on **both** sides. With a root key, certify the device. |
| Items under *Didn't merge* | Settle them: Sync and conflicts (p. 55). |

### Contracts

| Symptom | Fix |
|---|---|
| Install refused: "already runs" | Add an upgrade action from the running version (Upgrading a contract (p. 72)). |
| "is built into Mandala" | Built-in contracts change only with Mandala itself. |
| A query is refused at activation | It uses a stage the engine cannot maintain natively (DHLQ query reference (p. 67)). |

## Security

### The model

- **Self-hosted.** Your data and keys live on machines you control. There is no Mandala operator who could read, lock or take over your account.
- **Signed everything.** Every record change is signed by the key of whoever made it (a person, an agent, a device) and checked by the contract that governs it. A mistake or a forgery is refused, not recorded.
- **Rules in the engine, not in prompts.** Your accomplice cannot do what its delegation, its autonomy settings and the contracts do not allow, however it is asked.
- **Approval for effects.** Messages, payments and other outward actions wait for you, and are recorded with the exact action approved.
- **Your passkey is your root.** Devices are certified by it; a stolen server cannot add a device to your identity. Sensitive acts ask for a fresh Face ID.

### What it does not protect against today

- **A home set up before accounts has no sign-in** until you add a passkey and turn on Require sign-in (Users and permissions (p. 81)). Homes created with the first-run assistant require Face ID.
- **Whoever controls the machine can read its data**, including a VPS provider. Keep the most private things on hardware you hold.
- **Model providers see what your accomplice reads to answer.** Choose providers accordingly, or a local model.
- **WhatsApp's unofficial protocol** can get a number banned.

### Hardening checklist

- □ Mandala listens on `127.0.0.1` published only with `tailscale serve` no `funnel`, no port forwarding.
- □ Tailscale ACLs allow only your own devices to reach the machine.
- □ The host firewall denies incoming traffic except on `tailscale0` SSH by key only.
- □ Mandala runs as a dedicated, unprivileged user, with `NoNewPrivileges=yes` and `PrivateTmp=yes`.
- □ The data directory is mode `700`, `mandala.env` and `master.key` mode `600`.
- □ Automatic security updates for the OS.
- □ Encrypted backups, off the machine, tested by a restore.
- □ Sign-in required, and at least two ways back (Keys and recovery (p. 56)).
- □ Sign out sessions you don't recognise (Settings → Security).
- □ Forget lost devices at once (Settings → Devices and sync).
- □ Review **Without asking** and connected extensions now and then.
- □ Your Mac release key (`~/.mandala-release/update.key`) backed up offline and never on a server.

### Reporting a problem

If you find a security issue, report it privately to the maintainers rather than in a public issue.

## The Server edition

Coming soon

> Coming soon — a packaged Server edition with its own terminal interface. Today the engine is available as the `dhl-engine` command for developers.

The **Server edition** is Mandala's DHL engine without the Workspace: no web app, no accomplice, no vaults. It runs a business's contracts on the business's own servers, serves their generated API, and lets people's Mandalas act on it as clients, under the business's delegations.

Use it when an organisation's operations need their own always-on authority, separate from any one person's home: an online shop's orders and stock, a company's payables shared by several people.

### Planned

- One binary, configured by a file, run as one service.
- A terminal interface to install contracts, watch nodes and voters, inspect records and receipts.
- Clusters of several voters for availability (Clusters (p. 94)).

### Today: dhl-engine

```
cargo build --release -p dhl-engine --bin dhl-engine
dhl-engine compile contracts/std/business/payable.dhl
dhl-engine describe contracts/std/business/payable.dhl /tmp/payable-api
dhl-engine api <config.json>
```

`compile` checks a contract and prints its artifact id; `describe` writes its OpenAPI, GraphQL and protobuf interfaces and its description; `api` serves the generated API from a configuration naming the authority nodes and installed artifacts. See Command line (p. 101).

## Clusters

Coming soon

Coming soon — organisations on their own clusters. Every Mandala today runs a single node per home, with your devices syncing between homes.

### Personal versus organisation

- **Your devices merge.** Your laptop accepts writes offline and merges later; contracts declare how (Merge rules for sync (p. 71)).
- **An organisation agrees.** Where money and stock move (an order paid once, the last unit sold once), the organisation's servers form a cluster that agrees on one order through consensus (Raft). People's devices are clients: actions on the organisation queue as intents while offline and are submitted when back online; the cluster may refuse them, with a reason.

### What exists

The engine runs replicated authority with several voters (OpenRaft), tested with three processes surviving the loss of a leader and node restarts under contention. `dhl-engine cluster run <config>` starts a voter of the prototype cluster.

### What is being worked on

A certified cross-record transaction (a stock reservation) currently costs five to six sequential durable rounds, about 250 ms on three local voters. The protocol is being redesigned to need fewer rounds before clusters are offered for production.

# Reference


- Glossary (p. 95) — every Mandala word, defined once.
- Keyboard shortcuts (p. 96)
- HTTP API (p. 97) — the routes the app and your scripts use.
- Environment variables (p. 100) — every variable of `mandala.env`.
- Command line (p. 101) — `mandala` and `dhl-engine` commands.
- File formats (p. 102) — contracts, UI files, vault documents, types, views.
- FAQ (p. 103)
- Changelog (p. 104)
- What's coming (p. 105) — everything marked Coming soon, in one place.


## Glossary

**Accomplice** — The AI that works with you inside Mandala, defined by you. Your accomplice (p. 15)

**Action** — The only way a record changes: typed arguments, validation, then changes to the state. DHL concepts (p. 59)

**Agent** — An identity run by a person or organisation, such as your accomplice. Its grants never exceed its runner's.

**Approval** — A card showing exactly what an action will do, waiting for your decision by tap, notification or words. Approvals (p. 18)

**Artifact** — A compiled contract (schema + Wasm), identified by its hash. What actually runs.

**Back office** — The Data page and boards generated from contracts.

**Conflict** — A change from another device that no longer applies after merging; kept for you to settle. Sync and conflicts (p. 55)

**Constitution** — The text that tells your accomplice who it is and how it works. Meet your accomplice (p. 12)

**Contract** — A DHL file describing one kind of record, its actions and queries. DHL concepts (p. 59)

**Delegation** — A signed grant to act for another identity, with a scope and an optional end. Delegation (p. 49)

**Device key** — Each node's own key; signs its changes and identifies it on the mesh.

**DHL** — The language of contracts. **DHLQ** — its query language. DHL language reference (p. 63), DHLQ query reference (p. 67)

**Dharma** — The part of the engine that compiles and runs contracts and checks authority.

**Docs** — The vault holding this manual, shipped with each release. The Docs vault (p. 45)

**Durability class** — When a contract's writes are acknowledged: confirmed, grouped or buffered. Durability classes (p. 70)

**Exchange** — Signed documents sent directly between identities' nodes. Exchange between businesses (p. 51)

**Exchange key** — The key an identity signs exchanged documents with.

**Goethe** — Mandala's typed Markdown document system: vaults, types, views, tasks, links. Knowledge (p. 36)

**Home** — The node where your identity lives, always reachable; other devices join it.

**Identity** — A person, organisation, space or agent: a key and a record of who it is. Identities (p. 48)

**Initiative** — A piece of work your accomplice proposes towards one of your goals. Goals, initiatives and reflection (p. 21)

**Intent** — A signed change in a node's log, exchanged with your other nodes when syncing.

**Jev** — A risk and classification model hosted by TypeSafe, used optionally for approval by words, the risk check, smart boxes and routing.

**Karma** — The engine's storage, keeping accepted state durably and maintaining query projections.

**Merge rule** — How concurrent changes to a contract's records merge: commutes, last_writer, exclusive. Merge rules for sync (p. 71)

**Node** — One running Mandala with its data directory.

**Organisation** — A legal identity you own, such as a company.

**Record** — One instance of a contract (one invoice, one list), with its signed history. Also called a *subject*.

**Root key** — The authority that certifies which devices are yours: a paper key, security keys, trustees. Keys and recovery (p. 56)

**Passkey** — The key your device keeps for your account, unlocked by Face ID, Touch ID or Windows Hello. It signs you in and is one of your root authorities.

**Seal** — A drawing made from an identity's key, the same on every device, used to recognise it (Settings ⟶ Security).

**Session** — A signed-in browser or device, listed in Settings ⟶ Security.

**Step-up** — A fresh Face ID asked before a sensitive act (keys, people, money, autonomy, computer use, secrets).

**Shadow mode** — Your accomplice reflects and proposes but works on nothing. Goals, initiatives and reflection (p. 21)

**Space** — A shared, non-legal identity, such as a household.

**Standing rule** — An instruction your accomplice always follows. Memory and rules (p. 19)

**Tenant** — The identity whose records a contract's records belong to.

**UI file** — A file beside a contract that gives its records and actions labels, formats, forms and refusal sentences. UI files (p. 68)

**Vault** — A folder of Goethe documents with its types and views.

**Workbench** — Where you write, try and install contracts. The Workbench (p. 74)

**Workload** — A kind of work routed to a model: conversation, triage, follow-ups, sensitive, simple, complex. Models and workload routing (p. 79)

## Keyboard shortcuts

| Where | Keys | Does |
|---|---|---|
| Everywhere | ⌘B / Ctrl+B | Collapse or expand the sidebar |
| Everywhere | ⌘K / Ctrl+K | Search anything, and commands such as *Pin this* |
| Everywhere | Alt+P (⌥P) | Pin what is open (or unpin it with its Pin button) |
| Everywhere | Alt+Shift+1 … Alt+Shift+7 (⌥⇧1 … ⌥⇧7) | Open superpin 1 to 7 |
| Composer | Enter | Send |

| Where | Keys | Does |
|---|---|---|
| Composer | Shift+Enter | New line |
| Composer | Alt+1, Alt+2, Alt+3 | Send a suggestion |
| Anything visible | Alt+A (⌥A) | Ask your accomplice about what you point at |
| Code session | ⌘Enter / Ctrl+Enter | Send your message to the agent |
| Choice wizard | ↑ ↓, Enter | Move between options, choose |
| Choice wizard | Esc | Leave *Something else…* |
| Knowledge editor | ⌘B / Ctrl+B | Bold |
| Knowledge editor | ⌘I / Ctrl+I | Italic |
| Knowledge editor | ⌘E / Ctrl+E | Inline code |
| Knowledge editor | ⌘K / Ctrl+K | Link to a document |
| Knowledge editor | ⌘Enter / Ctrl+Enter | Tick or untick the task on this line |
| Knowledge editor | ⌘S / Ctrl+S | Save now |
| Knowledge editor | `[[` | Link to a document, with completion |
| Knowledge editor | ⌘-click / Ctrl-click on a link | Open the linked document |
| Knowledge, a field | Enter / Esc | Save / cancel the field |
| Knowledge sidebar | F2 | Rename |
| Knowledge sidebar | ⌘X, ⌘V / Ctrl+X, Ctrl+V | Cut and paste documents |
| Communications | j k, Enter, Esc, x, e, s, u, !, r, /, ? | See Communications (p. 35) (`?` lists them) |
| Mac app | ⌘Q | Quit (the node finishes what it is writing) |

Superpins use ⌥⇧ with a digit because ⌘1 … ⌘9 switch browser tabs and ⌥1 … ⌥3 send suggestions. See Pins (p. 42).

## HTTP API

The Workspace talks to its node over plain JSON routes under `/api`. When sign-in is required, every route needs a session cookie (sign in with a passkey; Users and permissions (p. 81)), and sensitive routes ask for a fresh Face ID. Writes are signed by the node as you, and actions that need approval still need it.

### Health and live updates

| Route | |
|---|---|
| `GET /health` | Up or not |
| `GET /api/bootstrap` | What the app needs on load |
| `GET /api/events` (WebSocket) | Live updates: committed changes and streaming text |

### Account and devices

| Route | |
|---|---|
| `GET /api/account` | Whether this node is set up, signed in, your first name and seal |
| `POST /api/account/create/begin\|register\|finish` | Create your identity with a passkey |
| `POST /api/account/signin/begin\|finish`, `POST /api/account/signout` | Sign in and out |
| `POST /api/account/stepup/begin\|finish` | A fresh Face ID for a sensitive act |
| `GET /api/account/sessions`, `POST /api/account/sessions/{id}/end` | Devices and browsers signed in |

| Route | |
|---|---|
| `/api/account/passkey`, `/api/account/settings`, `GET /api/account/recovery` | Passkeys, sign-in settings, ways back |
| `/api/pair/*` | Adding a device with a code |
| `/api/setup/model/*` | The first-run model step (ChatGPT sign-in, trying a model) |

### Accomplice

| Route | |
|---|---|
| `GET/POST /api/messages` | Your conversation; send a message (`{text, attachments}`) |
| `POST /api/messages/{id}/reactions` | React to a message |
| `POST /api/cancel` | Stop the running turn |
| `GET /api/approvals`, `POST /api/approvals/{id}` | Waiting approvals; decide (`{approve}`) |
| `GET /api/asks`, `POST /api/asks/{id}/answer` | Choice wizards |
| `GET/POST /api/branches`, `POST /api/branches/{id}/merge` | Side conversations |
| `GET /api/turns`, `GET /api/turns/{id}`, `GET /api/activity` | Turns and what they did |
| `GET/POST /api/memory`, `DELETE /api/memory/{id}` | Memory notes |
| `GET/POST /api/rules`, `DELETE /api/rules/{id}` | Standing rules |
| `GET/PUT /api/persona`, `PUT /api/content-mode` | Constitution, sensitive instructions; content mode |
| `/api/autonomy/*` | Goals, initiatives, reflections, settings |
| `GET /api/suggestions` | What to say next |
| `GET /api/artifacts/{id}`, `POST …/refresh`, `…/schedule` | Dashboards: read, refresh, schedule |
| `/copilotkit/*` | The chat runtime (AG-UI) |

### Voice

| Route | |
|---|---|
| `GET/PUT /api/voice`, `GET /api/voice/voices`, `POST /api/voice/preview` | Voice settings |
| `POST /api/voice/notes` | A voice message, transcribed |
| `GET /api/voice/call` (WebSocket) | A live call |

### Communications

| Route | |
|---|---|
| `GET /api/comms/list`, `/api/comms/counts`, `/api/comms/people` | The thread index |
| `GET /api/comms/threads/{id}`, `POST …/reply`, `…/star`, `…/allow` | Read, reply, star, allow a chat |
| `POST /api/comms/compose`, `/api/comms/translate` | New email; translation |
| `GET/PUT /api/comms/settings`, `/api/comms/signatures` | Settings and signatures |
| `/api/whatsapp/*`, `/api/telegram/accounts/*`, `/api/settings/telegram/*` | Chat accounts |

### Knowledge, tasks, library

| Route | |
|---|---|
| `POST /api/knowledge` | Goethe operations on a vault (`{operation, args, vault}`) |

| Route | |
|---|---|
| `POST /api/calc` | Evaluate calculation lines |
| `/api/collab/*` | Live co-editing |
| `GET/POST /api/tasks`, `POST /api/tasks/{id}/complete\|reopen\|reschedule` | Tasks |
| `GET /api/agenda` | The agenda |
| `GET/POST /api/library`, `GET /api/library/{id}/content`, `GET /api/blobs/{id}` | Files |
| `GET /api/manual/assets/{file}` | Pictures of this manual |

### Records

| Route | |
|---|---|
| `GET /api/data`, `POST /api/data/query`, `/preview`, `/act` | The back office |
| `/api/personal/{contract}/…`, `/api/business/{contract}/…` | Personal and business records |
| `GET /api/analytics` | Analytics |
| `POST /api/workbench/check`, `/install` | The Workbench |
| `/v1/*` | The generated DHL API (The generated API (p. 73)) |

### Identities, devices, exchange

| Route | |
|---|---|
| `GET/POST /api/identities`, `POST /api/identities/{handle}/owners\|members\|delegations\|vault`, `PUT /api/scopes` | Identities, ownership, delegation |
| `/api/sync/*` | Pairing, sync, conflicts |
| `/api/root/*` | Root keys |
| `/api/exchange/*` | Relationships and signed documents |

### Settings and integrations

| Route | |
|---|---|
| `GET /api/connections`, `PUT/DELETE /api/connections/{id}/key` | Model and service keys |
| `GET/PUT /api/settings/models`, `/routing`, `/autonomy`, `/google` | Routing, Jev, autonomy, Google |
| `/api/settings/integrations/*` | PostHog, Ahrefs, Meta Ads, GitHub, MCP |
| `/api/settings/apple-health/*`, `POST /api/ingest/apple-health`, `GET /api/personal/health` | Apple Health: settings, imports, goals, alerts |
| `/api/settings/meta/*`, `/api/settings/cloudflare/*` | Facebook & Instagram, Cloudflare |
| `/api/settings/updates`, `…/check`, `…/install` | Self-update |
| `/api/push/*` | Notifications |
| `/api/apple/*` | The Apple apps: overview, joining the mesh, the Mac update channel |
| `/api/code/*` | Coding sessions: projects, agents, sessions, answers, checks, reviews, tests, stages |

### Webhooks (separate port)

`127.0.0.1:8788`: `/hooks/github`, `/hooks/meta`, `/hooks/meta-media` only (Cloudflare and webhooks (p. 30)).

## Environment variables

Mandala reads its configuration from the environment, usually the file named by `EnvironmentFile=` in its systemd unit (`~/.mandala/mandala.env`). Empty values count as unset. Restart after a change.

### Core

| Variable | Default | Meaning |
|---|---|---|
| `ALMA_DATA_DIR` | `~/.alma` | The data directory. Set it (usually `~/.mandala`). |
| `ALMA_LISTEN` | `127.0.0.1:8787` | Address and port of the Workspace and API |
| `ALMA_WEB_ROOT` | the bundled build, if present | Directory of the built web app |
| `ALMA_LOG` | `info` | Log filter (`debug`, `info,alma_dharma=debug`) |
| `ALMA_TIMEZONE` | `Asia/Manila` | Your time zone (IANA name); routines run in it |

### Who you are

| Variable | Default | Meaning |
|---|---|---|
| `ALMA_OWNER_NAME` | unset | Your first name, for a node configured without the first-run assistant |
| `MANDALA_PRINCIPAL` | `julien` | Your internal handle |
| `MANDALA_TENANT` | `julien` | The tenant your records live in |
| `ALMA_AGENT_ID` | `alma` | Your accomplice's agent id |

Leave `ALMA_OWNER_NAME` and `MANDALA_PRINCIPAL` unset on a new home: the first-run assistant then asks your first name. Setting either marks the node as set up by environment (no assistant, no sign-in until you add a passkey). Never change the handles once records exist.

### Accounts

| Variable | Default | Meaning |
|---|---|---|
| `MANDALA_LOGIN` | the account's choice | `required` or `off` forces sign-in on or off |
| `MANDALA_RP_ID` | your tailnet name, or the host | The site name passkeys are bound to. Keep it stable. |
| `MANDALA_DEVICE_NAME` | the host name | This device's name when joining |
| `MANDALA_PAIR_LOOKUPS` | on | `off` disables finding the other device through the DHT when joining by code |

### Vaults and tasks

| Variable | Default | Meaning |
|---|---|---|
| `ALMA_GOETHE_JULIEN_ROOT` | unset | Your own vault's directory (the name is historical; it is yours) |
| `ALMA_GOETHE_AGENT_ROOTS` | unset | JSON: agent id → its private vault, `{"alma":"/path"}` |
| `ALMA_TASK_DOCUMENT` | `docs/agenda.md` | Where new tasks go in your vault |

### Routines

| Variable | Default | Meaning |
|---|---|---|
| `ALMA_MORNING_TIME` | `07:30` | Morning briefing, local `HH:MM`, or `off` |
| `ALMA_DIARY_TIME` | `23:30` | Nightly diary, or `off` |

### Contracts

| Variable | Default | Meaning |
|---|---|---|
| `MANDALA_CONTRACTS` | unset | Extra contract files or directories, separated by `:` |

Contracts installed from the Workbench (`<data>/contracts`) load automatically.

### Keys (copied into the encrypted store at each start)

`ALMA_OPENAI_API_KEY`, `ALMA_ANTHROPIC_API_KEY`, `ALMA_GEMINI_API_KEY`, `ALMA_XAI_API_KEY`, `ALMA_DEEPSEEK_API_KEY`, `ALMA_KIMI_API_KEY`, `ALMA_ZAI_API_KEY`, `ALMA_META_API_KEY`, `ALMA_BRAVE_API_KEY`, `ALMA_WAVESPEED_API_KEY`, `ALMA_TYPESAFE_API_KEY`.

### Services

| Variable | Default | Meaning |
|---|---|---|
| `ALMA_OLLAMA_BASE_URL` | `http://127.0.0.1:11434/v1` | Ollama's address |
| `ALMA_INWORLD_BASE_URL` | `https://api.inworld.ai` | Inworld's address |
| `MANDALA_WEBHOOK_LISTEN` | `127.0.0.1:8788` | The webhook listener (only `/hooks/*`) |
| `ALMA_CLOUDFLARED_PATH` | `/usr/bin/cloudflared` | The Cloudflare tunnel connector |
| `ALMA_TDLIB_LIBRARY` | `/opt/alma/telegram/lib/libtdjson.so` (`.dylib` on macOS) | TDLib for Telegram |

### Updates

| Variable | Meaning |
|---|---|
| `MANDALA_UPDATE_PUBLIC_KEY` | The release key's public half; only releases it signed are installed |
| `MANDALA_UPDATE_URL` | The release channel, if not the home's own |

### Device mesh

| Variable | Default | Meaning |
|---|---|---|
| `MANDALA_IROH_RELAY` | unset (no relay) | URL of your own iroh relay |
| `MANDALA_MDNS` | on once another device is trusted | `1` or `0` forces local discovery |

### Mac app

| Variable | Meaning |
|---|---|
| `MANDALA_HOME_URL` | The home the app's window shows by default |

## Command line

### mandala

| Command | Does |
|---|---|
| `mandala serve [--data-dir DIR] [--listen HOST:PORT]` | Runs Mandala. The flags override `ALMA_DATA_DIR` and `ALMA_LISTEN`. |
| `mandala version` | Prints this build's version (the updater checks a staged binary with it) |
| `mandala app` | The Mac app (macOS builds with the desktop feature); opening `Mandala.app` does the same. |
| `mandala sign-update --key FILE --generate` | Creates the Mac app's release key and prints its public half. |
| `mandala sign-update --key FILE --app mac --version V --file ZIP [--notes TEXT]` | Signs a release for the update channel. |

Internal subcommands, started by Mandala itself (one executable, no helpers):

| Command | Does |
|---|---|
| `mandala workbench-check` | Runs one Workbench trial (JSON on stdin) on a throwaway node |
| `mandala code-turn DIR` | Runs one turn of a coding session |

| Command | Does |
|---|---|
| `mandala code-permission DIR` | Answers a coding agent's permission questions (MCP on stdio) |

### dhl-engine

| Command | Does |
|---|---|
| `dhl-engine compile SOURCE` | Compiles a contract, prints dataset guard notes and its artifact id, stores the artifact (`DHARMA_DATA_ROOT`, default `.poc/dharma-agnostic`) |
| `dhl-engine describe SOURCE OUTPUT` | Writes `openapi.json`, `schema.graphql`, `api.proto`, and the contract description (`description.json`, `.cbor`, `diagnostic.wasm`), using `<stem>.ui.md` if present |
| `dhl-engine api CONFIG` | Serves the generated API from a configuration |
| `dhl-engine cluster run CONFIG` | Runs a cluster voter (prototype) |

### Deploy scripts

| Script | Does |
|---|---|
| `mandala/deploy/ship.sh` | Installs the current build into `~/.mandala` in the background, restarts, rolls back if it does not come up (Updates (p. 83)) |
| `mandala/deploy/release.sh build\|sign\|publish …` | Builds, signs and publishes a release for self-update (Updates (p. 83)) |
| `mandala/deploy/macos/install.sh DIR` | Installs a Mac build as a launchd agent |
| `applications/apple/scripts/*.sh` | Build, sign, notarize, release the Mac app; build the iPhone app |

## File formats

| File | Format | Reference |
|---|---|---|
| `*.dhl` | A DHL contract: Markdown with front matter (`namespace`, `version`, `merge`, `durability`) and ```` ```dhl ```` blocks | DHL language reference (p. 63) |
| `*.ui.md` | A UI file: front matter `contract`, ```` ```ui ```` blocks | UI files (p. 68) |
| `docs/**/*.md` in a vault | A Goethe document: optional YAML front matter (`type`, `title`, fields), then Markdown with `[[links]]`, tasks and calculations | Knowledge (p. 36) |
| `types/*.gdl` in a vault | A Goethe type: `type <name>`, `fields`, optional `actions` | Types, views and type actions (p. 46) |
| `views/*.gql` in a vault | A Goethe view: `view <Name>`, a source, stages | Types, views and type actions (p. 46) |
| `vault.json` in `<data>/vaults/<name>/` | Vault metadata: `label`, `owner` the Docs vault adds `title`, `about`, `release` and the digests of shipped pages | The Docs vault (p. 45) |
| `mandala.env` | `KEY=VALUE` lines | Environment variables (p. 100) |

### Goethe tasks

```
- [ ] Text of the task @due(2026-11-03) @scheduled(2026-11-01) @priority(high)
- [x] Done task @done(2026-10-09)
```

Nested tasks are indented under their parent. `docs/diary/` in a vault is reserved for your accomplice's diary.

**Goethe documents**

```
---
type: contact
title: Ana Reyes
emails: [ana@example.com]
---

# Ana Reyes

Purchasing at [[suppliers/seafrigo]].
```

## FAQ

**Is there a hosted Mandala I can sign up for?** No. Mandala is self-hosted only: you run it on a Mac, a home server or a VPS you control. That is what makes it yours. Choose where Mandala runs (p. 5)

**Do I need to be a developer?** Today, installing a server needs comfort with a terminal (Install on Linux (p. 6)); after that, a setup assistant asks your first name and Face ID. A downloadable Mac app and one-click VPS setup are Coming soon. Using Mandala needs no technical knowledge.

**Which AI model should I use?** Your best general model for conversation and a small fast one for triage. Mandala works with OpenAI (ChatGPT sign-in or API key), Anthropic (API key), Google Gemini (API key), xAI Grok (sign-in or key), DeepSeek, Kimi, Z.ai, Meta, and local models through Ollama. Choose a model (p. 10)

**Can I use my Claude Pro or Max subscription?** No. Anthropic does not allow Claude subscriptions to power other apps. Use an Anthropic API key.

**Does my data go to the AI provider?** What your accomplice reads to answer a request goes to the model provider you chose, under its terms. Nothing else does. With Ollama, nothing leaves your machine. Privacy, connection by connection (p. 32)

**Can my accomplice send emails or pay on its own?** Not unless you allow it. By default it asks before every action that changes something outside the conversation; you approve each one. Payments and deletions always keep their approval card. Approvals (p. 18), What your accomplice may do alone (p. 19)

**Is the accomplice called Alma?** Alma is the name of the first accomplice, built for Mandala's creator. Yours is whoever you define: a plain assistant or an accomplice with its own name. Some screens still show "Alma" until personas land. Meet your accomplice (p. 12)

**Can several people use one Mandala?** Not yet: one person per home today. Invitations for colleagues and household members are Coming soon. Inviting people (p. 50)

**Is WhatsApp safe to connect?** It works through WhatsApp's unofficial multi-device protocol. WhatsApp can ban numbers using unofficial clients. Only link a number if you accept that risk. WhatsApp (p. 27)

**Is there a password?** No. Your account is your first name and Face ID (a passkey). Users and permissions (p. 81)

**What happens if my server dies?** Restore your latest backup on a new machine (Backups and restore (p. 82)). A second paired node also keeps a live copy (Pair two nodes (p. 55)).

**What if I lose my devices and keys?** Use another way back: your synced passkey, another device, your recovery kit, or people you trust. Settings → If you lose your devices shows how strong yours are. If every way back is lost, nobody can give the account back. Keys and recovery (p. 56)

**Can I edit these docs?** Yes: Docs is a normal vault. Pages you edit stop receiving updates, so prefer adding your own pages that link to the shipped ones. The Docs vault (p. 45)

**Where do I write my business's rules?** In DHL contracts, from the Workbench in Data, or by asking your accomplice to draft one with you. Tutorial: your first contract (p. 60)

## Changelog

### October 2026

#### 11 October

- **Accounts**: create your identity with a first name and Face ID; sign in with a passkey; sessions listed in Settings → Security; a fresh Face ID for keys, people, money, autonomy, computer use and secrets; your seal; **Add a device** with a temporary QR or code (a node joins over the device mesh); **If you lose your devices**, your ways back in plain words.
- **Choose what powers your accomplice** at the first run: sign in with ChatGPT or Grok, an API key (Google Gemini added), or a local model, tried live.
- **Self-update** from signed releases: off, ask, or automatic at night, with rollback if the new version does not come up.
- **Integrations**: Cloudflare (tunnels, DNS) and a webhook listener; GitHub webhooks; Facebook Pages and Instagram; Apple's own Health export, a daily health digest, goals and Personal → Health; "not connected yet" messages that say where to add the key. Claude by API key only.
- **Code**: permission questions as approval cards (with Approve/Deny on the phone), Codex through its app server, ACP agents (Gemini CLI, Qwen Code, opencode), CI checks with bounded fix rounds, reviewer sessions, optional Docker sandboxes. The Hub extension is removed.
- **Show and tell**: dashboards made of blocks with interactive charts and tables, refreshed on demand or on a schedule; point at anything and ask; suggestions of what to say next.
- **Knowledge**: a document manager (folders, drag and drop, rename, archive and restore, undo) and tidy-up plans reviewed before they apply.
- **Communications**: the whole index, instant filters, keyboard shortcuts, the source on every row, attachments in a side panel and to the Library.

#### 10 October

- **Docs**: this manual, shipped as the Docs vault on every node, and used by your accomplice to answer how-to questions with links.
- **Talking with your accomplice**: links to everything it mentions, choice wizards, live drafting in the canvas, `show` (reports, tables, charts, cards) kept in the Library, reactions, branches, approval by your words, picture and file attachments, Google Docs and Sheets live in the canvas.
- **Voice**: calls with a compact widget, voice messages transcribed, spoken replies.
- **Communications**: mailboxes, Gmail tabs, stars, smart boxes, HTML mail, a rich and Markdown composer, signatures, translation, several WhatsApp numbers and Telegram accounts.
- **WhatsApp**: link your number as a companion device by pair code.
- **Integrations**: settings for PostHog, Ahrefs, Meta Ads, GitHub, MCP servers, Telegram; Apple Health import; large tool families load on demand.
- **Code**: coding sessions with Claude Code and Codex; newsletters rendered and sent with approval.
- **Back office**: UI files for contracts; delegations checked by contracts, with end dates; organisations grant their creator.
- **Root keys**: paper key, security keys and trustees; only certified devices sync once a root exists.
- **Device mesh**: nodes find and reach each other by device key (iroh), with DHT and local discovery.
- **Workload routing with Jev** and the **risk check Analytics** the **DHL Workbench** with contract upgrades (`migrate`); **durability classes** push notifications and the installable web app; Goethe live co-editing.
- **Engine**: faster writes (one disk sync per write), indexed queries 1 s → 8 ms at 10,000 records, smaller batch frames. Older versions cannot read records written by this one: back up before updating.

#### 9 October

- **One binary**: the accomplice moved into Mandala; its state is signed records in the DHL engine.
- **Identities and scopes**: persons, organisations, spaces and agents; owners for every account and vault; the Act as switcher; a vault per identity.
- **Personal library**: groceries and money pages; records for health, home and travel.

- **Sync between your devices**: the signed intent log, merge rules, conflicts, pairing; Goethe vaults sync as text CRDTs.
- **Exchange between businesses**: relationships, signed documents, introductions.
- **Knowledge**: the Tasks table, type actions, views.
- **Payables** board.

## What's coming

These are designed and being built. Each links to the page that describes it. The view **Coming soon** in this vault lists the pages concerned.

### Getting started

- **The rest of the setup assistant**: meeting your accomplice on screen (spectrum, form, name, look, voice), bringing in what you have, bringing your life in, how much it may do alone. The first run (p. 10)
- **Downloads**: a signed Mac app, packaged Linux builds. Install on a Mac (p. 8), Install on Linux (p. 6)
- **Mandala sets itself up on a VPS**: Hetzner first, then Linode/Akamai, DigitalOcean, Vultr, OVH, Scaleway. Install on a VPS (p. 9)

### Your accomplice

- **Personas** with versions, and names threaded through the product. Meet your accomplice (p. 12)
- **Evening memory cards**, a memory policy. Memory and rules (p. 19)
- **The shadow-week reveal** and earned autonomy. What your accomplice may do alone (p. 19)
- **A briefing that ends**, quiet hours, at most two interruptions a day. Morning briefing, follow-ups and diary (p. 20), Notifications (p. 80)
- **Native calls** on the iPhone. Voice messages and calls (p. 17)
- **Computer use**: your accomplice using your Mac, with its own cursor and a Stop. The Mac app as a second node (p. 54)

### Connections

- **Connect Google** from Settings. Set up your email (p. 25)
- **WhatsApp media** and QR linking. WhatsApp (p. 27)
- **Apple Health from the iPhone app**. Apple Health (p. 28)
- **More extensions**: Microsoft 365, IMAP and CalDAV, Slack, Shopify, Stripe, Xero/QuickBooks and more.

### Workspace

- **Simpler navigation** with spaces; **search everything**. Workspace guide (p. 34)
- **Home and travel pages** bank statement import. Personal (p. 37)
- **Back office**: record pages, server tables and pickers, compose, saved views. Data (p. 39), UI files (p. 68)
- **The Ledger**. Activity (p. 41)
- **Projects** for agent work in Tasks. Tasks (p. 39)

### Businesses, people and devices

- **Inviting people**, roles in words, an accomplice per person. Inviting people (p. 50)
- **Granting delegations from Settings**. Delegation (p. 49)
- **Speaking a business into being**, organisation agents. Businesses and teams (p. 48)
- **Console recovery** on your own machines; trustees through Exchange; a printable recovery kit. Keys and recovery (p. 56)
- **The iPhone app** on TestFlight and the App Store. The iPhone app (p. 54)

### Operations

- **The Server edition** with its terminal interface; **clusters**. The Server edition (p. 93), Clusters (p. 94)
- **Operations dashboards**. Monitoring (p. 90)

# How-to guides

| Title | Section | Summary |
|---|---|---|
| Backups and restore (p. 82) | Administration | Back up the data directory safely, restore it, and what a second node does and does not replace. |
| Keys and credentials (p. 80) | Administration | How Mandala stores API keys and sign-ins, and how to add, replace and remove them. |
| Logs (p. 84) | Administration | Where Mandala writes what it does, and how to get more detail. |
| Models and workload routing (p. 79) | Administration | Route each kind of work to the right model, set reasoning effort, and keep costs in check. |
| Notifications (p. 80) | Administration | Get your accomplice's alerts, replies and approval requests on your phone and computers. |
| Updates (p. 83) | Administration | Let Mandala update itself from signed releases, or install a build by hand — with automatic rollback either way. |
| Users and permissions (p. 81) | Administration | Your account, sign-in and sessions, Face ID for sensitive acts, and who else can reach your Mandala. |
| Testing contracts (p. 75) | Building with DHL | Prove that a contract accepts what it should and refuses what it should — before it holds real records. |
| Upgrading a contract (p. 72) | Building with DHL | Change a contract that already has records — signed upgrades, migrate blocks and archived versions. |
| The Workbench (p. 74) | Building with DHL | Write a contract, try it on a throwaway node, and install it — from the app or with your accomplice. |
| Acting for a company (p. 50) | Businesses and teams | Switch who you act as, and what changes across the app and for your accomplice. |
| Exchange between businesses (p. 51) | Businesses and teams | Recognize counterparts, send and receive signed documents, and introduce people. |
| Inviting people (p. 50) | Businesses and teams | Bring colleagues and household members into your home, each with their own identity and accomplice. |
| Who owns what (p. 49) | Businesses and teams | Give each mailbox, chat account, conversation and vault an owner — and move a single chat to your company. |
| Ahrefs (p. 29) | Connecting your life | SEO data for your accomplice, within a daily units budget. |
| Apple Health (p. 28) | Connecting your life | Push daily health measurements and workouts from your iPhone into your personal records. |
| Cloudflare and webhooks (p. 30) | Connecting your life | Let services reach Mandala through a dedicated tunnel, and let your accomplice manage DNS and zones with approval. |
| Set up your email (p. 25) | Connecting your life | Gmail in Communications — mailboxes, signatures, translation, and what your accomplice may do. |
| Facebook and Instagram (p. 30) | Connecting your life | Publish to your Pages and Instagram accounts with approval, and let comments reach your accomplice. |

| Title | Section | Summary |
|---|---|---|
| GitHub (p. 31) | Connecting your life | Repositories your accomplice may read and change, with approval for every change. |
| Google Calendar, Drive, Docs and Sheets (p. 26) | Connecting your life | Per-account, per-service access to your calendars, files and documents. |
| MCP servers (p. 32) | Connecting your life | Add tools from any remote MCP server, reviewed one by one. |
| Meta Ads (p. 29) | Connecting your life | Ad accounts your accomplice may read, and where it may propose changes. |
| PostHog (p. 29) | Connecting your life | Product analytics your accomplice can read, with events and thresholds that wake it. |
| Telegram (p. 26) | Connecting your life | Sign in with your own Telegram accounts and choose, chat by chat, what your accomplice reads. |
| Web search and pictures (p. 32) | Connecting your life | Brave Search for the web, WaveSpeed for generated pictures. |
| WhatsApp (p. 27) | Connecting your life | Link your WhatsApp number as a companion device, like WhatsApp Web. |
| Install the web app on your phone (p. 53) | Devices and sync | Add Mandala to your Home Screen, with its own icon, window and notifications. |
| Join a device with a code (p. 57) | Devices and sync | Add a new phone, computer or browser by scanning a temporary QR or typing a short code, approved with Face ID. |
| Keys and recovery (p. 56) | Devices and sync | What decides which devices are yours — and how you get back in if you lose them. |
| The Mac app as a second node (p. 54) | Devices and sync | Join your Mac to your home and show your Mandala from your home. |
| Pair two nodes (p. 55) | Devices and sync | Make two Mandala nodes sync — by device key, over Tailscale or the device mesh. |
| Choose a model (p. 10) | Getting started | Connect the AI model your accomplice thinks with — a sign-in, an API key or a local model. |
| The first run (p. 10) | Getting started | Create your identity with a first name and Face ID, choose what powers your accomplice, and you're in. |
| Install on a Mac (p. 8) | Getting started | Mandala for Mac — one app that is a full node, with its window and menu bar item. |
| Install on a VPS (p. 9) | Getting started | Run your Mandala home on a rented server you control. |
| Install on Linux (p. 6) | Getting started | Build Mandala, install it as a systemd service, and open it over Tailscale. |
| Meet your accomplice (p. 12) | Getting started | Decide who your accomplice is — from a plain assistant to an accomplice with a personality. |
| Monitoring (p. 90) | Sysops | Know that Mandala is up and healthy — health check, service state, disk and the app's own signals. |
| Tailscale and HTTPS (p. 88) | Sysops | Reach Mandala privately from your devices over HTTPS, with a stable name. |
| Troubleshooting (p. 91) | Sysops | Common problems, what they mean, and how to fix them. |
| Agenda (p. 41) | Workspace guide | Your Google calendars and dated tasks in one calendar, by day or week. |

| Title | Section | Summary |
|---|---|---|
| Code (p. 40) | Workspace guide | Run coding agents in your repositories from any device — approvals, CI fixes, reviews and pull requests. |
| Communications (p. 35) | Workspace guide | Mail, Telegram and WhatsApp in one place, by person, with your accomplice at hand. |
| Dashboards (p. 38) | Workspace guide | The payables board — supplier invoices from received to paid, every step signed. |
| Data (p. 39) | Workspace guide | The back office — every record your rules govern, queries, actions with previews, and the Workbench. |
| Folders, moves and tidy-ups (p. 44) | Workspace guide | Keep a vault in order — folders, moves that keep links, and tidy-up plans you review before they apply. |
| Types, views and type actions (p. 46) | Workspace guide | Give documents a shape with types, query them with views, and add actions to types. |
| Knowledge (p. 36) | Workspace guide | Vaults of Markdown documents with types, links, tasks, calculations and live co-editing. |
| Library (p. 37) | Workspace guide | Every file in one place — attachments, uploads and your accomplice's work, filed with reasons. |
| Personal (p. 37) | Workspace guide | Groceries, money and health — lists by aisle, accounts and budgets, measurements and goals. |
| Pins (p. 42) | Workspace guide | Bookmarks to anything in the Workspace or on the web, in folders, with up to seven superpins always at the top of the sidebar. |
| Tasks (p. 39) | Workspace guide | Every task across your documents, filtered, grouped and scheduled by dragging. |
| Approvals (p. 18) | Your accomplice | Approve what your accomplice prepared — with a tap, from your phone, or in your own words. |
| What your accomplice may do alone (p. 19) | Your accomplice | Choose, family by family, what your accomplice does without an approval card. |
| Choices, reactions and branches (p. 17) | Your accomplice | Answer with a tap, react with an emoji, and take a message aside to think it through. |
| Memory and rules (p. 19) | Your accomplice | What your accomplice remembers, the rules it always follows, and how to change both. |
| Sensitive mode (p. 22) | Your accomplice | An explicit, adult-only opt-in for private conversations with your own accomplice. |
| Talking with your accomplice (p. 15) | Your accomplice | The conversation, attachments, replies, and the accomplice panel on every page. |
| Voice messages and calls (p. 17) | Your accomplice | Talk instead of typing — voice messages, spoken replies and live calls. |

# Coming soon

| Title | Section | Availability | Summary |
|---|---|---|---|
| Users and permissions (p. 81) | Administration | Partial | Your account, sign-in and sessions, Face ID for sensitive acts, and who else can reach your Mandala. |
| UI files (p. 68) | Building with DHL | Partial | Say what records, actions and queries mean to people — labels, formats, forms and refusals in words. |
| Delegation (p. 49) | Businesses and teams | Partial | Signed grants to act for someone — with a scope, an optional end date, checked by the rules. |
| Exchange between businesses (p. 51) | Businesses and teams | Partial | Recognize counterparts, send and receive signed documents, and introduce people. |
| Businesses and teams (p. 48) | Businesses and teams | Partial | Run companies and households from your own Mandala — identities, ownership, delegation and exchange. |
| Inviting people (p. 50) | Businesses and teams | Coming soon | Bring colleagues and household members into your home, each with their own identity and accomplice. |
| Apple Health (p. 28) | Connecting your life | Partial | Push daily health measurements and workouts from your iPhone into your personal records. |
| Set up your email (p. 25) | Connecting your life | Partial | Gmail in Communications — mailboxes, signatures, translation, and what your accomplice may do. |
| Google Calendar, Drive, Docs and Sheets (p. 26) | Connecting your life | Partial | Per-account, per-service access to your calendars, files and documents. |
| Devices and sync (p. 53) | Devices and sync | Partial | One Mandala across your server, Mac and phone — how devices join, sync, and stay yours. |
| The iPhone app (p. 54) | Devices and sync | Coming soon | The native Mandala app — push approvals with Face ID, sharing, Health, and Siri. |
| Keys and recovery (p. 56) | Devices and sync | Partial | What decides which devices are yours — and how you get back in if you lose them. |
| The Mac app as a second node (p. 54) | Devices and sync | Partial | Join your Mac to your home and show your Mandala from your home. |
| The first run (p. 10) | Getting started | Partial | Create your identity with a first name and Face ID, choose what powers your accomplice, and you're in. |
| Install on a Mac (p. 8) | Getting started | Partial | Mandala for Mac — one app that is a full node, with its window and menu bar item. |
| Install on a VPS (p. 9) | Getting started | Partial | Run your Mandala home on a rented server you control. |

| Title | Section | Availability | Summary |
|---|---|---|---|
| Meet your accomplice (p. 12) | Getting started | Partial | Decide who your accomplice is — from a plain assistant to an accomplice with a personality. |
| Clusters (p. 94) | Sysops | Coming soon | Several voters agreeing on one order of changes, for an organisation's operations. |
| Monitoring (p. 90) | Sysops | Partial | Know that Mandala is up and healthy — health check, service state, disk and the app's own signals. |
| The Server edition (p. 93) | Sysops | Coming soon | The DHL engine without the Workspace, for running a business's rules on its own servers. |
| Data (p. 39) | Workspace guide | Partial | The back office — every record your rules govern, queries, actions with previews, and the Workbench. |
| Personal (p. 37) | Workspace guide | Partial | Groceries, money and health — lists by aisle, accounts and budgets, measurements and goals. |
| Tasks (p. 39) | Workspace guide | Partial | Every task across your documents, filtered, grouped and scheduled by dragging. |